Memetic Computing manuscript No.
(will be inserted by the editor)

GP-RARS: Evolving Controllers for the Robot Auto Racing Simulator

Yehonatan Shichel · Moshe Sipper

Received: June 6, 2011/ Accepted: date

Abstract We use evolutionary computation techniques to create real-time reactive controllers for a race-car simulation game: RARS (Robot Auto Racing Simulator). Using genetic programming to evolve driver controllers, we create highly generalized game-playing agents, able to outperform most human-crafted controllers and all machine-designed ones on a variety of game tracks.

1 Introduction

The domain of games is a computational field wherein Artificial Intelligence (AI) plays an important role. From chess to checkers, Pac-Man to poker, games have always been an AI "playground". Since games are a celebration of human intelligence, cognition, and spirit, they have attracted many AI enthusiasts, aiming to compete with and even beat human players at their own game.

Programming Games are a subset of this domain, in which the games are played by computer programs rather than human players. This class of games introduces an interesting challenge: instead of *being* the optimal player, the goal is to *write* the optimal playing program. Usually, these programs are hand-coded by human programmers; however, in some cases, machine-learning techniques are utilized for the creation or optimization of the controllers.

Y. Shichel · M. Sipper
Department of Computer Science,
Ben-Gurion University, Beer-Sheva, Israel

Y. Shichel
E-mail: shichel@gmail.com

M. Sipper
E-mail: sipper@cs.bgu.ac.il

In this paper we use Genetic Programming (GP) to create a game controller program by means of evolution. As our benchmark we have chosen the game of RARS (Robot Auto Racing Simulator), which is an open-source, car-race simulator. This game was chosen mainly because of its extensive human-written driver library, and the substantive amount of published works that describe machine-learning approaches applied to RARS—enabling us to perform significant comparisons between our results and both human- and machine-designed solutions.

This task is considered a hard problem because race-car control requires a high level of expertise in various game aspects, such as speeding, steering, and race-line planning, and, moreover, the controller should ultimately outperform existing solutions, created by both humans and various AI approaches, as described in Section 2.

This paper is organized as follows: Section 2 describes previous work, while Section 3 introduces the game of RARS. Section 4 presents our approach for evolving RARS controllers, followed by results in Section 5. Finally, Section 6 presents concluding remarks and future work.

2 Previous work

Controlling a moving vehicle is considered a complex problem, both in simulated and real-world environments. Dealing with physical forces, varying road conditions, unexpected opponent behavior, damage control, and many other factors, render the car-racing problem a fertile ground for artificial intelligence research. Below we survey several works on evolving controllers for cars in simulated environments.

Wloch and Bentley [24] used Genetic Algorithms (GAs) to optimize setup parameters, such as tire air pressure, gear change rates, and spring tension, in the "Formula One Challenge '99-'02" simulator; modifying the car setup rather than its controlling software, they were able to show improvements in the overall performance.

Floreano et al. [10] used coevolution of Artificial Neural Networks (ANNs) to develop both image feature selection (filtering an image in order to extract various features) and active vision (selecting parts of the image to focus on) to create a controller for the "CarWorld" open-source simulator.[1] They developed a controller able to complete a lap on several given tracks, relying only on visual inputs, as seen from the driver's seat.

Togelius and Lucas [22] employed various approaches based on GAs and ANNs in order to train a simulated radio-controlled car to drive on simple tracks, in which the controllers possessed complete knowledge of the track structure via a simulated overhead camera. In another, rather unorthodox work, Togelius et al. [23] used GAs to evolve the tracks rather than the controllers, and tried to maximize the "fun factor" for the game players by suggesting tracks that were challenging yet not overly hard so as to cause frustration. (In a similar unorthodox vein, Sipper [19] evolved environments to fit a robot.)

On or about the same time, Chaperot [4] and Chaperot and Fyfe [5] used GAs and ANNs to create motorcycle controllers for the "Motocross—The Force" simulator, which features competitive bike driving across a three-dimensional terrain, including complex rigid-body physics.

Tanev et al. [21] used GAs to optimize the parameters of a real-world, radio-controlled car controller. They demonstrated an increase in performance during the course of evolution, and the emergence of obstacle-avoiding behavior once obstacles were introduced onto the track.

RARS, the Robot Auto Racing Simulator,[2] attracted numerous academic researchers and hobbyists, and was one of the first platforms to enable objective comparison between the performance of controller algorithms, by holding open, online racing competitions on a regular basis. In addition to many controllers hand-coded by hobbyist programmers, various AI techniques were used to create, train, and optimize RARS controllers.

Several researchers used ANNs within the RARS framework: Coulom [7] applied temporal difference reinforcement learning to train an ANN to drive a car around a track, while Pyeatt and Howe [17] trained multiple ANNs to perform low-level tasks—such as driving and overtaking—and a higher-level mechanism to switch between the low-level behaviors.

Sáez et al. [18] and Eleveld [9] used GAs to find an optimal path around a RARS track, a highly effective method for known tracks without stochastic effects, but one that leads to very poor performance on unknown tracks or in nondeterministic situations.

Stanley et al. [20] presented a combined ANN and GA approach, using Neuro-Evolution of Augmenting Topologies (NEAT) to evolve and train a RARS-based collision warning system. This approach combined a conventional ANN training algorithm for the network weights with an evolutionary algorithm that modified their topologies. Although their main focus was on the creation of a collision warning system rather than the development of a fast driver, their evolved controllers were able to complete a lap in good time.

Rule-based solutions, created using reinforcement learning, were suggested by Cleland [6] and led to the creation of rather competitive RARS controllers. Ng et al. [14] trained RARS controllers to imitate the behavior of a "good" human-crafted controller, using the Modular Neuro-Fuzzy (MoNiF) approach—a combination of fuzzy classifying functions, which were used to create discrete input values for artificial neural networks.

TORCS (The Open Race Car Simulator[3]), which is based on RARS, has been gaining popularity over the past couple of years, and several TORCS-related papers have been published. Notable works include the application of fuzzy classification functions to the creation of competitive controllers [15, 16], parameter optimization of a hand-coded controller using an evolutionary strategy [1], and the imitation of successful machine- or human-crafted controllers by using either ANNs [13] or NEAT and k-nearest neighbor classifiers [2].

Ebner and Tiede [8] showed that genetic programming (GP) can be successfully applied to evolving TORCS-playing agents, however, their evolved controllers were not able to compete successfully with manually constructed drivers, and their generalization capabilities were not tested.

