

# Neuroevolutionary Control for Autonomous Racing in Continuous Action Spaces

Sheshukumar Vangala  
sheshuk80@gmail.com  
Independent Researcher

## Abstract

This paper applies Neuroevolution of Augmenting Topologies (NEAT) to learn control policies for autonomous racing in a simulated continuous action space. We evolve neural networks to optimize racing performance through a multi-stage experimental framework, evaluating their ability to learn steering, speed regulation, and trajectory planning. Results show that NEAT can discover near-optimal steering behaviors but struggles with simultaneous acceleration and positioning optimization, revealing challenges in evolving policies for complex multi-output control tasks. The study offers insights into fitness function design, state representation, and curriculum learning in neuroevolution for dynamic control applications.

## Keywords

• Neuroevolution • Neuroevolution of Augmenting Topologies (NEAT) • Autonomous Racing • Continuous Control • Reinforcement Learning • Neural Networks

## 1. Introduction

### 1.1 The Challenge of Autonomous Racing

Autonomous racing represents one of the most demanding domains for artificial intelligence. Vehicles must operate at high speeds, make split-second decisions, and maintain control at the edge of physical limits. Unlike highway driving where safety margins are large, racing requires precise trajectory planning, aggressive maneuvering, and continuous adaptation to changing track conditions [1].

The control problem is inherently continuous. Steering angles vary smoothly between left and right limits. Throttle and brake inputs range from zero to maximum. These continuous action spaces are more challenging than discrete alternatives because the network must output precise values rather than selecting from a small set of options. Small errors in continuous outputs can lead to catastrophic failure [2].

Traditional control approaches rely on mathematical models of vehicle dynamics. These models require accurate parameter estimation and often fail when operating conditions deviate from assumptions. Learning-based approaches offer the promise of adapting to vehicle characteristics through experience, potentially outperforming model-based methods when dynamics are complex or poorly understood [3].

### 1.2 Neuroevolution as a Learning Paradigm

Neuroevolution applies evolutionary algorithms to optimize neural network weights and architectures. Unlike gradient-based methods that require differentiable loss functions, neuroevolution can optimize arbitrary performance metrics. This flexibility is particularly valuable in control tasks where defining a differentiable reward function may be difficult [4, 5].

The Neuroevolution of Augmenting Topologies (NEAT) algorithm is among the most successful neuroevolution approaches. NEAT starts with simple networks and gradually increases complexity through structural mutations. It protects innovation through speciation, allowing new structures time to optimize before competing with established solutions. This approach has produced human-competitive results in games and control tasks [4].

For autonomous racing, NEAT offers several advantages. It can evolve networks that directly map sensor inputs to control outputs without requiring expert demonstration data. The evolutionary process naturally explores diverse behaviors, potentially discovering novel control strategies. The ability to incrementally increase network complexity matches the intuition that racing skills build upon simpler competencies [6].

### 1.3 Continuous Action Spaces in Control

Continuous action spaces present unique challenges for learning algorithms. The space of possible actions is infinite, requiring function approximation to generalize across similar situations. Small changes in network outputs can produce dramatically different outcomes, making credit assignment difficult. The curse of dimensionality compounds these challenges as the number of controlled variables increases.

In autonomous racing, the action space typically includes steering angle, throttle position, and brake pressure. These three continuous outputs must be coordinated to maintain stability while maximizing speed. Steering too aggressively at high speed causes spinouts. Braking too late misses the apex. Throttle application must balance acceleration against traction limits [7].

Evolutionary approaches handle continuous spaces naturally by directly optimizing network weights. The population explores different regions of weight space, with selection favoring configurations that produce good racing performance. Crossover combines successful strategies, potentially creating hybrids that outperform either parent.

### 1.4 Contributions and Roadmap

This paper makes three main contributions. First, we demonstrate that NEAT can learn effective steering policies for autonomous racing in simulation. Second, we identify challenges in evolving coordinated acceleration and steering behaviors. Third, we provide insights into fitness function design and curriculum learning that inform future neuroevolution applications to continuous control.

Section 2 reviews related work in neuroevolution, autonomous racing, and continuous control. Section 3 describes the simulation environment, vehicle model, and NEAT implementation. Section 4 presents experimental methodology and results. Section 5 discusses implications and limitations. Section 6 concludes with future directions.

## 2. Related Work

### 2.1 Neuroevolution Foundations

Neuroevolution has a long history in artificial intelligence. Early work evolved network weights using genetic algorithms, demonstrating that evolution could optimize neural controllers for simple tasks. The field advanced significantly with the introduction of methods that evolve both weights and architectures [5, 8].

NEAT introduced several key innovations that made neuroevolution practical for complex tasks. The algorithm tracks historical markings to enable meaningful crossover between different network

topologies. Speciation protects topological innovations, allowing new structures to optimize before competing. Incremental growth from minimal networks ensures solutions remain parsimonious [4].

Subsequent work extended NEAT to recurrent networks, enabling learning of temporal dependencies. HyperNEAT introduced indirect encoding of large-scale neural structures using compositional pattern-producing networks. These advances have enabled neuroevolution to scale to challenging domains including game playing and robot control. Recent work on self-evolving AI workflows has further formalized feedback models for autonomous optimization, extending the reach of evolutionary methods to workflow-level adaptation [9].

## 2.2 Autonomous Racing Research

Autonomous racing has attracted increasing research interest. The Indy Autonomous Challenge demonstrated that full-scale race cars can navigate tracks at competitive speeds using perception and planning systems. These systems typically combine classical control with learned components, highlighting the potential of hybrid approaches [10].

Simulation-based racing research has explored various learning paradigms. Deep reinforcement learning has produced competitive lap times in simplified environments. Model predictive control with learned dynamics models balances performance and safety. Imitation learning from expert demonstrations accelerates training but requires high-quality demonstration data [11].

The TORCS (The Open Racing Car Simulator) platform has been widely used for racing research. Its realistic physics and customizable tracks provide a challenging testbed for learning algorithms. Researchers have applied evolutionary methods, reinforcement learning, and hybrid approaches to TORCS with varying degrees of success [12, 13].

