Evolving Driving Controllers using Genetic Programming

Marc Ebner and Thorsten Tiede

***Abstract*—Computational gaming requires the automatic generation of virtual opponents for different game levels. We have turned to artificial evolution to automatically generate such game players. In particular, we have used Genetic Programming to automatically evolve computer programs for computer gaming. With Genetic Programming, in theory, it is possible to generate any kind of program. The programs are not constrained as much as they are in other computational learning approaches, e.g. neural networks. We show how Genetic Programming improved upon a manually crafted race car driver (proportional controller). The open race car simulator TORCS was used to evaluate the virtual drivers.**

I. Introduction and Motivation

For computational games it is very important to be able to create a diverse range of interesting opponents against which a game player can measure its abilities. If the artificial game player is too easy to outperform, the actual player may quickly lose interest in the game. He may also lose interest, if the virtual opponent is too good at playing the game. Such virtual game players can of course be constructed manually. However, we have turned to artificial evolution to automatically generate game players using Genetic Programming [1], [2], [3]. Humans, after all, are product of natural evolution which created game playing ability in the first place. With this contribution we show how artificial evolution was able to improve upon a hand crafted virtual driver.

Genetic programming has already been used by several researchers in the context of game playing. Reynolds [4] evolved corridor following behavior for a vehicle driving in a 2D world. He also used coevolution to evolve players for the game of tag [5]. Siegel and Chaffee [6] evolved programs which could play Tetris. Koza [1] and Rosca [7] evolved programs which could control an agent in the Pac Man game. Schloman and Blackford [8] evolved player strategies for Quake 2. Anderson [9] evolved control algorithms for the arcade game Asteroids. Additional 2D space game behaviors were addressed by Jackson[10] and Francisco and dos Reis [11], [12].

Ciesielski et al. [13] used Genetic Programming to evolve different behaviors for the RoboCup tournament. Bajurnow and Ciesielski [14] suggested to use layered learning to evolve more complex behaviors. Corno et al. [15] evolved assembly language programs for the game of corewar. Crawford-Marks et al. [16] developed a Quidditch simulator and coevolved teams for this game using the stack-based programming language Push. Langdon and Poli [17] evolved players for the game of pong which outperformed human players. Shichel et al. [18] evolved Robocode players which finished third place in a competition of 27 players. Doherty and O'Riordan [19] evolved team behaviors for combative computer games. Agapitos et al. [20] used Genetic Programming to evolve controller representations for a simulated car racing game. They also compared the performance of the evolved controllers to neural network controllers e.g. multilayer perceptrons [21]. Agapitos et al. found that the neural network controllers performed better and generalized better. Togelius et al. [22] experimented with multi-population competitive coevolution within the same simulation environment. Agapitos et al. [23] used multi-objective optimization to jointly optimize different objectives such as distance traveled, performance relative to competitors number of collisions or speed. Tanev and Shimohara [24] used strongly typed genetic programming to evolve parameters which are used to control a remotely operated scale model of a car. The control algorithm was first evolved in simulation and then ported to the real car. Tanev and Shimohara report human competitive performance of the optimized driver when compared to a human controlled radio car.

Marc Ebner and Thorsten Tiede are with the Eberhard Karls Universität Tübingen, Wilhelm Schickard Institut für Informatik, Abt. Rechnerarchitektur, Sand 1, 72076 Tübingen, Germany, (phone: +49-7071-29-78978; fax: +49-7071-29-5091; email: marc.ebner@wsii.uni-tuebingen.de).

Fig. 1. Screenshot from the open source race car simulation TORCS.

II. TORCS – A virtual environment for testing race car drivers

For our experiments, we have used the open source race car simulation TORCS (`torcs.sourceforge.net`). TORCS was created by Eric Espié and Christophe Guionneau. The current maintainer of the project is Bernhard Wymann. The TORCS simulator provides 30 different tracks on which driving abilities can be compared. A player can choose one of 42 different cars and and he can also choose

978-1-4244-4815-9/09/$25.00 ©2009 IEEE

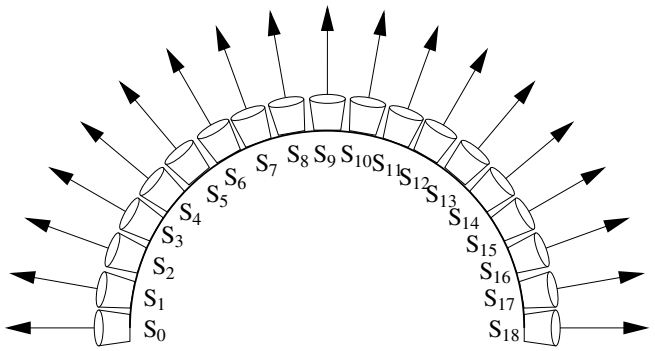

Fig. 2. The race car is equipped with 19 sensors S_i with $i \in \{0, ..., 18\}$ arranged around a half circle. Each sensor returns the distance in meters to the track boundary. Distances larger than 100 are clipped to 100m.

among 50 opponents to race against. The virtual car can be steered using a joystick, an actual steering wheel (if supported), the mouse or the keyboard. The graphics output is in 3D featuring lighting, smoke, skidmarks on the road and glowing brake disks (Figure 1). Up to four different players can race against each other using a split screen mode. It is possible for users to develop their own robots which can be used to drive a car by following the TORCS guidelines.