Finally, Cardamone et al. [3] used real-time Neuro-Evolution of Augmenting Topologies (rtNEAT) to evolve a TORCS controller from scratch and optimize its performance on unseen tracks during the course of a single game, unlike the usual application of learning techniques, which were applied prior to the race itself.

Some of the above RARS-based works provide the exact lap times of the generated controllers. In Section

[1] http://carworld.sourceforge.net

[2] http://rars.sourceforge.net

[3] http://torcs.sourceforge.net

5, we will inspect these results and compare them with our own.

3 Robot Auto Racing Simulator

RARS is an open-source, car-race simulator, written in C++. It was created by several individual programmers in 1995, and evolved since then into a complex racing system. This game employs a detailed physical engine, including most of the forces relevant to moving cars, such as acceleration and deceleration, frictional factors, and centripetal forces. This game enjoyed a large and lively gamer community, until recently, and RARS tournaments were held regularly between the years 1995 and 2004.

The goal of the game is fairly simple: one or more cars race on a given track. The cars are positioned at the starting line, and simultaneously start moving when the race begins. Cars are damaged upon collision or when driving off the track. When a car reaches the starting line, which also acts as the finishing line, a lap counter is incremented. The winner is the driver whose car finished first a given number of laps.

A RARS controller is a C++ class with a single method, which receives the current race *situation* and determines the desired speed and wheel angle of the car. The simulation engine queries the controller approximately 20 times per "game second", and advances the car according to the returned decisions and physical constraints. The *situation* argument provides the agent (car controller) with detailed information about the current race conditions, such as current speed and direction, road curvature, fuel status, and nearby car positions.

Controlling the car is done by two actuators: speed and steering. The speed actuator specifies the desired speed of the car, while the steering actuator specifies the desired wheel angle. The simulation engine uses both values to calculate the involved physical forces and compute the car's movement. Extreme values, such as high speed or a steep steering angle, may result in slippage or skidding, and must be taken into consideration when crafting a controller.

RARS controllers should be able to perform well on a variety of tracks and scenarios. The basic RARS package contains several simple tracks of various shapes, such as oval, round-rectangular, and figure 8-shaped tracks. In addition, each of the RARS tournaments contains several tracks of higher complexity, which are not included in the basic package. Some of these are replicas of real-world tracks (such as the *Sepang International Circuit*[4]), while others are fictional tracks that were designed by the tournament administrator.

RARS tournament rules divide the game-playing controllers into two classes, differing in a single aspect: pre-computation. Agents of the first class—planning agents—are allowed to *inspect the track prior to the race*, and apply a computational process to the track data. This is usually used to produce a precise driving plan—a series of radii and speeds—according to which the car should drive. The second class of agents—reactive agents—are not given the track plan, and their actions rely only on the road conditions observable by the driver in accordance with its physical position at any given time.

Since pure planning agents do not take stochastic factors (such as nearby cars or random friction factors) into consideration, they are rendered useless in many situations; therefore, most of this class's agents employ some degree of reactive behavior in addition to the pre-calculated driving plan. By and large, planning agents outperform reactive agents, because they are better prepared to handle the road conditions, and their precise knowledge regarding the road curvature for any track segment allows them to be prepared for unexpected road features.

Both problems—reactive driving and optimal path planning—are of interest to the AI community. In this paper we focus on reactive agents.

4 Evolving a race-car controller

We chose to focus on the task of creating purely reactive agents for single-car, single-lap races. In this game variant, each race includes one car, attempting to achieve the best lap time.

We used Koza-style GP [11], in which a population of LISP expressions is evolved (Figure 1). Each agent is controlled by two LISP expressions—one for the speed actuator and the other for the steering actuator. Whenever the controller is queried by the RARS simulation engine, both expressions are evaluated and their results are passed back as the desired speed and steering values.

4.1 Functions and terminals

The LISP expressions are defined over the set of functions and terminals described in Table 1 and Figure 2. These are divided into several groups:

[4] http://malaysiangp.com.my

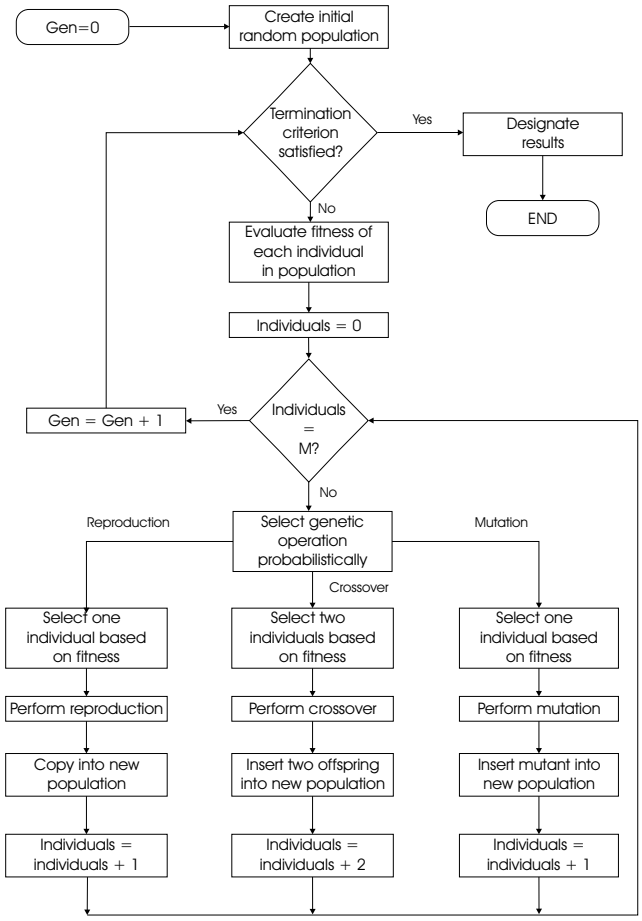


Fig. 1 Generic genetic programming flowchart (based on Koza [11]). M is the population size and Gen is the generation counter. The termination criterion can be the completion of a fixed number of generations or the discovery of a good-enough individual. (Note that the above generic flowchart shows one mutation operator whereas we use two: structural mutation and ERC mutation—see Section 4.3).