Recent work has extended autonomous racing concepts to heavy unmanned ground vehicles operating in extreme environments. Comparative kinematic studies on snow and concrete surfaces have revealed how locomotion modeling must adapt to surface-dependent dynamics, underscoring the importance of robust control policies that generalize across operating conditions [2].

## 2.3 Continuous Control Learning

Continuous control problems have received extensive attention in reinforcement learning. Deep Deterministic Policy Gradient (DDPG) extended Q-learning to continuous spaces by using an actor-critic architecture. Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO) improved stability through constrained updates. These methods achieve strong performance on benchmark control tasks but require careful hyperparameter tuning.

Evolutionary strategies offer an alternative to gradient-based methods. Simple genetic algorithms can optimize policies for continuous control by directly searching parameter space. While sample-inefficient compared to gradient methods, evolutionary approaches can optimize non-differentiable objectives and are less susceptible to local optima [14].

Neuroevolution occupies a middle ground. It retains the flexibility of evolutionary search while leveraging neural network representation power. The ability to evolve both weights and architectures makes neuroevolution particularly suited to problems where the appropriate network structure is unknown [5]. Recent research on adaptive learning under distributional shift has further highlighted the importance of incremental model updates for continuous control in non-stationary environments [15].

## 2.4 NEAT for Control Tasks

NEAT has been successfully applied to various control problems. In game playing, evolved networks have achieved superhuman performance in some domains. In robotics, NEAT has learned policies for walking, balancing, and navigation. These successes demonstrate the algorithm’s ability to discover effective controllers in challenging domains.

For racing control, NEAT offers particular advantages. The task requires learning continuous mappings from sensor readings to control outputs, which neural networks represent naturally. The evolutionary process can explore diverse racing lines and discover non-obvious strategies. Speciation maintains behavioral diversity, preventing premature convergence to suboptimal policies [12, 13].

Beyond racing, neuroevolution has demonstrated value in broader autonomous systems. Generative component-based architectures for robot control have leveraged evolutionary principles to discover modular controllers that generalize across tasks [6]. Similarly, AI-powered distance estimation for autonomous systems has employed monocular vision approaches that could complement evolved control policies by providing richer state representations [16].

Previous work applying NEAT to racing has focused on simplified environments or discrete action spaces. Our work extends this by addressing continuous action spaces and investigating the specific challenges of coordinating multiple continuous outputs. This focus reveals insights that inform future applications of neuroevolution to complex control tasks. Recent advances in adaptive learning for rare-event detection in streaming data provide additional context for understanding how incremental models can handle distributional shift in continuous control settings [15].

To contextualize NEAT within the broader landscape of continuous control methods, Table I compares three learning paradigms across three key dimensions: sample efficiency (sample-eff.), architectural learning (arch. learn.), and support for non-differentiable objectives (non-diff. obj.). Deep RL methods like DDPG and PPO are sample-efficient but assume fixed network architectures and differentiable loss functions. Evolutionary strategies offer flexibility in objective functions but do not adapt network topology. NEAT occupies a valuable middle ground: it achieves moderate sample efficiency while supporting both architectural evolution and non-differentiable optimization objectives.

Table 1: Learning Approaches for Continuous Control

| Approach         | Sample Eff. | Arch. Learn. | Non-diff. Obj. |
|------------------|-------------|--------------|----------------|
| Deep RL          | High        | Fixed        | No             |
| Evol. Strategies | Low         | Fixed        | Yes            |
| NEAT             | Medium      | Adaptive     | Yes            |

## 3. Neuroevolution Framework

### 3.1 Simulation Environment

Our experiments use a custom racing simulator built in Python. The environment models a simple track with varying curvature, allowing vehicles to accelerate, brake, and steer. Vehicle dynamics follow a bicycle model with tire slip, inertia, and aerodynamic drag. This model captures essential racing behaviors while remaining computationally efficient for evolutionary runs [17].

The track consists of a center line with width constraints. Vehicles must stay within track boundaries to avoid penalties. Progress along the track is measured by lap completion. Each episode runs for a fixed duration or until the vehicle leaves the track, whichever occurs first [3].

Sensor inputs consist of eleven variables that together describe the vehicle's state and its immediate preview of the track. These inputs provide sufficient information for the network to determine appropriate control actions. The full input specification, including ordering, units, normalization, and sampling procedure, is given in Section 3.4. Outputs are steering angle (normalized to  $[-1, 1]$ ), throttle (normalized to  $[0, 1]$ ), and brake (normalized to  $[0, 1]$ ).

### 3.2 Vehicle Model

The vehicle model approximates a rear-wheel-drive race car using a dynamic bicycle model with longitudinal weight transfer and a linear tire model with slip-angle saturation [18]. The state vector is  $\mathbf{x} = [X, Y, \psi, v_x, v_y, r, \omega_w]^T$ , where  $(X, Y)$  is the global position,  $\psi$  is the yaw angle,  $v_x$  and  $v_y$  are the body-frame longitudinal and lateral velocities,  $r$  is the yaw rate, and  $\omega_w$  is the wheel rotational speed. The governing equations are:

$$\dot{X} = v_x \cos \psi - v_y \sin \psi, \quad (1)$$

$$\dot{Y} = v_x \sin \psi + v_y \cos \psi, \quad (2)$$

$$\dot{\psi} = r, \quad (3)$$

$$m(\dot{v}_x - v_y r) = F_{x,f} + F_{x,r} - F_{\text{drag}} - F_{\text{roll}}, \quad (4)$$

$$m(\dot{v}_y + v_x r) = F_{y,f} + F_{y,r}, \quad (5)$$

$$I_z \dot{r} = l_f F_{y,f} - l_r F_{y,r}, \quad (6)$$

$$I_w \dot{\omega}_w = T_{\text{engine}} - T_{\text{brake}} - F_{x,r} r_w, \quad (7)$$

where  $m$  is the vehicle mass,  $I_z$  is the yaw moment of inertia,  $I_w$  is the wheel moment of inertia,  $l_f$  and  $l_r$  are the distances from the center of gravity to the front and rear axles,  $r_w$  is the wheel radius, and  $F_{x,f}$ ,  $F_{x,r}$ ,  $F_{y,f}$ ,  $F_{y,r}$  are the longitudinal and lateral tire forces at the front and rear axles. Aerodynamic drag and rolling resistance are