Loiacono et al. [25] have developed a race car client for the WCCI 2008 competition. This car racing competition software extends TORCS with a client-server architecture that separates the development of car controllers from TORCS. Participants in the competition can develop different robots by simply modifying the client side. They do not have to fully understand how TORCS works. The server-robot module is integrated into TORCS. It is used to read out the current state of the car's sensors and also has access to additional information from TORCS. Table I gives an overview about the different sensors which are available from the car.

Data include the current orientation of the car along the track (α), the distance of the car measured from the start line (distFromStart) , the total distance traveled (distRaced), the time which has elapsed on the current track (t_{curLap}), the time it took to complete the last lap (t_{lastLap}) and the current position in the race (racePos). Additional data which is available is the total damage which has been incurred to the car (damage), the remaining fuel (fuel), the current gear (gear), the rounds per minute of the engine (rpm), the velocity of the car in the direction of the track (v_x), the velocity of the car perpendicular to the direction of the track (v_y) as well as the velocity of the four wheels $v_{\text{wheel,i}}$ with $i \in \{1, 2, 3, 4\}$. Each car is also equipped with 19 distance sensor which measure the distance from the car to the edge of the track (S_i with $i \in \{0, ..., 18\}$). Sensors are also available to measure the distance to other cars in the race (O_i with $i \in \{1, ..., 18\}$).

All of these data fields are periodically sent from the server to the client. The client reacts to these data and tries to keep the car on the track and finish the race as number one. The car can be steered by specifying the position of the steering wheel (steering). It can be accelerated by pushing the gas pedal (gas pedal). It is decelerated via the brake pedal (brake pedal). The driver can also set the gear (gear). All effectors are listed in Table II. The client sends the values for these effectors to the server. This process continues until the robot either finishes the race or the car leaves the race track.

We have built upon the client which was supplied for the WCCI 2008 competition [27]. Loiacono et al. [26] summarize the results of the WCCI 2008 competition. Apart from a neural network controller, all other submissions were rule based controllers. We try to evolve computer programs using simulated evolution. In particular, we have used Genetic Programming which allows us to evolve arbitrary computer programs.

III. Evolution of virtual race car drivers

Genetic Programming [1], [2], [3] is an automatic method to automatically generate computer programs. For our experiments we have used the Evolutionary Computation in Java (ECJ) package developed by Luke et al. (`www.cs.gmu.edu/~eclab/projects/ecj`). We start off with a population of possible solutions to our problem. Our problem is to complete the race in the shortest amount of time. Therefore, each solution is a race car driver. Given two race car drivers, the race car driver who has finished the track in the shortest amount of time is clearly the better driver. If both drivers steer off the track, the driver who leaves the track last is the better driver. Given a population of race car drivers of varying expertise (some may not be able to drive successfully at all), we select the drivers which perform best. The drivers are selected using the Darwinian principle "survival of the fittest". The successful drivers are then modified slightly to create a new population of drivers and the process is repeated. Over several generations, this process causes the drivers to adapt to the problem (finishing the race).

For our experiments we have used tree-based Genetic Programming [1], [2]. With tree-based Genetic Programming, programs are represented as trees. Each tree consists of internal and external nodes. The external nodes are used to provide input to the program. Each tree computes an output based on the input supplied through the external nodes. Since each car is basically controlled by turning the steering wheel and by specifying whether it should accelerate or decelerate we have decided that a car driving program, i.e. an individual of the population, consists of two trees. The first tree computes the steering direction of the steering wheel. The second tree computes whether the car should accelerate or decelerate. The gear is set automatically depending on the rpm of the motor. Table III shows the set of terminal symbols (external nodes) and the set of elementary functions (internal nodes) which are used for the first tree. The set of terminal symbols and elementary functions which are used for the second tree are shown in Table IV. Steering and the acceleration/deceleration can be controlled independently since we use two separate trees for each. However, the acceleration/deceleration tree has access to terminal symbols v_x (velocity of the car) and also the difference between the

TABLE I
SENSORS AVAILABLE FROM A SIMULATED CAR OF THE RACING SIMULATOR TORCS [26].