– *Basic game status indicators*, which return real-time information regarding the status of the car, as provided by the RARS simulation engine.
– *Complex game status indicators*, which also return real-time information. This indicator set expands the basic set with indicators that are not provided directly by the game engine, but instead are calculated by our software. These indicators, such as *distance to the next obstacle*, require complex trigonometric functions and code loops, which are beyond the complexity capabilities of the GP code model we used, and hence are impossible to develop by means of evolution. These building blocks are actually human-made functions, driven by intuition, and can be very powerful when introduced into the evolutionary process.
– *Numerical constants*, which include the constants 0, 1, and ERC (Ephemeral Random Constant) [11]. An ERC is initialized to a random number during the creation of the initial population, and retains its value throughout the evolutionary run unless modified by mutation.
– *Mathematical functions*.
– *Conditional statements*.

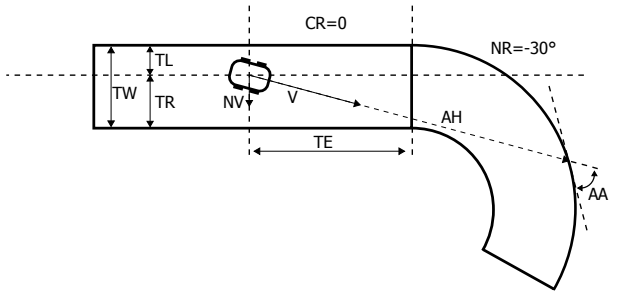


Fig. 2 RARS game indicators (basic and complex).

Some additional technical points:

– Game indicators, both basic and complex, were normalized to fit a common scale.
– Distance values TE, NL, TL, TR, TW, AH, CR, and NR are in feet, divided by 400.
– Velocity values V and NV are in feet per second, divided by 100.
– The angle indicator AA is specified in radians.
– Radii values specify both the radius and the direction of the track segment: positive values indicate a counter-clockwise turn, negative values indicate a clockwise turn, and a value of zero represents a straight track segment.

4.2 Fitness function

The fitness function performs a crucial role in evolution, as it should reflect the quality of the inspected individual. Two aspects were taken into consideration when defining fitness—track selection and fitness-value calculation:

4.2.1 Track selection

The track on which the individuals are evaluated should be as diverse as possible. A homogeneous track (an oval one, for example) might yield specialized agents, which perform well on the given track but show poor performance on other tracks. A heterogeneous track, which contains many distinct features, is likely to yield more generalized drivers, able to drive well on any given track.

We inspected the RARS track library and chose the *Sepang International Circuit*. This track exhibits

Table 1 Functions and terminals used to evolve race cars.

Basic Game Status Indicators	
CR	**C**urrent **R**adius: Radius of current track segment
NR	**N**ext **R**adius: Radius of next track segment
TE	**T**o **E**nd: Distance to end of current track segment
NL	**N**ext **L**ength: Length of next track segment
V	**V**elocity: Current velocity of car
NV	**N**ormal **V**elocity: Drift speed towards road shoulder
TL	**T**o **L**eft: Distance to left road shoulder
TR	**T**o **R**ight: Distance to right road shoulder
TW	**T**rack **W**idth
Complex Game Status Indicators	
AH	**AH**ead: Distance car can move in its current heading without veering off road
AA	**A**head **A**ngle: Angle of road shoulder, relative to car's heading, found by AH terminal
Numerical Constants	
ERC	ephemeral random constant
0	zero constant
1	one constant
Mathematical Functions	
$+(x, y)$	adds x and y
$-(x, y)$	subtracts y from x
$*(x, y)$	multiplies x by y
$\%(x, y)$	'safe-divide' x by y: if $y = 0$, returns 0 otherwise returns the division of x by y
$\text{abs}(x)$	absolute value of x
$\text{neg}(x)$	negative value of x
$\tan(x)$	tangent of x
Conditional Statements	
$\text{IFG}(x, y, \alpha, \beta)$	if $x > y$, returns α, otherwise returns β
$\text{IFP}(x, \alpha, \beta)$	if x is positive, returns α, otherwise returns β

many common track features, such as sharp and moderate curves, U-turns, and straight segments of varying lengths.

4.2.2 Fitness calculation

Two related fitness functions were used in order to measure the quality of a driver: *Race Distance* and *Modified Race Time*:

- *Race Distance* is the distance, in feet, which the car traverses during a 250-game-second period. When this function is used during evolution, the goal is to maximize the fitness value of the individuals.
- *Modified Race Time* is the time, in game seconds, required by the car to complete the race. Because some agents fail to complete a single lap (due to extremely slow driving or suffering a fatal crash—a phenomenon not uncommon in early generations), we amended this simple measure. The modified measure is a comparison-based fitness measure, which does not produce a quantitative fitness value, but instead compares two (or more) individuals and determines the fitter of the lot.[5]
  Such a measure can be used only with comparison-based selection methods, such as tournament selection. Specifically, when comparing two controllers that finished a single lap, the one with the shortest lap time is considered to be fitter. If one of the controllers was not able to complete the lap, it is considered less fit than the one that did finish. If both controllers were not able to finish the lap, the one that traveled the farthest is considered to be the fittest.
  Using Modified Race Time we were able to directly address the challenge at hand—evolving controllers with the shortest lap time—while maintaining a diverse population in early generations, wherein no controller is able to complete a single lap.

4.3 Evolutionary parameters

The evolutionary parameters were carefully chosen through a long calibration process. In this process, various evolutionary runs were executed in an attempt to measure the influence of each evolutionary parameter. The final set of parameters was as follows:

- *Population size*: 250 individuals. Using larger populations did not yield significantly better results, but smaller populations were not able to produce good results.
- *Generation limit*: A value of 255 generations was used. Usually, the population reached an observed peak performance between generations 150 and 200, so best-of-run individuals often emerged before the limit was reached.
- *Selection method*: Tournament of 3 individuals. In this method, the selection of a single individual is done by randomly choosing three individuals, and returning the fittest among them.
  Different tournament group sizes were tested during the calibration process; groups larger than 4 individuals yielded faster convergence to non-optimal solutions, while groups of 2 individuals resulted in slow convergence to non-optimal solutions.
- *Breeding operators*: Breeding operators are used in order to create a new generation from an existing one. The resulting individuals should resemble their parents, yet differ in some characteristics. We used four breeding operators, applied with various probabilities:

[5] We consider this fitness function to be "semi-quantitative" because it does not produce a single numerical value but instead decides which individual of the given candidates is the fittest.