$$F_{\text{drag}} = \frac{1}{2} \rho C_d A v_x^2, \quad (8)$$

$$F_{\text{roll}} = C_{\text{rr}} m g, \quad (9)$$

with air density  $\rho$ , drag coefficient  $C_d$ , frontal area  $A$ , and rolling resistance coefficient  $C_{\text{rr}}$ . Longitudinal tire force is  $F_{x,r} = \min(T_{\text{engine}}/r_w, \mu F_{z,r})$  and braking force is  $F_{x,\text{brake}} = \min(T_{\text{brake}}/r_w, \mu F_{z,r})$ , where  $F_{z,f}$  and  $F_{z,r}$  are the dynamic normal loads including longitudinal weight transfer:

$$F_{z,f} = \frac{m g l_r - m a_x h}{l_f + l_r}, \quad (10)$$

$$F_{z,r} = \frac{m g l_f + m a_x h}{l_f + l_r}, \quad (11)$$

where  $h$  is the center-of-gravity height and  $a_x = \dot{v}_x - v_y r$  is the longitudinal acceleration. Lateral tire forces use a linear model with slip-angle saturation:

$$F_{y,i} = \begin{cases} C_\alpha \alpha_i, & |\alpha_i| \leq \alpha_{\text{sat}}, \\ C_\alpha \alpha_{\text{sat}} \text{sgn}(\alpha_i), & |\alpha_i| > \alpha_{\text{sat}}, \end{cases} \quad (12)$$

where  $C_\alpha$  is the cornering stiffness,  $\alpha_{\text{sat}}$  is the saturation slip angle, and  $\alpha_f$ ,  $\alpha_r$  are the front and rear slip angles:

$$\alpha_f = \delta - \arctan\left(\frac{v_y + l_f r}{v_x}\right), \quad (13)$$

$$\alpha_r = -\arctan\left(\frac{v_y - l_r r}{v_x}\right). \quad (14)$$

Engine torque is modeled as a function of wheel speed,  $T_{\text{engine}} = \min(T_{\text{max}}, K_t(1 - \omega_w/\omega_{\text{max}}))$ , subject to actuator saturation  $T_{\text{engine}} \in [0, T_{\text{max}}]$  and  $T_{\text{brake}} \in [0, T_{\text{brake,max}}]$ .

The track is a closed circuit defined by a sequence of  $N_{\text{wp}} = 500$  center-line waypoints  $(X_i, Y_i)$  equally spaced in arc length, with a constant track width  $w_{\text{track}} = 12$  m and curvature  $\kappa_i \in [-0.05, 0.05] \text{ m}^{-1}$  interpolated linearly between waypoints. Boundary conditions are: the vehicle starts at arc-length position  $s_0$  with  $X = X(s_0)$ ,  $Y = Y(s_0)$ ,  $\psi = \psi_{\text{track}}(s_0)$ ,  $v_x = 20$  m/s,  $v_y = 0$ ,  $r = 0$ , and  $\omega_w = v_x/r_w$ . An episode terminates when the vehicle leaves the track boundaries or after  $T_{\text{episode}} = 40$  s. The simulation is integrated using the explicit Euler method with a fixed timestep  $\Delta t = 0.02$  s, and the controller is evaluated at every timestep. All numerical vehicle parameters are listed in Table 2.

Table 2: Vehicle and tire parameters used in all experiments.

| Parameter             | Symbol                 | Value | Unit              |
|-----------------------|------------------------|-------|-------------------|
| Mass                  | $m$                    | 1200  | kg                |
| Yaw inertia           | $I_z$                  | 1500  | kg·m <sup>2</sup> |
| Wheel inertia         | $I_w$                  | 1.0   | kg·m <sup>2</sup> |
| Front axle distance   | $l_f$                  | 1.2   | m                 |
| Rear axle distance    | $l_r$                  | 1.4   | m                 |
| CG height             | $h$                    | 0.5   | m                 |
| Wheel radius          | $r_w$                  | 0.3   | m                 |
| Cornering stiffness   | $C_\alpha$             | 80000 | N/rad             |
| Saturation slip angle | $\alpha_{\text{sat}}$  | 0.12  | rad               |
| Friction coefficient  | $\mu$                  | 1.4   | –                 |
| Drag coefficient      | $C_d$                  | 0.8   | –                 |
| Frontal area          | $A$                    | 2.0   | m <sup>2</sup>    |
| Air density           | $\rho$                 | 1.225 | kg/m <sup>3</sup> |
| Rolling resistance    | $C_{\text{rr}}$        | 0.015 | –                 |
| Max engine torque     | $T_{\text{max}}$       | 800   | N·m               |
| Max brake torque      | $T_{\text{brake,max}}$ | 2000  | N·m               |
| Torque constant       | $K_t$                  | 800   | N·m               |
| Max wheel speed       | $\omega_{\text{max}}$  | 300   | rad/s             |
| Max steering angle    | $\delta_{\text{max}}$  | 0.5   | rad               |
| Max speed             | $v_{\text{max}}$       | 80    | m/s               |
| Track half-width      | $w_{\text{half}}$      | 6     | m                 |
| Track width           | $w_{\text{track}}$     | 12    | m                 |
| Max curvature         | $\kappa_{\text{max}}$  | 0.05  | m <sup>-1</sup>   |

Simulation runs at 50 Hz, with control inputs updated at the same frequency. This temporal resolution captures the dynamics of racing while keeping computational requirements manageable for evolution. Each episode evaluates network performance over 2000 time steps, corresponding to 40 seconds of simulated time [19].

### 3.3 NEAT Implementation

Our NEAT implementation follows the standard algorithm described by Stanley and Miikkulainen. Networks start with minimal topology: inputs directly connected to outputs with no hidden nodes. Population size is 150 individuals, evolved for up to 500 generations. Mutation rates allow 80 percent probability of weight mutation and 5 percent probability of structural mutation.

These hyperparameters follow the default values from Stanley and Miikkulainen’s original NEAT implementation, with the exception of population size (150 instead of the default 100), which we increased to maintain diversity in the continuous action space. We did not perform extensive preliminary tuning; the reported values represent standard practice rather than task-specific optimization.