Name	Range/Units	Description
α	$[-\pi, \pi]$/rad	orientation of car relative to the current street orientation
distFromStart	$[0, \infty]$/m	distance from start line to current position of car measured along race track
distRaced	$[0, \infty]$/m	total distance traveled since beginning of race
t_{curLap}	$[0, \infty]$/s	elapsed time on current lap
t_{lastLap}	$[0, \infty]$	time elapsed for last lap
racePos	1, 2, ...	current rank of car in race
damage	$[0, \infty]$/point	total dame incurred to car
fuel	$[0, \infty]$/l	fuel left in fuel tank
gear	$\{-1, 0, 1, ..., 6\}$	current gear position (-1: backwards, 0: neutral)
rpm	$[2000-7000]$/rpm	rounds per minute of the motor
d_y	$[-1, 1]$	displacement of car from center of track (normalized to 1)
v_x	$[-\infty, \infty]/\frac{\text{km}}{\text{m}}$	velocity of car in track direction
v_y	$[-\infty, \infty]/\frac{\text{km}}{\text{m}}$	velocity of car perpendicular to track direction
$v_{\text{wheel,i}}$	$[0, \infty]/\frac{\text{rad}}{\text{s}}$	velocity of wheel $i \in \{1, 2, 3, 4\}$
S_i	$[0, 100]$/m	dist. to track boundary measured by 19 sensors $i \in \{0, ..., 18\}$ as shown in Figure 2
O_i	$[0, 100]$/m	distance to opponents measured by 18 sensors $i \in \{1, ..., 18\}$

TABLE II
ACTUATORS OF A SIMULATED CAR OF THE RACING SIMULATOR TORCS [26].

Name	Range	Description
gas pedal	$[0, 1]$	acceleration
brake pedal	$[0, 1]$	brake (0: don't brake, 1: full brake)
steering	$[-1, 1]$	orientation of steering wheel (-1: maximum left, 1: maximum right)
gear	-1, 0, 1,..., 6	shift into gear as specified
meta control	0, 1	meta control flag (0: do nothing, 1: restart race)

TABLE III
SET OF TERMINAL SYMBOLS AND ELEMENTARY FUNCTIONS OF TREE 1 WHICH WAS USED TO STEER THE CAR.

Name	Arguments	Description
ERC1	0	ephemeral random constant with range $[-1, 1]$
ERC150	0	ephemeral random constant with range $[-150, 150]$
c_p	0	constant for hand-crafted proportional controller $c_p = -0.0234$
LR0	0	average difference between left and right track sensors $\frac{1}{2}(S_{15} + S_{14}) - \frac{1}{2}(S_3 + S_4)$
abs	1	absolute value of argument
+	2	sum of both arguments
-	2	difference of both arguments
*	2	product of both arguments
/	2	protected division

TABLE IV
SET OF TERMINAL SYMBOLS AND ELEMENTARY FUNCTIONS OF TREE 2 WHICH WAS USED TO OPERATE GAS AND BRAKES.

Name	Arguments	Description
ERC1	0	ephemeral random constant with range $[-1, 1]$
ERC50	0	ephemeral random constant with range $[-150, 150]$
c_1	0	first constant (used by hand-crafted gas/brake controller) $c_1 = -0.022$
c_2	0	second constant (used by hand-crafted gas/brake controller) $c_2 = 100$
LR1	0	difference between left and right front facing track sensors $(S_8 - S_{10})$
S_9	0	front facing sensor
v_x	0	velocity of car
abs	1	absolute value of argument
+	2	sum of both arguments
-	2	difference of both arguments
*	2	product of both arguments
/	2	protected division

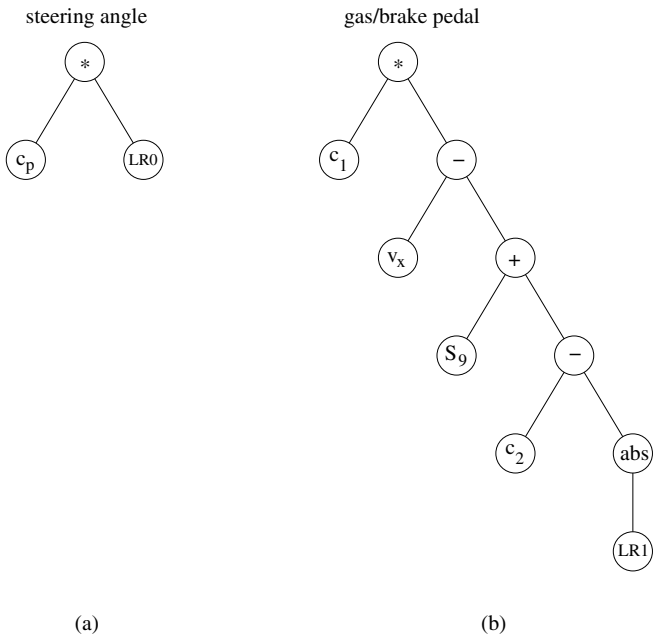

Fig. 3. Sample individual. Manually constructed using the set of terminal symbols and the set of elementary functions shown in Table III and Table IV. (a) tree which controls the steering angle of the car. (b) tree which controls the gas/brake pedal.

front facing sensors S_8 and S_{10}. Therefore it is possible to evolve a correlated steering and acceleration/deceleration behavior.