- Reproduction (40%): Selects one individual, using the selection method described above, and passes it onto the next generation as is. Other reproduction probabilities, including no reproduction at all, were tested during the calibration phase. We found that a lower reproduction rate resulted in faster convergence, but not necessarily to optimal solutions. We surmise that a high reproduction rate allowed enough good individuals to move unmodified into the next generation, thus affording the preservation of their properties without incurring the risk of damaging them by mutation or crossover.
- Crossover (50%): Selects two individuals, using the selection method described above, and creates two new individuals by substituting random subtrees between them. Bloat [12] is controlled by setting a tree depth limit of 8 and choosing subtrees such that the resulting trees will not exceed this limit. This operator is used in order to introduce new individuals that are based on existing individuals' characteristics.
- Structural mutation (5%): Randomly selects one individual and creates a new one by choosing a random tree node, discarding its rooted subtree, and growing a new subtree instead. Bloat control is achieved through the same mechanism that is used in crossover. Structural mutation is used in order to introduce variants of existing individuals; however, due to its destructive potential, it is used in small doses.
- ERC mutation (5%): Randomly selects one individual and modifies its ERCs. This is done by randomly choosing an ERC node within the individual and modifying its numerical value. This operator is used to fine-tune the constant values that are used as evolutionary building blocks.

– *Creation of initial population* was done according to Koza's ramped-half-and-half method [11]. For each tree depth between 4 and 8, an equal number of trees is created. Half the trees are grown to the exact given depth, while the other half is grown to a depth *up to* the given depth. Trees are grown using random functions from the function set such that the resulting tree is a legal LISP expression.

5 Results and analysis of an evolved driver

We executed ten evolutionary runs with the Race Distance fitness function, and ten runs with the Modified Race Time fitness function. An individual's fitness was calculated on the Sepang International Circuit. The progress of the two best evolutionary runs is shown in Figures 3 and 4. We extracted one individual from each of these runs: GP-Single-1 (evolved using Race Distance fitness) and GP-Single-2 (evolved using Modified Race Time), both found by performing ten independent races per each individual in the last generation, and choosing the individual with the best average lap time.

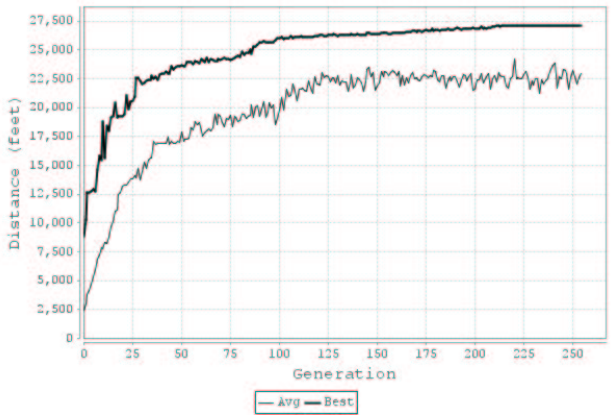


Fig. 3 Fitness vs. Time plot of the best evolutionary run using the Race Distance fitness measure. The thick line denotes the best fitness of each generation, while the thin line denotes the average fitness of the population in each generation.

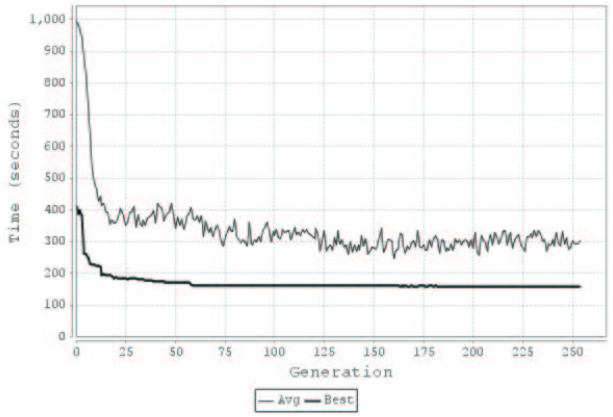


Fig. 4 Fitness vs. Time plot of the best evolutionary run using the Modified Race Time fitness measure. Since this fitness measure does not produce a numerical value (but uses a comparative model instead), it cannot be plotted straightforwardly. Hence, to properly plot this run we used the following method: drivers that were able to complete a single lap were plotted using their lap time, while drivers that were not able to complete a single lap were assigned an arbitrary lap time value of 1000 seconds.

Figure 5 shows the performance of the GP-Single-2 driver on several tracks from the RARS library, clearly exhibiting advanced driving features. The car slows down before curves in proportion to their sharpness, to eliminate the risk of losing control; moreover, the controller attempts to increase the path radius by entering and exiting the curve from the outer shoulders and touching the inner shoulder at mid-curve, thus enabling the car to travel at higher speeds without the risk of skidding.

Table 2 Comparison of evolved drivers on the sepang track. The entire set of reactive human-crafted controllers from the latest RARS distribution was tested. Results vary slightly from race to race due to the simulation's stochastic nature; therefore, each race was performed 100 times per driver, and the results were used to calculate the average timings, as well as standard deviation values and standard error values (in parentheses).

Rank	Driver	Lap Time (seconds)
1	GP-Single-2	159.8 ± 0.6 (std. error: 0.06)
2	Vector	160.9 ± 0.1 (0.01)
-	GP-Single-1	160.9 ± 0.3 (0.03)
4	WappuCar	161.7 ± 0.1 (0.01)
5	Apex8	162.5 ± 0.2 (0.02)
6	Djoefe	163.7 ± 0.1 (0.01)
7	Ali2	163.9 ± 0.1 (0.01)
8	Mafanja	164.3 ± 0.2 (0.02)
9	SBv1r4	165.6 ± 0.1 (0.01)
10	Burns	167.8 ± 5.6 (0.56)
11	Eagle	169.3 ± 0.6 (0.06)
12	Bulle	169.4 ± 0.3 (0.03)
13	Magic	173.9 ± 0.1 (0.01)
14	JR001	178.3 ± 0.2 (0.02)

A comparison with human-crafted reactive drivers on the sepang track is shown in Table 2. Note that lap times vary from race to race due to random factors in the friction coefficient formula aimed at simulating real-world conditions, such as dirt and debris on the race track. Therefore, each race was performed 100 times per driver, and the results were used to calculate the average timings, as well as standard deviation and standard error values. Our top evolved drivers were able to rank first and second out of 14 contestants.