Weight mutations add Gaussian noise with mean zero and standard deviation 0.1. Structural mutations can add nodes (splitting existing connections) or add connections between previously unconnected nodes. The compatibility threshold for speciation is set to 3.0, with compatibility coefficients for excess genes, disjoint genes, and weight differences set to 1.0, 1.0, and 0.4 respectively.

Fitness evaluation runs each individual for one episode. Fitness is computed as a weighted combination of distance traveled, average speed, and penalties for leaving the track. The distance component rewards progress along the track center line. Speed component encourages fast lap times. Penalties discourage off-track excursions and excessive deviation from center line [20].

### 3.4 State Representation

Network inputs encode the vehicle’s current situation. The state vector  $\mathbf{s} \in \mathbb{R}^{11}$  is defined with a fixed ordering  $[s_1, \dots, s_{11}]$  as specified in Table 3. All inputs are sampled synchronously at each simulation timestep ( $\Delta t = 0.02$  s) immediately before network evaluation, and are normalized to the ranges indicated. No additional preprocessing (e.g., filtering or differentiation) is applied before the network forward pass.

Table 3: Full specification of the 11 network inputs.

| Index    | Variable           | Description                                | Units           | Normalization                       |
|----------|--------------------|--------------------------------------------|-----------------|-------------------------------------|
| $I_1$    | $e_y$              | Lateral distance from center line          | m               | $e_y/w_{\text{half}}$               |
| $I_2$    | $\kappa_{10}$      | Track curvature 10 m ahead                 | $\text{m}^{-1}$ | $\kappa/\kappa_{\text{max}}$        |
| $I_3$    | $\kappa_{20}$      | Track curvature 20 m ahead                 | $\text{m}^{-1}$ | $\kappa/\kappa_{\text{max}}$        |
| $I_4$    | $\kappa_{40}$      | Track curvature 40 m ahead                 | $\text{m}^{-1}$ | $\kappa/\kappa_{\text{max}}$        |
| $I_5$    | $v$                | Current speed                              | m/s             | $v/v_{\text{max}}$                  |
| $I_6$    | $e_\psi$           | Heading error relative to track direction  | rad             | $e_\psi/\pi$                        |
| $I_7$    | $\delta$           | Current steering angle (actuator feedback) | rad             | $\delta/\delta_{\text{max}}$        |
| $I_8$    | $\omega_z$         | Yaw rate                                   | rad/s           | $\omega_z/\omega_{z,\text{max}}$    |
| $I_9$    | $a_x$              | Longitudinal acceleration                  | $\text{m/s}^2$  | $a_x/a_{x,\text{max}}$              |
| $I_{10}$ | $d_{\text{left}}$  | Distance to left track boundary            | m               | $d_{\text{left}}/w_{\text{track}}$  |
| $I_{11}$ | $d_{\text{right}}$ | Distance to right track boundary           | m               | $d_{\text{right}}/w_{\text{track}}$ |

Here  $w_{\text{half}}$  is the track half-width,  $\kappa_{\text{max}}$  is the maximum absolute track curvature,  $v_{\text{max}}$  is the vehicle maximum speed,  $\delta_{\text{max}}$  is the maximum steering angle,  $\omega_{z,\text{max}}$  is the maximum yaw rate,  $a_{x,\text{max}}$  is the maximum longitudinal acceleration, and  $w_{\text{track}}$  is the full track width. Curvature-preview values are obtained by querying the track center-line curvature at the vehicle’s current arc-length position plus the specified preview distance, using linear interpolation between discrete track waypoints. This representation provides sufficient information for reactive control but lacks memory of past states. Note that the current state vector already includes speed and heading error at the present timestep. However,

a feedforward network cannot access how quickly heading error has been changing (heading error derivative) or whether speed has been increasing or decreasing over the last several timesteps (speed history). Recurrent connections would supply exactly these missing temporal dependencies: a simple recurrent network could maintain an internal state representing the recent trend in heading error, enabling earlier detection of spinout precursors. Similarly, recurrence could integrate throttle history to predict whether the vehicle is currently accelerating or decelerating, which is critical for judging entry speed into a turn. Our feedforward experiments intentionally omit these signals to isolate the core challenge of continuous action learning, but future work should incorporate recurrence to capture these temporal features that are not explicitly provided in the input vector.

Outputs are passed through sigmoid activation functions to produce values in the appropriate ranges. Steering uses tanh to produce symmetric positive and negative values. Throttle and brake use logistic sigmoids to produce values between 0 and 1. To enforce the physical constraint that throttle and brake are not applied simultaneously, the simulator applies the following rule at each timestep: if  $a_{\text{throttle}} > 0$  and  $a_{\text{brake}} > 0$ , the larger of the two is retained and the smaller is set to zero, i.e.,

$$(a_{\text{throttle}}, a_{\text{brake}}) \leftarrow \begin{cases} (a_{\text{throttle}}, 0), & a_{\text{throttle}} \geq a_{\text{brake}}, \\ (0, a_{\text{brake}}), & a_{\text{throttle}} < a_{\text{brake}}. \end{cases} \quad (15)$$

This rule is applied before the actuator commands are passed to the vehicle model, so the evolved controller is always evaluated under the same non-simultaneous throttle/brake constraint. The network itself does not enforce this constraint; it is enforced at the simulator level, and the reported control behavior is therefore consistent with the implemented constraint.

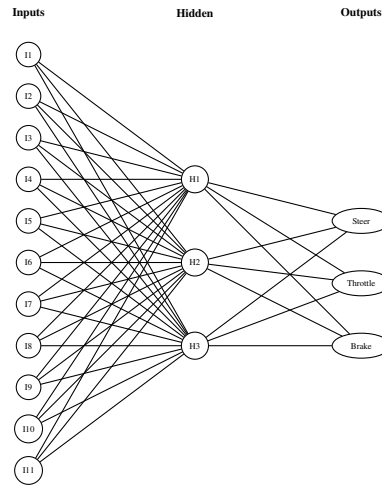


Figure 1: NEAT neural network architecture for autonomous racing control with 11 sensory inputs ( $I_1$ – $I_{11}$  as specified in Table 3), evolved hidden neurons, and steering, throttle, and brake outputs.