The idea behind the set of terminal symbols and the set of elementary functions which were used for the two trees is that evolution would build or improve a proportional controller [28]. We evolve the error expression used by the proportional controller. The goal of the steering controller is that it should keep the car in the middle of the road. The car is exactly in the middle of the road if the left and right sensors show approximately the same measurements. In order to get reliable measurements, we averaged the sensor data from S_{15} and S_{14} to get a reading towards the right hand side of the car. We averaged the sensor data from sensors S_3 and S_4 to get a reading towards the left side of the car. Both are subtracted from each other to obtain an error measure which is available through a terminal symbol (LR0). Such an error measure allows easy control of a vehicle to steer toward the center of the track and is reminiscent of a Braitenberg vehicle [29]. Braitenberg showed how crossed connections from the sensors to the actuators can be used to steer a vehicle. The other symbols (ephemeral constants and arithmetic functions) were supplied to fully construct a proportional controller. An ephemeral random constant is selected once the node is generated from the allowed range and then stays constant during the life of the node. The constant c_p was useful for a manually constructed controller that's why we included it in the set of elementary functions.

Figure 3(a) shows a tree which actually steers the car and was constructed manually using the nodes from Table III. This tree is evaluated in the car client. It receives its input from the terminal node LR0 and transforms this data into an output value. This output value is used to provide the angle of the steering wheel for the next iteration. For each time step of the simulation this tree is evaluated and issues the desired steering angle to the robot server. The acceleration/deceleration command is issued in the same time step but is computed by the second tree.

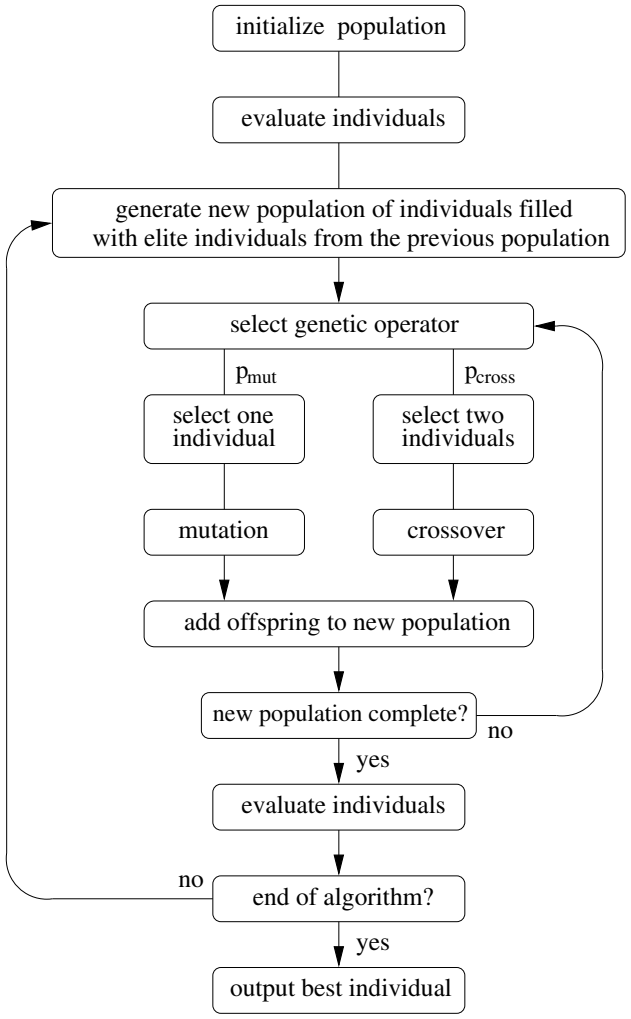

Fig. 4. Outline of the evolutionary algorithm which we used to evolve race car drivers. Starting from an initial population of individuals, a new generation of individuals is created by selecting highly fit individuals and then applying the genetic operators mutation (Figure 5) and crossover (Figure 6). This process is repeated until a maximum number of generations has been reached.

The nodes which are available to construct the tree to accelerate or decelerate include sensors which are oriented toward the front of the car. The function LR1 provides the difference between the sensors S_8 and S_{10}. The frontal sensor S_9 is also included in the set of functions. In order to react to the current speed of the car, the current velocity of the car was also included (v_x). We again included arithmetic functions and constants which we thought would be useful. Constants c_1 and c_2 were used for a manually constructed tree which controls the speed of the car as shown in Figure

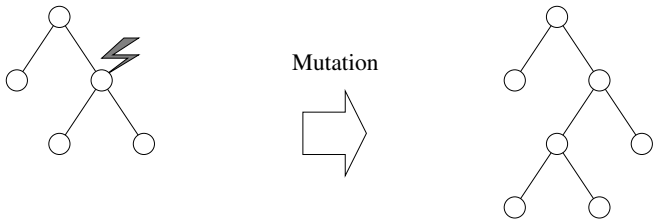

Fig. 5. Mutation operator. A randomly selected node is replaced by a newly generated subtree.