Both evolved drivers exhibit shorter lap times than any human-crafted driver in their class (excluding *Vector*, which shares the second-best result with *GP-Single-1*). However, since many machine-learning techniques tend to prefer specialization over generalization, the performance of our evolved drivers should be checked on tracks other than sepang—which was used for fitness calculation in the evolutionary process. In order to perform such a comparison we evaluated each human-crafted driver along with our own evolved drivers on 16 tracks, taken from the August 2004 RARS tournament. This tournament is the most recent one for which the source code of human-crafted drivers is available online, thus allowing us to compare the results between our drivers and the human-crafted ones. The results are shown in Table 3.

Out of 14 drivers (all but ours designed by humans), the evolved drivers ranked second and third. These results show that the evolved solutions exhibit a high degree of generalization, and are able to successfully solve instances of the problem that were not included in their original training set.

To further inspect the evolved drivers and their generalization capabilities, we tested their performance on the Aug. 2004 season with two scenarios that were not targeted in the training phase: multiple-lap races and multiple-car races. These scenarios require different behaviors than single-lap races, as well as several indicators that are not available to our evolved drivers, such as damage levels and information about nearby cars.

The performance on multiple-lap scenarios was rather poor. Our controllers were unable to complete a single race because they reached a critical damage level and broke down after a few laps. It appears that the evolved drivers gain a small amount of damage per lap—a harmless phenomenon in single-lap races— but after several laps the accumulated damage level reaches a critical level and prevents the drivers from finishing the race. This problem will probably be alleviated by evolving multiple-lap drivers.

Multiple-car scenarios, however, proved surprisingly good, as seen in Table 4. Our controllers reached the first and fourth places, scoring better than Vector—the winner of the single-lap challenge, and Mafanja—the winner of the original Aug. 2004 season. Considering the fact that the evolved controllers do not have information regarding their surrounding cars we conclude that multiple-car behaviors—such as overtaking and collision avoidance—are of less importance in this game, compared to the task of driving as fast as possible. If damage control is not a consideration a brutal drive-through strategy is apparently sufficient for our controllers to gain the leading position, and, once gained, the expected behavior is similar to the single-car scenario behavior. It appears that Vector is less successful in multi-car scenarios (as seen in Table 4), and Mafanja is less successful in the fast-driving challenge (as seen in Table 3), hence GP-Single-2 was able to rank first in this challenge.

Comparison with machine-generated solutions discussed in Section 2 was done by recording the performance of our evolved drivers on each track for which machine-generated results were reported. Table 5 lists the results of the evolved drivers in comparison with other machine-generated solutions.

Again, due to the stochastic nature of the simulation, each race was performed 100 times per driver and average results were noted along with standard deviation and standard error values. However, since we had only the reported results for the machine-generated drivers—rather than an executable version—no statistical information was available for them.

The evolved drivers perform better than any machine-generated reactive solution. Furthermore, the tracks used for these comparisons were *not* included in the training

Table 3 Comparison of evolved drivers with human-crafted drivers on 16 tracks from the August 2004 season, based on 10 races per controller and using the IndyCar points system, wherein the twelve fastest drivers receive 20 points (best driver), 16, 14, 12, 10, 8, 6, 5, 4, 3, 2, and 1 point (worst driver), respectively; in addition the driver that leads the most laps receives an additional bonus point, and the winner of the qualifications round—if one is held—receives an additional point. We held no qualification round and the race consisted of a single lap, hence the fastest driver received 21 points. The total score of each driver is simply the sum of its single race scores. Each driver's rank per race is listed along with its total seasonal score (rightmost 'Total' column).

Rank	Driver	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	Total
1	Vector	1	1	6	5	13	1	8	1	1	1	3	9	3	2	2	2	229
2	GP-Single-2	3	4	10	7	1	3	3	3	2	2	1	5	1	5	11	1	215
3	GP-Single-1	4	3	12	1	5	2	1	9	4	7	4	1	13	4	3	5	186
4	Mafanja	2	5	8	3	2	7	4	7	5	4	6	4	4	3	4	4	177
5	SBv1r4	9	6	11	6	6	4	5	2	3	5	5	2	2	8	9	6	151
6	Eagle	10	2	1	13	11	13	6	8	7	3	2	8	12	1	1	8	144
7	WappuCar	8	7	9	4	8	5	2	5	9	6	7	3	7	11	6	10	119
8	Djoefe	6	10	3	9	4	9	9	4	10	9	9	10	5	6	5	3	117
9	Burns	5	8	7	8	3	8	7	6	6	8	10	6	6	7	7	7	109
10	Magic	11	9	2	2	10	6	10	12	8	11	11	7	10	10	8	11	81
11	Ali2	7	11	4	10	7	11	11	11	11	10	8	11	8	9	10	9	63
12	Apex8	12	12	5	11	9	10	13	10	12	12	12	12	9	12	12	12	35
13	JR001	13	13	13	12	12	12	12	13	13	13	13	13	11	13	13	13	6
14	Bulle	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	0

Table 4 Comparison of evolved drivers with human-crafted drivers on 16 tracks from the Aug. 2004 season, based on 10 races per controller and using the IndyCar points system, on a 3-lap, multiple-car scenario. Each driver's average score per race and total seasonal score is listed. The rightmost 'Orig.' column shows the original score of the Aug. 2004 season.