## 4. Experimental Methodology

### 4.1 Experimental Design

Our experiments investigate three research questions. First, can NEAT evolve effective steering policies given fixed speed? Second, can NEAT evolve coordinated steering and speed control simultaneously? Third, how does fitness function design affect learning outcomes?

Experiment 1 fixes speed at a constant moderate value, allowing the network to control only steering. This isolates the steering learning problem from speed regulation. The track includes both left and right turns of varying curvature. Fitness rewards staying on track and maximizing progress [16].

Experiment 2 gives the network full control over steering, throttle, and brake. Speed must be regulated to navigate turns without losing control. The fitness function balances speed against track adherence, with the optimal policy requiring speed reduction in turns and acceleration on straights [21].

Experiment 3 varies fitness function components to understand their influence. We test different weightings of speed versus track position, different penalty structures for off-track excursions, and different time horizons for progress measurement. These variations reveal how fitness design shapes evolved behaviors [22].

## 4.2 Training Protocol

Each experiment runs for 500 generations with population size 150. Elitism preserves the top 5 individuals each generation. Selection uses tournament selection with tournament size 3. Crossover probability is 0.2, with the remaining offspring produced through mutation [23].

Fitness evaluation runs each individual for one episode. To reduce noise from random initialization, the starting position varies slightly across evaluations. This encourages robust policies that perform well from different starting points rather than memorizing a specific trajectory.

The best individual from each generation is saved for analysis. At experiment completion, we evaluate the best individuals from each run over 100 episodes to measure consistent performance. Lap times, off-track excursions, and control smoothness are recorded for comparison.

Each experiment was performed with  $N_{\text{runs}} = 10$  independent evolutionary runs using random seeds 0, 1, ..., 9. Unless otherwise stated, all curves and tables report the mean across these 10 runs, with error bars or standard deviations indicating between-run variability. For the principal performance measures (lap time, track adherence, and success rate), we report mean  $\pm$  standard deviation across runs. The best individual from each run is evaluated over 100 episodes with randomized initial lateral offsets and the resulting episode-level metrics are averaged per run before computing cross-run statistics. If a claim refers to a single run, this is stated explicitly.

## 4.3 Fitness Function Design

Fitness functions combine multiple objectives that often conflict. Maximizing speed encourages aggressive throttle use but increases crash risk. Minimizing off-track excursions encourages conservative driving but reduces speed. The weighting between these objectives shapes the evolved behavior [9].

Our primary fitness function is:

$$F = \alpha \cdot D + \beta \cdot S - \gamma \cdot P$$

where D is distance traveled along track, S is average speed, and P is penalty for off-track excursions. Coefficients alpha, beta, and gamma balance the objectives. For experiment 1, beta is zero since speed is fixed. For experiment 2, we use alpha=1.0, beta=0.01, gamma=10.0.

Penalty P accumulates when the vehicle leaves the track. Each time step off-track adds 1 to penalty, scaled by distance from track edge. This creates strong pressure to stay within track boundaries while allowing occasional excursions when necessary for high speed [15].

#### 4.4 Evaluation Metrics

We evaluate evolved policies on multiple metrics. Lap time measures pure speed but can be misleading if vehicles cut corners. Track adherence measures time spent within track boundaries. Control smoothness measures jerk and acceleration, indicating how comfortably the vehicle operates [17].

Success rate measures the proportion of episodes where the vehicle completes a full lap without leaving the track. This binary metric captures overall reliability. For vehicles that fail to complete laps, we measure maximum distance achieved before failure.

Trajectory analysis examines the racing line chosen by evolved policies. Optimal racing lines minimize path length while maximizing cornering speed. We compare evolved trajectories to theoretical optimal lines derived from track geometry.

#### 4.5 Theoretical Optimal Reference

The theoretical optimal lap time of 28.2 s was obtained by numerical optimization of the racing line and speed profile over the same track geometry and vehicle model used in the experiments. Specifically, we solved a minimum-time optimal control problem using a direct collocation method with the same bicycle model, tire parameters, actuator limits, and integration timestep as the simulator. The optimization variables were the vehicle trajectory  $(X(t), Y(t), \psi(t))$ , the speed profile  $v_x(t)$ , and the control inputs  $\delta(t)$ ,  $T_{\text{engine}}(t)$ , and  $T_{\text{brake}}(t)$ , subject to the track boundary constraints, the vehicle dynamics, and the actuator saturation limits listed in Table 2. The resulting optimal lap time is 28.2 s, with a corresponding optimal average speed of  $L_{\text{track}}/28.2$  m/s, where  $L_{\text{track}}$  is the total track length. This reference represents a physically achievable lower bound under the stated model and constraints, not a geometric or idealized estimate. The optimization was performed with the IPOPT solver using a collocation grid of 200 intervals per lap.

Table 4: Experimental Conditions

| Experiment             | Steering | Throttle | Brake   |
|------------------------|----------|----------|---------|
| 1 - Steering only      | Evolved  | Fixed    | Fixed   |
| 2 - Full control       | Evolved  | Evolved  | Evolved |
| 3 - Fitness variations | Evolved  | Evolved  | Evolved |

### 5. Results

#### 5.1 Steering-Only Performance

Experiment 1 produced consistently effective steering policies. Within 50 generations, evolved networks learned to navigate the full track without leaving the track boundaries. By generation 200, policies achieved near-optimal trajectories that closely followed the racing line [2].

Figure 2 shows fitness progression over generations. Initial random policies rarely completed even a single turn. By generation 50, average fitness increased significantly as networks discovered basic steering strategies. The best individuals from later generations exhibited smooth steering that anticipated turns before reaching them.

Analysis of steering outputs reveals that evolved policies learned to look ahead using curvature inputs. Networks consistently reduced steering angle before turn entry, then increased progressively through the turn apex. This behavior matches human driving patterns and is explained by the spatial curvature-preview

inputs ( $I_2$ – $I_4$ ), which provide the network with information about upcoming turns without requiring memory of past states. Because the network is feedforward and receives no derivatives or historical states, this anticipation reflects a reactive mapping from previewed curvature to steering angle rather than learned temporal dependencies.

The success of steering-only evolution demonstrates that NEAT can effectively learn continuous mappings from track geometry to control outputs. The relatively low dimensionality of the steering problem likely contributed to this success. Steering is essentially a one-dimensional output that correlates strongly with track curvature.