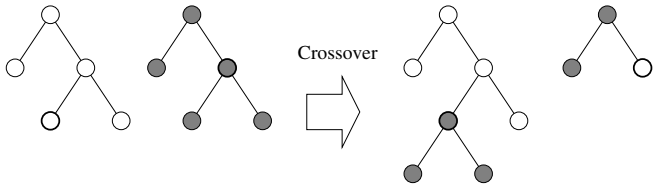

Fig. 6. Crossover operator. Two subtrees are exchanged between two selected individuals.

3(b). The output of this tree is used to specify the value of the desired acceleration (gas pedal) if the value returned is positive. Otherwise the absolute value of the tree output is used to specify the value of the brake (brake pedal).

The evolutionary algorithm which we use to evolve such individuals is shown in Figure 4. The randomly generated individuals of the first generation are generated using ramped half and half initialization using a minimum depth of 2 and a maximum depth of 6 [1]. Maximum depth of generated trees during evolution is limited to 17. Each individual is evaluated on five different tracks which are shown in Figure 7. We initially used only a single track to evolve race car drivers. However, the drivers did not generalize very well to unknown tracks. The first and fourth track feature a left turn at the beginning of the race track. The second, third and fifth track feature a right turn at the beginning of the track. The five tracks also differ in the length of the straight part of the track before the first turn shows up. The manually constructed driver is able to stay on two of the tracks for the allotted time steps. The fitness f of an individual is the average of the performance f_i with $i \in \{1, 2, 3, 4, 5\}$. Each individual is evaluated for 1000 time steps on each track. No other drivers are present on the track during evolution. The fitness on each track is given as

$$f_i = d_{\text{max}} - d \tag{1}$$

where d is the distance traveled along the track. The distance is subtracted from the maximum possible distance which the car can drive along the track within 1000 time steps and is computed using $d_{\text{max}} = \frac{\text{maxTimeSteps}}{\text{timeStepsPerSecond}} \cdot v_{\text{max}}$ where v_{max} is the maximum velocity of the car.

We have used tournament selection with a tournament size of 7. The best 3 individuals are always copied into the next generation. Since the fitness may vary slightly from one evaluation to the next, we maintain a running average of all evaluations of an individual. The remaining individuals of the next generation are filled by applying one of two operators (mutation and crossover) to selected individuals. Each operator is applied with a probability of 50%. The mutation operator selects a random node of the tree and generates a new random subtree. The crossover operator selects two individuals and then exchanges two random subtrees between these two individuals to generate two offspring. Internal nodes are selected with a probability of 90% and external nodes are selected with a probability of 10%.

IV. Experiments and Results

We conducted four experiments. Each experiment starts off with a population of 200 individuals. We first tried to improve upon a manually constructed individual. For this experiment 1, we inserted the manually constructed individual which is shown in Figure 3 into the first generation. We also tried to evolve a successful controller from scratch (experiment 2). We repeated these two experiments with an extended function set (experiments 3 and 4). For these experiments we added two elementary functions `sum` and `last`. The function `sum` takes a single argument and sums up this argument over all time steps. For each time step, the current sum is returned. The function `last` stores the argument and returns the value which was computed during a previous evaluation of the node. By adding these two elementary functions which also have side effects it is possible to evolve PID controllers in experiments 3 and 4.

For each experiment, we conducted five runs with different starting seeds. Figure 8 shows the fitness statistics. The top four graphs show the minimum fitness for experiments 1-4. The bottom four graphs show the average fitness. Experiments 1 and 3 created the best car drivers after 50 generations. For these experiments, the manually constructed controller was added to the initial population. Artificial evolution was able to considerably improve upon the manually constructed controller. Evolution was not able to evolve a comparable controller from scratch within the same number of generations and a population size of 200 individuals. We used a t-test to investigate statistical significance between the four experiments. Starting from an entirely random population made the problem more difficult (statistically significant with a confidence of 94.7%). Making the set of elementary functions more powerful by adding elementary functions which also allow the evolution of PID controllers did not help. There is no statistical significant difference between experiments 1 and 3 and also not between 2 and 4.