Rank	Driver	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	Total	Orig.
1	GP-Single-2	11	15.1	2.5	13.4	20.3	17.8	19.5	16.3	16.6	16.7	20.9	17.3	16.9	11.1	6.6	20.4	242.4	-
2	Mafanja	14.9	11.5	13	13	13.2	10.8	12	11.9	10.6	15.6	8.6	13.1	11.5	16.9	12.9	10.3	199.8	236
3	Vector	14.9	16.3	8.3	2.9	0	8.1	9.5	14.6	17.3	14.7	14.2	2.6	13.1	13.2	14.8	6.4	170.9	190
4	GP-Single-1	16.8	12	2	15.6	4.1	14.3	17.1	7.8	10.4	11.2	11	11.1	1.6	11.7	12.4	9.1	168.2	-
5	Djoefe	10.4	5.4	20.1	7.4	13	6.1	6.7	14.6	7.9	6.4	7.5	5.9	13.3	8.5	14.4	15.2	162.8	139
6	Burns	8.5	8.3	8.7	10	12.4	7.9	6.6	7.6	8.5	7.6	6.4	7.6	10.7	8.6	9.8	11.4	140.6	144
7	WappuCar	7.3	4.7	5.9	9	7.6	9.6	10.8	6.7	5.4	6.4	9	11	8.2	5.3	6.6	4.5	118	160
8	Bulle	3.4	9.7	4.8	11.6	3.1	4.7	9.1	4.4	6	8.9	3.2	14.6	5.8	4.4	2.6	3.8	100.1	137
9	Ali2	7.5	3.3	12.6	3.7	8.4	4.9	3	4.9	4.3	3.1	8.8	2.8	7.3	5.4	8.9	6.7	95.6	100
10	Magic	3.4	5.5	7.2	10.4	4.2	11.4	3	1.3	4.1	2.7	3.1	5.7	4.9	7.7	5.3	6.6	86.5	60
11	SBv1r4-1	1.6	4.1	1.1	2.5	3.5	2.7	2.9	7.7	1.8	3.9	4	7.1	3.8	1.8	2.2	4.3	55	85
12	Eagle	0.2	5	6.2	0.2	9.1	0	0.4	1.1	7.5	4.4	4.8	0.3	0.7	5.8	1.5	1.7	48.9	51
13	Apex8	2.3	0.8	9.6	1.3	3	3.8	0.9	2.9	1.4	0.4	0.4	2.2	3.8	1.1	3.9	1.5	39.3	67
14	JR001	0	0.3	0	1	0.1	0.3	0.5	0.4	0.2	0	0.1	0.9	0.4	0.5	0.1	0.1	4.9	28

Table 5 Comparison of evolved drivers with machine-generated drivers.

Author	Track	Lap Time (seconds)		
		Reported	GP-Single-1	GP-Single-2
Ng et al.	v03	59.4	55.5 ± 1.4 (std. error: 0.14)	**49.3** ± 0.1 (0.01)
	oval	33.0	31.0 ± 0.1 (0.01)	**30.7** ± 0.1 (0.01)
	complex	209.0	**199.4** ± 5.9 (0.59)	204.4 ± 1.3 (0.13)
Coulom	clkwis	38.0	37.7 ± 0.1 (0.01)	**36.5** ± 0.1 (0.01)
Cleland	v01	37.4	37.9 ± 1.6 (0.16)	**35.0** ± 0.1 (0.01)

set of the evolved drivers, but *were* used to train most of the machine-generated solutions.

The human-crafted controllers described in Tables 2 and 3 were built for multiple-lap, multiple-car scenarios, in which additional behavioral patterns—such as overtaking slow opponents, damage control, and fuel consumption monitoring—might be required. However, we assume that most human-crafted controllers would attempt to drive as fast as possible when no opponents are nearby, which is the case in the single-car scenario. Each of the machine-generated controllers was designed for the single-car scenario, differing only in the race-length parameter: Ng et al.'s controllers were trained on 60-lap races, Coulom's controllers were trained on a single-lap scenario, and Cleland's incorporated very long training phases during a single race, usually featuring hundreds of laps. All three controllers, however, aimed at reducing the average time of a single lap.

As over-specialization is a common phenomenon in many machine-learning approaches, the emergence of generalized solutions is nontrivial. We surmised that our choice of a complex track for fitness evaluation, combined with a sound yet simple set of genetic building blocks, contributed greatly to the generalization capabilities.

To further explore this hypothesis, we executed several evolutionary runs using track v01, which is a fairly simple one (compare Figure 5c, depicting v01, with Figure 5f, depicting sepang—which we used during evolution). The individuals evolved in these runs were highly competitive when driving on their training track (v01): the best-of-run was able to complete a lap in 34.1 (± 0.6) seconds (averaged over 100 runs), a result which

Fig. 6 GP-Single-2: Expression for wheel angle, α.

```
(% (% (% (% (IFG 0.702 AH AA (* NV -0.985))
(- AH (neg AH))) (- (% 1.0 (% V AH)) (neg AH)))
(- (- (* NV (neg NV)) (neg AH)) (neg AH))) (- (% 1.0
(% V AH)) (neg (% (% 1.0 (% V AH)) (% V AH)))))
```

Fig. 7 GP-Single-2: Expression for speed, v.

```
(IFP (abs (% V AH)) (- (% 1.0 (% V AH)) (neg (-
(* NV (* NV -0.868)) (neg AH)))) (% (neg (- (-
(* NV (neg TR)) (neg AH)) (neg AH))) ( - (% 1.0
(% V AH)) (neg (% (* NV (neg NV)) (% V AH))))))
```

is 3.8 seconds better than GP-Single-1 and 0.9 seconds better than GP-Single-2 on average. However, on unseen tracks, taken from Table 5, this controller's performance was rather poor: on v03 it completed a lap in 90.0 ($\pm$ 19.2) seconds on average, on clkwis it completed a lap in 71.7 ($\pm$ 17.9) seconds on average, and on sepang it failed altogether, having reached a critical damage level before completing a single lap.

The large error margins also suggest that the controller behavior was inconsistent on unseen tracks; it was probably intolerant to subtle random factors on such tracks, since its behavior was specialized to the simple track on which it was trained. Hence, we conclude that our choice of a *complex* track contributed greatly to the generalized nature of the evolved controllers.

In an attempt to further understand the evolved controllers we analyzed their code (refer to Table 1 for definitions of functions and terminals used in code). As explained in Section 4, each driver comprises two LISP expressions: one provides the desired wheel angle α, while the other provides the desired speed v. Both expressions for GP-Single-2 are shown, respectively, in Figures 6 and 7. Although seemingly complex, these expressions can be simplified manually:

$$\alpha = \Psi \cdot \left(\frac{1}{2AH} \cdot \frac{1}{\frac{AH}{V}+AH} \cdot \frac{1}{2AH-NV^2} \cdot \frac{1}{\frac{AH}{V}-(\frac{AH}{V})^2}\right), \quad (1)$$

where:

$$\Psi = \begin{cases} AA & AH < 0.7 \\ -0.98 \cdot NV & AH \geq 0.7, \end{cases}$$

and

$$v = |AH \cdot (\frac{1}{V} - 1) + 0.87 \cdot NV^2|. \quad (2)$$

These equations intimate at the logic behind the evolved controller's decisions. The Ψ element in Equation 1 shows that the steering behavior depends on the distance, AH, to the upcoming curve: when the next turn is far enough, the controller slightly adjusts the wheel angle to prevent drifting off track; when approaching a curve, however, the controller steers according to the relative curve angle—steep curves will result in extreme wheel angle values.