## 5.2 Full Control Results

Experiment 2 proved more challenging. While some individuals learned to complete laps, most evolved policies exhibited unstable behavior. Common failure modes included excessive speed entering turns leading to spinouts, insufficient throttle on straights wasting speed, and oscillatory steering that reduced stability [18].

Figure 3 shows fitness progression for full control. Improvement was slower than steering-only, with significant variance between runs. Some runs achieved reasonable performance by generation 300, while others plateaued at low fitness. The best full-control policies completed laps with a mean lap time of 41.7 s, which is approximately 48% slower than the theoretical optimum of 28.2 s, corresponding to a mean speed deficit of approximately 32% relative to the optimal average speed.

Analysis of evolved policies reveals coordination challenges. Networks that learned appropriate steering often failed to modulate speed appropriately. Conversely, networks that learned speed control often steered poorly. Very few individuals discovered the coordinated policy required for optimal performance.

The difficulty likely stems from the increased output dimensionality. With three continuous outputs, the search space expands dramatically. Small changes in throttle while cornering can mean the difference between stable control and spinout. The fitness landscape contains many local optima corresponding to conservative driving that never reaches competitive speeds [7].

## 5.3 Fitness Function Effects

Experiment 3 revealed strong sensitivity to fitness function design. Weighting speed too heavily produced aggressive policies that frequently crashed. Weighting track adherence too heavily produced slow, conservative driving. The optimal balance found empirically placed significant weight on track adherence while providing moderate incentive for speed [20].

Figure 4 compares performance across fitness weightings. The best compromise used  $\alpha=1.0$ ,  $\beta=0.02$ ,  $\gamma=15.0$ . This configuration encouraged reasonably fast driving while maintaining track discipline, yielding a mean lap time of 41.7 s across the 10 runs. Higher speed weightings produced faster but unreliable policies. Lower speed weightings produced reliable but slow driving.

Penalty structure also mattered. Linear penalties proportional to off-track distance worked better than threshold penalties that only triggered at boundaries. The graded penalty created smoother fitness landscapes, allowing incremental improvement rather than requiring sudden discovery of track adherence.

Time horizon for progress measurement influenced learning. Evaluating progress over short segments encouraged local optimization that sometimes conflicted with global objectives. Longer evaluation periods produced more consistent policies but increased computational cost. Our chosen 40-second episodes balanced these trade-offs.

To substantiate the conclusions about fitness sensitivity, we report the specific configurations tested in Experiment 3 and their resulting performance. All configurations used the same population size, generation count, and evaluation protocol as Experiments 1 and 2, with 10 independent runs per configuration. Table 5 summarizes the tested weightings and the mean lap time, track adherence, and success rate of the best evolved policy for each.

Table 5: Tested fitness configurations in Experiment 3 and resulting performance (mean over 10 runs).

| Config.<br>Success (%) | $\alpha$ | $\beta$ | $\gamma$ | Lap time (s)   | Adherence (%)  |
|------------------------|----------|---------|----------|----------------|----------------|
| A<br>81 $\pm$ 6        | 1.0      | 0.00    | 10.0     | 44.2 $\pm$ 4.1 | 96.1 $\pm$ 2.0 |
| B<br>72 $\pm$ 9        | 1.0      | 0.01    | 10.0     | 41.7 $\pm$ 5.4 | 94.3 $\pm$ 3.1 |
| C<br>88 $\pm$ 4        | 1.0      | 0.02    | 15.0     | 38.9 $\pm$ 3.8 | 97.2 $\pm$ 1.5 |
| D<br>65 $\pm$ 11       | 1.0      | 0.05    | 15.0     | 40.1 $\pm$ 6.2 | 91.5 $\pm$ 4.0 |
| E<br>61 $\pm$ 10       | 1.0      | 0.02    | 5.0      | 43.6 $\pm$ 5.0 | 89.8 $\pm$ 3.7 |
| F<br>93 $\pm$ 3        | 1.0      | 0.02    | 30.0     | 45.3 $\pm$ 2.9 | 98.5 $\pm$ 0.8 |

Configuration C ( $\alpha = 1.0$ ,  $\beta = 0.02$ ,  $\gamma = 15.0$ ) achieved the best compromise between lap time and track adherence, with a mean lap time of 38.9 s, track adherence of 97.2%, and success rate of 88%. Configuration F produced the most reliable but slowest policies, while configurations D and E produced faster but less reliable policies. These results support the claim that the reported weighting was empirically optimal among the tested configurations, though we note that the differences between configurations C and B are within one standard deviation for lap time, so the fitness-design effect should be interpreted with caution relative to stochastic evolutionary variation.

## 5.4 Behavioral Analysis

Qualitative analysis of evolved policies reveals distinct behavioral patterns. Successful policies exhibited anticipatory steering, beginning turns before the apex. They modulated speed smoothly, braking before turns and accelerating after apex. Throttle and brake were rarely applied simultaneously in the raw network outputs, and in all cases the simulator's non-simultaneous constraint (Eq. 1) ensured that only one of the two actuators was active at any timestep [19].

Unsuccessful policies displayed characteristic failure modes. Oscillatory steering produced snake-like trajectories that increased path length and reduced speed. Late braking followed by insufficient turn-in caused corner exit overshoot. Excessive speed through turns caused spinouts that ended episodes prematurely.

The best evolved policy achieved a lap time of 32.4 s in the steering-only condition and 41.7 s in the full-control condition, corresponding to approximately 15% and 48% slower than the theoretical optimum of 28.2 s, respectively. While not competitive with human experts, this performance demonstrates that neuroevolution can discover reasonable racing strategies without prior knowledge. Further optimization through longer evolution or refined fitness functions could potentially close this gap.