The performance of the best evolved driver on tracks (a), (c), and (e) is shown in Figure 9. The plots show the lateral offset of the car along the track. The steering angle and the desired acceleration (gas pedal) and the desired deceleration (brake pedal) of the driver are also shown. The plots show the performance of the best evolved driver of generation 0, 10, 20, 30, 40, and 50. The best car at generation 0 left the track when evaluated on tracks (c), (d), and (e). The plots clearly show the driver improved during the course of evolution. In generation 50, the driver is able to keep the car on the track and actually manages to drive in the center of the track for

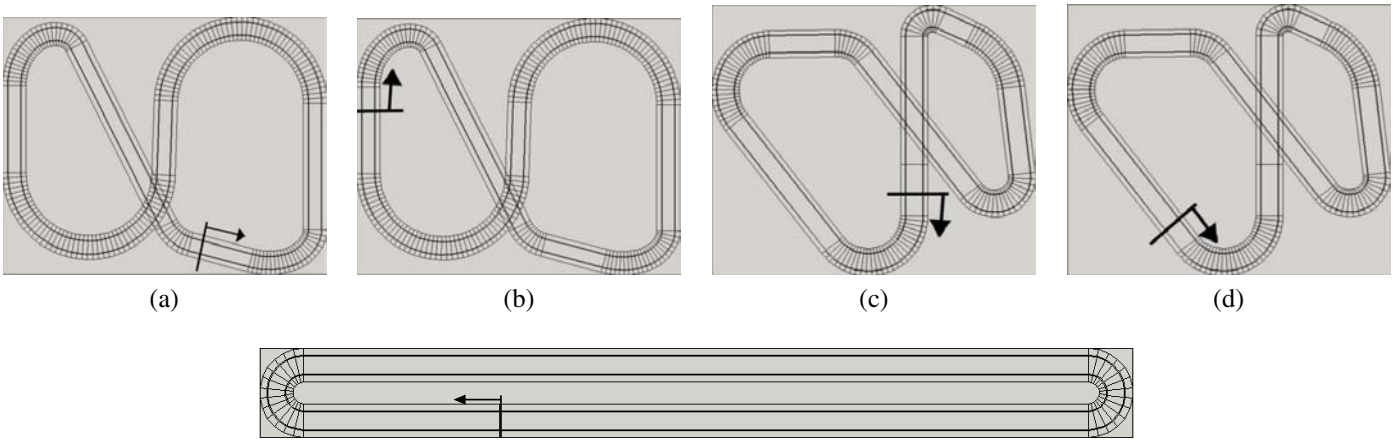

Fig. 7. Track (a)-(e) were used to evaluate the performance of the evolved drivers. The starting position along the track is also shown.

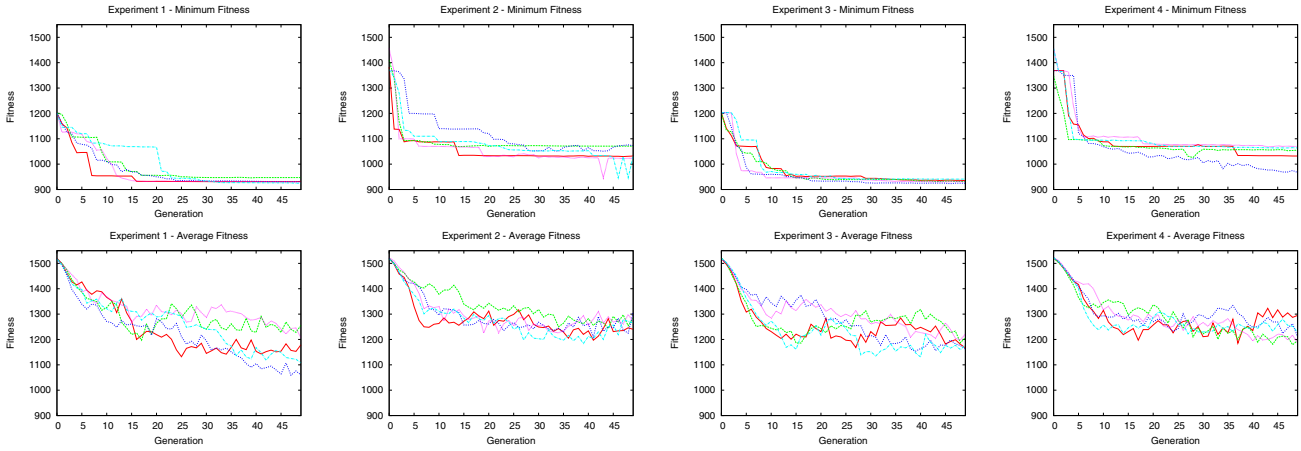

Fig. 8. Fitness statistics for experiments 1-4. The minimum fitness for each generation is shown in the top four graphs. The average fitness for each generation is shown in the bottom four graphs.

most of the simulation. Whenever the car drifts off to the left, the evolved driver steers the car to the right and vice versa. What can also be seen nicely in the plots showing the performance of generation 50 is that the evolved driver anticipates that the car is about to veer of the track. It actually brakes slightly before the car is about the leave the center of the track.

At the first generation, the best individual is able to drive for a maximum distance of 498m (on track (b)). At the end of evolution the best individual was able to cover a distance of 642m for this same track. On track (e), the best controller from generation 0 actually left the track after only 66m. However, after generation 50, it was able to drive for 624m on track (e) without leaving the track. The best evolved controller uses only the current speed and the front facing sensor (S_9) to control the acceleration of the car. It is currently not able to compete with other more elaborate manually constructed drivers. However such drivers usually have access to knowledge about the track curvature. It may be possible to evolve better controllers using a different representation, e.g. by supplying the angle to the distance sensor with the largest response as suggested by Butz and Lönneker [30].