The AH indicator is used in Equation 2 too, and we observe that the desired speed is also determined by the distance to the next curve, among other factors.

It is evident that the expression AH/V is used quite frequently: one instance is seen in the speed equation, and three instances in the steering equation. Given that AH is the distance to the upcoming road shoulder, and V is the current velocity, this expression is simply the *time to crash* indicator: when will the car veer off-road if it keeps its current speed and heading. As this indicator is undoubtedly important for a race-car controller, and *wasn't provided as a genetic building block*, evolution found a way of expressing it—and used it extensively.

6 Concluding remarks and future work

In this work we used GP to create RARS controllers, finding that the evolutionary approach yields high-performance controllers, able to compete successfully both with human-crafted and machine-generated controllers.

The evolved drivers demonstrate a high degree of generalization, enabling them to perform well on most tracks—including ones that were not used during the evolutionary process. We noted that using a complex track for fitness evaluation, coupled with a comprehensive yet simple set of genetic building blocks, contributed greatly to our controllers' generalization capabilities. We also observed the emergence of useful code patterns, such as the *time to crash* indicator. Such patterns were repeatedly used in the evolved individuals' code, acting as evolution-made genetic building blocks.

Having focused on single-car, single-lap races, we can expand our research to more complex scenarios, including multiple cars, multiple laps, damage control and pit stops for repairs and refueling. This can be done during evolution, with the guidance of an appropriate fitness function, and not just post-evolutionarily, as we did.

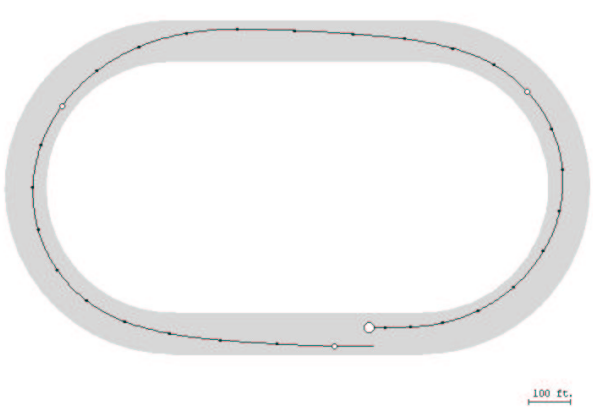
Another approach might use genetic algorithms to pre-compute an optimal path, to be combined with a GP-evolved controller in charge of following the path, for either single-car or multiple-car scenarios. Using GAs for path optimization has been done before (e.g., Eleveld's *DougE1* [9]) but never in combination with a machine-learning approach to the path-following behavior.

In addition, the RARS engine may be replaced with its successor, TORCS. This latter has, among others, the option of racing against human-*controlled* (as opposed to human-*crafted*) drivers, which is another interesting challenge.

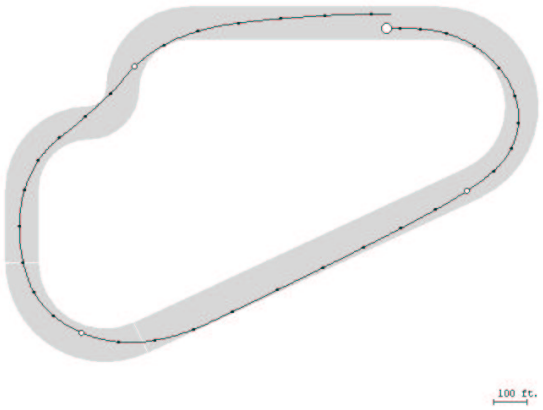
References

1. Butz, M.V., Lönneker, T.D.: Optimized sensory-motor couplings plus strategy extensions for the TORCS car racing challenge. In: CIG'09: Proceedings of the 5th International Conference on Computational Intelligence and Games, pp. 317–324. IEEE Press, Piscataway, NJ, USA (2009)
2. Cardamone, L., Loiacono, D., Lanzi, P.L.: Learning drivers for TORCS through imitation using supervised methods. In: CIG'09: Proceedings of the 5th International Conference on Computational Intelligence and Games, pp. 148–155. IEEE Press, Piscataway, NJ, USA (2009)
3. Cardamone, L., Loiacono, D., Lanzi, P.L.: On-line neuroevolution applied to the open racing car simulator. In: CEC'09: Proceedings of the Eleventh Congress on Evolutionary Computation, pp. 2622–2629. IEEE Press, Piscataway, NJ, USA (2009)
4. Chaperot, B.: Motocross and artificial neural networks. In: Game Design and Technology Workshop (2005)
5. Chaperot, B., Fyfe, C.: Improving artificial intelligence in a motocross game. In: IEEE Symposium on Computational Intelligence and Games (2006)
6. Cleland, B.: Reinforcement learning for racecar control. Master's thesis, The University of Waikato (2006)
7. Coulom, R.: Reinforcement learning using neural networks, with applications to motor control. Ph.D. thesis, Institut National Polytechnique de Grenoble (2002)
8. Ebner, M., Tiede, T.: Evolving driving controllers using genetic programming. In: CIG'09: Proceedings of the 5th International Conference on Computational Intelligence and Games, pp. 279–286. IEEE Press, Piscataway, NJ, USA (2009)
9. Eleveld, D.: Douge1 (2003). URL http://rars.sourceforge.net/selection/douge1.txt
10. Floreano, D., Kato, T., Marocco, D., Sauser, E.: Co-evolution of active vision and feature selection. Biological Cybernetics **90(3)**, 218–228 (2004)
11. Koza, J.R.: Genetic Programming: On the Programming of Computers by Natural Selection. MIT Press, Cambridge, Mass. (1992)
12. Langdon, W.B.: Size fair and homologous tree genetic programming crossovers. Genetic Programming and Evolvable Machines **1**(1/2), 95–119 (2000)
13. Muñoz, J., Gutierrez, G., Sanchis, A.: Controller for TORCS created by imitation. In: CIG'09: Proceedings of the 5th International Conference on Computational Intelligence and Games, pp. 271–278. IEEE Press, Piscataway, NJ, USA (2009)
14. Ng, K.C., Scorcioni, R., Trivedi, M.M., Lassiter, N.: Monif: A modular neuro-fuzzy controller for race car navigation. IEEE International Symposium on Computational Intelligence in Robotics and Automation **0**, 74 (1997)
15. Onieva, E., Pelta, D.A., Alonso, J., Milanés, V., Pérez, J.: A modular parametric architecture for the TORCS racing engine. In: CIG'09: Proceedings of the 5th International Conference on Computational Intelligence and Games, pp. 256–262. IEEE Press, Piscataway, NJ, USA (2009)
16. Perez, D., Recio, G., Saez, Y., Isasi, P.: Evolving a fuzzy controller for a car racing competition. In: CIG'09: Proceedings of the 5th International Conference on Computational Intelligence and Games, pp. 263–270. IEEE Press, Piscataway, NJ, USA (2009)
17. Pyeatt, L.D., Howe, A.E.: Learning to race: Experiments with a simulated race car. In: Proceedings of the Eleventh International Florida Artificial Intelligence Research Society Conference, pp. 357–361. AAAI Press (1998)
18. Sáez, Y., Perez, D., Sanjuan, O., Isasi, P.: Driving cars by means of genetic algorithms. In: Rudolph, G., Jansen, T., Lucas, S.M., Poloni, C., Beume, N. (eds.) Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN X), *Lecture Notes in Computer Science*, vol. 5199, pp. 1101–1110. Springer (2008). URL http://dblp.uni-trier.de/db/conf/ppsn/ppsn2008.html
19. Sipper, M.: On the origin of environments by means of natural selection. AI Magazine **22**(4), 133–140 (2001)
20. Stanley, K., Kohl, N., Sherony, R., Miikkulainen, R.: Neuroevolution of an automobile crash warning system. In: GECCO'05: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, pp. 1977–1984. ACM, New York, NY, USA (2005)
21. Tanev, I., Joachimczak, M., Shimohara, K.: Evolution of driving agent, remotely operating a scale model of a car with obstacle avoidance capabilities. In: GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 1785–1792. ACM, New York, NY, USA