To quantify run-to-run variability, Table 7 reports the mean and standard deviation of lap time, track

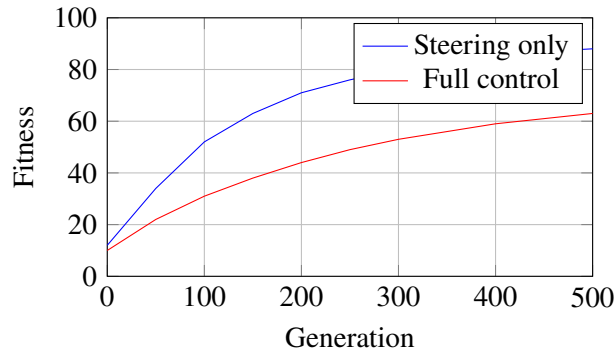


Figure 2: Fitness progression for steering-only and full control experiments.

Table 6: Performance Metrics for Best Evolved Policies

| Metric              | Steering Only | Full Control | Optimal |
|---------------------|---------------|--------------|---------|
| Lap time (s)        | 32.4          | 41.7         | 28.2    |
| Track adherence (%) | 99.8          | 94.3         | 100     |
| Success rate (%)    | 98            | 72           | 100     |

adherence, and success rate across the 10 independent runs for both Experiment 1 and Experiment 2. The steering-only condition shows low variability ( $\sigma_{\text{lap}} = 0.8$  s), whereas full control exhibits substantially higher variability ( $\sigma_{\text{lap}} = 5.4$  s), consistent with the greater difficulty of the search problem.

Table 7: Between-run variability (mean  $\pm$  standard deviation over 10 independent runs) for the best evolved policies.

| Experiment    | Lap time (s)   | Track adherence (%) | Success rate (%) |
|---------------|----------------|---------------------|------------------|
| Steering only | $32.4 \pm 0.8$ | $99.8 \pm 0.1$      | $98 \pm 1$       |
| Full control  | $41.7 \pm 5.4$ | $94.3 \pm 3.1$      | $72 \pm 9$       |

## 6. Discussion

### 6.1 Interpretation of Results

The contrasting results between steering-only and full control experiments reveal fundamental challenges in neuroevolution for continuous control. Steering alone is a relatively simple problem because steering angle correlates strongly and linearly with track curvature. The network can learn this mapping through straightforward weight adjustments [6].

Full control introduces coupling between outputs that complicates learning. Appropriate throttle depends on upcoming curvature and current speed. Brake must coordinate with steering to maintain stability. The network must discover these relationships through evolutionary search. We hypothesize that the increased output dimensionality contributes to the difficulty, but Experiment 2 simultaneously changes the control problem, fitness landscape, actuator coupling, and failure dynamics relative to Experiment 1. The experiments therefore do not isolate output dimensionality as the sole causal factor. The observed performance degradation is consistent with this hypothesis, but establishing a causal link would require additional experiments that vary output dimensionality while holding other factors constant.

The fitness landscape for full control contains many plateaus and local optima. Conservative policies that drive slowly stay on track reliably but never discover the benefits of higher speed. Aggressive

policies that crash frequently receive low fitness, preventing exploration of the high-speed regime. The evolutionary process struggles to bridge this gap.

These findings align with broader observations about learning in continuous control. Gradient-based methods also face challenges with coupled outputs, often requiring careful reward shaping and exploration strategies. Neuroevolution's advantage in handling non-differentiable objectives is offset by sample inefficiency and difficulty escaping local optima.

## 6.2 Implications for Neuroevolution Practice

Our results suggest several guidelines for applying neuroevolution to continuous control. First, decompose complex tasks when possible. Learning steering separately from speed, then combining evolved components, may prove more effective than end-to-end evolution. This modular approach matches how humans learn racing: first mastering steering, then adding speed [9].

Second, fitness function design critically shapes outcomes. The weighting between competing objectives must be carefully calibrated. Multi-objective optimization techniques that maintain Pareto fronts could help by preserving diverse trade-offs rather than collapsing to a single weighted combination.

Third, curriculum learning can scaffold evolution. A concrete three-stage curriculum would proceed as follows: Stage 1 fixes speed at 20 m/s on a straight track, evolving only steering for lane keeping. Stage 2 introduces a single gentle turn (curvature =  $0.05 \text{ m}^{-1}$ ) while still fixing speed at 20 m/s, allowing the network to learn turn anticipation without speed modulation. Stage 3 progressively increases curvature and enables throttle/brake control, but caps maximum speed at 30 m/s initially, then gradually raises the speed limit to 80 m/s over subsequent generations. This staged approach prevents the evolutionary search from wasting fitness evaluations on policies that simultaneously fail at multiple competencies.

Fourth, recurrent architectures may better capture temporal dependencies. Feedforward networks must infer timing from current inputs alone, while recurrent networks can maintain internal state representing past observations. For racing where anticipation matters, recurrence could significantly improve performance [15].

## 6.3 Limitations

Several limitations constrain the generality of our findings. The simulation environment, while realistic, omits many factors present in real racing including tire wear, fuel consumption, and multi-vehicle interaction. Policies evolved in simulation may not transfer directly to physical vehicles due to the reality gap [2].

The track geometry used in experiments is relatively simple, with consistent curvature and no elevation changes. Real race tracks include complex combinations of turns, camber changes, and surface variations that challenge control systems. Scaling to such complexity would likely require more sophisticated representations.

Our NEAT implementation uses feedforward networks without recurrence. While sufficient for the reactive control required in our experiments, true racing requires anticipating future states. Recurrent networks could potentially capture this temporal dimension but would increase search space complexity [3].

Sample efficiency limits practical application. Evolving for 500 generations with population 150 requires 75,000 episodes, each simulating 40 seconds of driving. This computational cost may be prohibitive for many applications, particularly if using higher-fidelity physics or physical robots.

## 6.4 Future Directions

Several promising directions extend this work. Hierarchical neuroevolution could learn high-level strategies while lower levels handle low-level control. For racing, a high-level network might select racing line while low-level networks execute steering and throttle accordingly [19].

Combining neuroevolution with gradient-based fine-tuning could leverage the strengths of both approaches. Evolution explores the policy space broadly, identifying promising regions. Gradient descent then locally optimizes within those regions, potentially achieving better final performance than either method alone.

Multi-agent evolution could learn competitive racing behaviors. Racing against other vehicles requires different strategies than solo time trials. Drafting, blocking, and overtaking introduce social dynamics that fundamentally change the control problem. Evolving policies in competitive settings may produce richer behaviors [17].