Agapitos et al. [20] noted that evolved neural network and genetic programming drivers tend to oscillate quickly between different driving commands. In contrast to the experiments by Agapitos, with our representation, it should be noted that the evolved drivers show rather smooth steering behavior on tracks (a)-(d). Agapitos et al. also compared the performance of stateless and stateful controllers and noted that they did not find any difference between these two representations. In our experiments it seems that using stateful controllers, i.e. those which included the functions (`sum` and `last`) made the problem more difficult.

V. Conclusions

We have used Genetic Programming to evolve symbolic expressions which provide an error measure to control a virtual car (proportional controllers). Evolution was able to improve upon a manually constructed controller. Each car controller consisted of two symbolic expressions, the first controlling the steering angle and the second controlling the

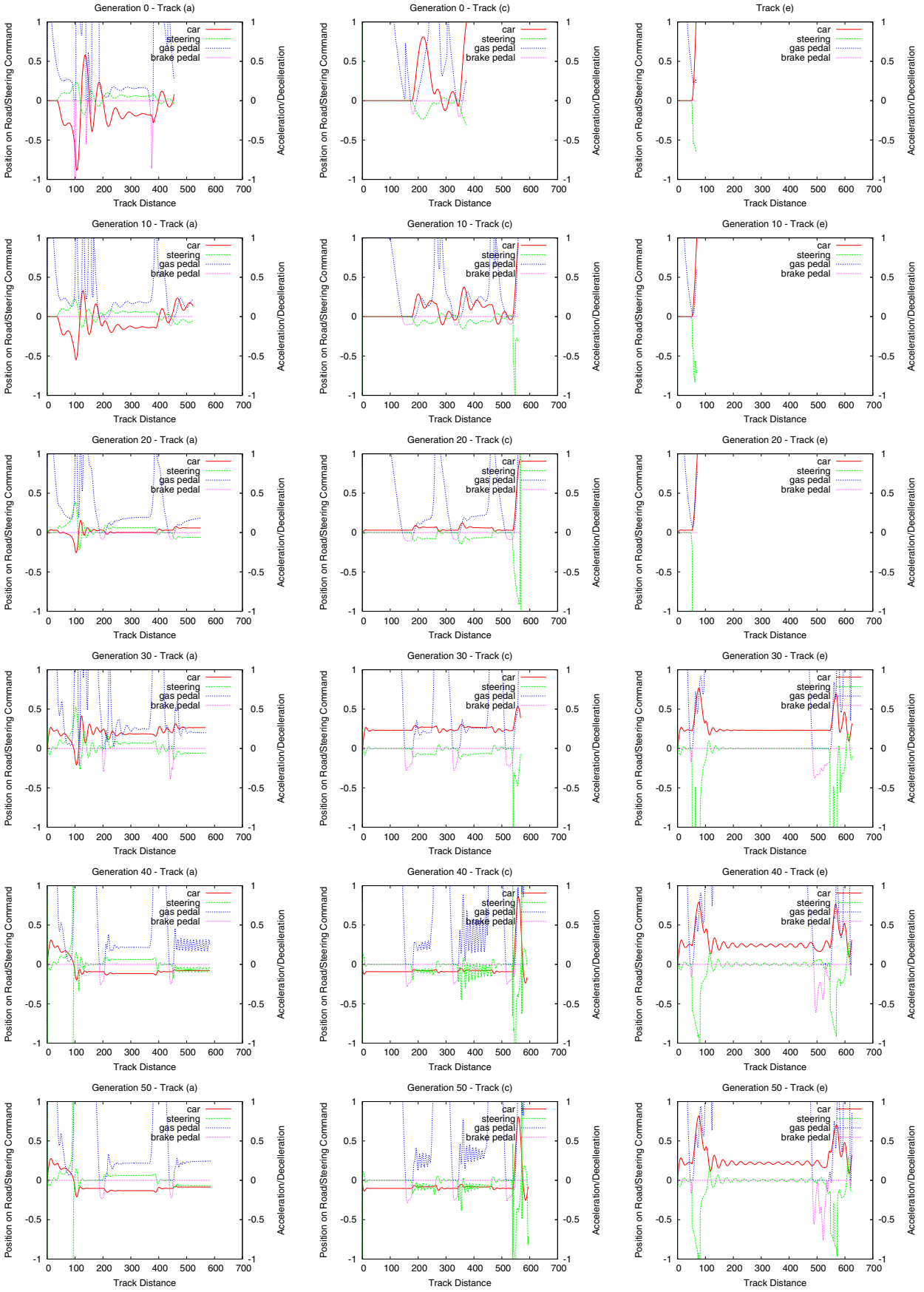

Fig. 9. Performance of the best evolved race car driver on tracks (a), (c), and (e) shown in Figure 7.

acceleration/deceleration of the car. The symbolic expression were represented as trees which were constructed by the Genetic Programming paradigm. In our experiments we found that it was particularly important to evaluate a single controller in different situations, i.e. on different tracks, in order to obtain a reliable assessment of the quality of the individual's ability to drive the car. Individuals evaluated only on single tracks tended to be overfitted to the track which was used during evolution.