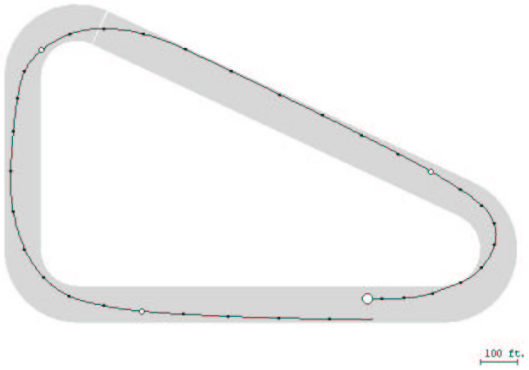
(2006)
22. Togelius, J., Lucas, S.M.: Evolving controllers for simulated car racing. In: Proceedings of the Congress on Evolutionary Computation (2005)
23. Togelius, J., Nardi, R.D., Lucas, S.M.: Towards automatic personalised content creation in racing games. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games (2007)
24. Wloch, K., Bentley, P.J.: Optimising the performance of a formula one car using a genetic algorithm. In: In Proceedings of Eighth International Conference on Parallel Problem Solving From Nature, pp. 702–711 (2004)



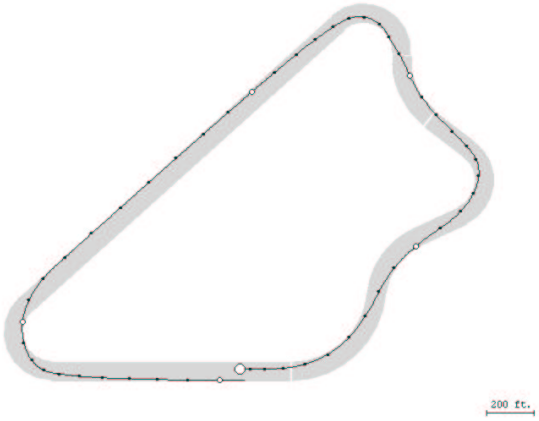
(a) oval [30.8 seconds]



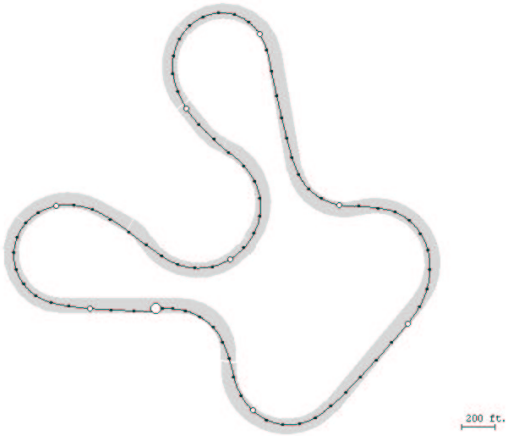
(b) clkwis [36.5 seconds]



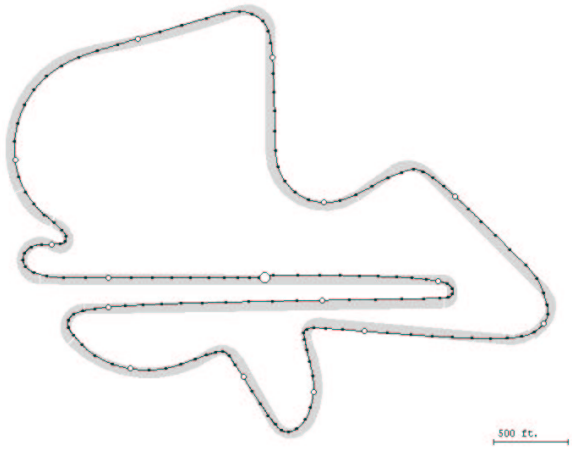
(c) v01 [35.0 seconds]



(d) aug04/race13 [50.8 seconds]



(e) aug04/race5 [83.1 seconds]



(f) sepang [159.5 seconds]

Fig. 5 Performance of GP-Single-2 on six tracks. Black dots represent one-second intervals, while white dots represent ten-second intervals. The large white dot is the starting point, from which the car starts moving. Some advanced driving techniques can be observed from these figures by examining the path line and the density of the time marker dots—which implicitly indicate the car's speed at any given time. The car slows down when approaching sharp curves, thus reducing the risk of skidding (tracks (c), (d), and (f)). Tracks (b) and (d) exhibit a special slalom behavior, where the controller doesn't follow the curvature of the road, but drives straight through the slalom instead. Finally, tracks (b), (c), (d), and (f) depict the controller's attempt to maximize the path radius by touching the inner shoulder at mid-curve, thus allowing the car to travel faster within the curve.