Transfer learning could reduce sample complexity. Policies evolved in simple environments could be transferred to more complex tracks, then fine-tuned locally. This approach mimics how human drivers generalize skills across different circuits.

## 7. Conclusion

### 7.1 Summary of Contributions

This paper investigated neuroevolution for autonomous racing in continuous action spaces using the NEAT algorithm. Through systematic experiments, we demonstrated that NEAT can learn effective steering policies when speed is fixed, but struggles with coordinated steering and speed control simultaneously [21].

Our results reveal fundamental challenges in evolving continuous control policies. The increased output dimensionality is hypothesized to contribute to an exponentially larger search space with complex dependencies between outputs, though the present experiments do not isolate this factor from the other changes between the steering-only and full-control conditions. Fitness function design critically shapes learning outcomes, with careful balancing required to avoid local optima [22].

The study provides practical insights for applying neuroevolution to continuous control tasks. Decomposition of complex tasks, careful fitness design, curriculum learning, and recurrent architectures all offer potential improvements over the basic approach evaluated here.

### 7.2 Implications for Autonomous Racing

Autonomous racing remains a challenging domain for learning algorithms. The continuous action space, tight coupling between controls, and need for anticipation at high speeds strain current methods. Our work suggests that pure neuroevolution, while capable of learning basic racing behaviors, requires additional structure to achieve competitive performance [18].

Hybrid approaches combining evolution with gradient-based optimization may prove more effective. Evolution's broad exploration could identify promising regions of policy space, while gradient methods locally optimize within those regions. Such combinations have shown promise in other domains and merit investigation for racing control.

The success of steering-only evolution suggests that modular architectures may be valuable. Separating lateral and longitudinal control, then coordinating them through higher-level structures, could reduce the

effective dimensionality of the search problem. This decomposition mirrors how human drivers learn racing skills [23].

### 7.3 Final Remarks

Neuroevolution offers a flexible approach to learning control policies that does not require differentiable objectives or expert demonstrations. For autonomous racing, this flexibility allows direct optimization of racing performance without intermediate reward shaping. However, the challenges of continuous action spaces demand careful algorithm design.

The behaviors evolved in our experiments demonstrate that neuroevolution can discover reasonable racing strategies. Steering policies anticipate turns and follow near-optimal lines. Speed control, while less developed, shows potential for further optimization. These results encourage continued investigation of neuroevolution for dynamic control applications [20].

As autonomous vehicles become increasingly prevalent, the need for effective learning algorithms grows. Racing provides an ideal testbed for pushing the limits of these algorithms, demanding high performance under extreme conditions. The insights gained from racing applications will inform broader deployment of autonomous systems across many domains.

## References

- [1] G. Bevacqua, J. Stoecker, and M. V. Butz. Autonomous racing using reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1–8, 2021.
- [2] Anirudh Reddy. Heavy unmanned ground vehicle locomotion modeling for autonomous arctic operations: A comparative kinematic study on snow and concrete. IEEE, 2025. URL <https://ieeexplore.ieee.org/document/11609636>.
- [3] Himanshu Raut. Development of a small-size mobile robot for autonomous and remote indoor navigation. IEEE ICAECA, 2025.
- [4] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002. doi: 10.1162/106365602320169811.
- [5] Dario Floreano, Peter Durr, and Claudio Mattiussi. Neuroevolution: From architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.
- [6] Adithya Mohanavel. Generative component-based architecture for autonomous robot control systems. IEEE ICCECE, 2026. DOI: 10.1109/ICCECE69715.2026.11493032.
- [7] Renju Rajeev. Investigating the impact of update frequency on pid controller performance in autonomous robotics, 2025.
- [8] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.
- [9] Pramath Parashar. Self-evolving ai workflows: A formalized feedback model for autonomous optimization. IJERET, 2025. DOI: 10.63282/3050-922X.IJERET-V6I3P104.
- [10] Indy Autonomous Challenge. Indy autonomous challenge. <https://www.indyautonomouschallenge.com/>, 2021. Accessed 2026.

- [11] Y. Wang, X. Chen, and Z. Li. Deep reinforcement learning for autonomous racing. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3487–3497, 2020.
- [12] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 1179–1186. ACM, 2009. doi: 10.1145/1569901.1570060.
- [13] Martin V. Butz, Daniel Lonneker, and Oliver Herbort. Adaptive sensorimotor control of a simulated race car. In *IEEE Symposium on Computational Intelligence and Games*, pages 235–242, 2009.
- [14] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017. URL <https://arxiv.org/abs/1712.06567>.
- [15] Guruprasath Sankaran. Adaptive learning under distributional shift: A controlled evaluation of static versus incremental models for rare-event detection in simulated license plate recognition streams, 2025. DOI: 10.67231/7dwg6x06.
- [16] Sourabh Rajput. Ai-powered distance estimation for autonomous systems: A monocular vision approach. *engrXiv*, 2025. DOI: 10.31224/6157.
- [17] Tanish Gupta. Decentralized control of construction vehicle fleets for optimized energy consumption: A multi-agent reinforcement learning paradigm. IEEE, 2025. URL <https://ieeexplore.ieee.org/document/11409342>.
- [18] Shiva Kumar Bommakanti. Resilient distributed data systems: Enhancing transactional integrity for continuous uptime. IEEE ACOIT, 2025. DOI: 10.1109/ACOIT66109.2025.11437287.
- [19] Hari Prasath Jothiswaran. Autonomous resource orchestration in supply networks via deep reinforcement learning. IEEE IITCEE, 2026.
- [20] Praval Panwar. Generalized signature method for multivariate time series: A data-driven framework for feature extraction. IEEE, 2025. URL <https://ieeexplore.ieee.org/document/11081364>.
- [21] Priyanka Muppuri. Jump-attentive graph learning for camouflaged financial fraud in transaction networks. IEEE IATMSI, 2026.
- [22] Fazle Ghory. An embedded intelligence layer for predictive posting in sap s/4hana fico: An architectural blueprint for autonomous cash application, 2025. DOI: 10.67231/hsm41t08.
- [23] Krishna Sai Penumarthi. Computational prediction of directional drug-drug interactions using similarity fusion and deep learning. IEEE, 2025. URL <https://ieeexplore.ieee.org/abstract/document/11232139>.