REFERENCES

[1] J. R. Koza, *Genetic Programming. On the Programming of Computers by Means of Natural Selection*. Cambridge, Massachusetts: The MIT Press, 1992.

[2] ——, *Genetic Programming II. Automatic Discovery of Reusable Programs*. Cambridge, Massachusetts: The MIT Press, 1994.

[3] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming - An Introduction: On The Automatic Evolution of Computer Programs and Its Applications*. San Francisco, California: Morgan Kaufmann Publishers, 1998.

[4] C. W. Reynolds, "Evolution of corridor following behavior in a noisy world," in *From animals to animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior, Brighton, England, 1994*, D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, Eds. The MIT Press, 1994, pp. 402–410.

[5] ——, "Competition, coevolution and the game of tag," in *Artificial Life IV, July 6-8*, R. A. Brooks and P. Maes, Eds. Cambridge, MA: The MIT Press, 1994, pp. 59–69.

[6] E. V. Siegel and A. D. Chaffee, "Genetically optimizing the speed of programs evolved to play tetris," in *Advances in Genetic Programming 2*, P. J. Angeline and K. E. Kinnear, Jr., Eds. Cambridge, MA: MIT Press, 1996, pp. 279–298.

[7] J. P. Rosca, "Generality versus size in genetic programming," in *Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds. MIT Press, 28–31 Jul. 1996, pp. 381–387.

[8] J. Schloman and B. Blackford, "Genetic programming evolves a human competitive player for a complex, on-line, interactive, multi-player game of strategy," in *Genetic and Evolutionary Computation Conference. Proceedings of the Genetic and Evolutionary Computation Conference, July 7-11, San Francisco, California*. New York, NY: ACM, 2007, pp. 1951–1958.

[9] E. F. Anderson, "Off-line evolution of behaviour for autonomous agents in real-time computer games," in *Parallel Problem Solving from Nature - PPSN VII, September 7-11, Granada, Spain*, J. J. Merelo-Guervos, P. Adamidis, H.-G. Beyer, J.-L. Fernandez-Villacanas, and H.-P. Schwefel, Eds. Berlin: Springer-Verlag, 2002, pp. 689–699.

[10] D. Jackson, "Evolving defence strategies by genetic programming," in *Proceedings of the 8th European Conference on Genetic Programming, Lausanne, Switzerland, March 30*, M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, and M. Tomassini, Eds. Berlin: Springer-Verlag, 2005, pp. 281–290.

[11] T. Francisco and G. M. J. dos Reis, "Evolving combat algorithms to control space ships in a 2D space simulation game with co-evolution using genetic programming and decision trees," in *Proceedings of the Genetics and Evolutionary Computation Conference Workshop Proceedings: Defense Applications of Computational Intelligence (DAC), Atlanta, GA, July 12-16*. New York, NY: ACM, 2008, pp. 1887–1892.

[12] ——, "Evolving predator and prey behaviours with co-evolution using genetic programming and decision trees," in *Proceedings of the Genetics and Evolutionary Computation Conference Workshop Proceedings: Defense Applications of Computational Intelligence (DAC), Atlanta, GA, July 12-16*. New York, NY: ACM, 2008, pp. 1893–1900.

[13] V. Ciesielski, D. Mawhinney, and P. Wilson, "Genetic programming for robot soccer," in *RoboCup 2001: Robot Soccer World Cup V, Seattle, Washington*, A. Birk, S. Coradeschi, and S. Tadokoro, Eds. Berlin: Springer-Verlag, 2002, pp. 319–324.

[14] A. Bajurnow and V. Ciesielski, "Layered learning for evolving goal scoring behavior in soccer players," in *Proceedings of the 2004 IEEE Congress on Evolutionary Computation, June 20-23, Portland, Oregon*. IEEE Press, 2004, pp. 1828–1835.

[15] F. Corno, E. Sanchez, and G. Squillero, "On the evolution of corewar warriors," in *Proceedings of the 2004 IEEE Congress on Evolutionary Computation, June 20-23, Portland, Oregon*. IEEE Press, 2004, pp. 133–138.

[16] R. Crawford-Marks, L. Spector, and J. Klein, "Virtual witches and warlocks: A quidditch simulator and quidditch-playing teams coevolved via genetic programming," in *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference, July 26, Seattle, Washington*, M. Keijzer, Ed., 2004.

[17] William B. Langdon and R. Poli, "Evolutionary solo pong players," in *IEEE Congress on Evolutionary Computation September 2-5, Edinburgh, UK*. IEEE Press, 2005, pp. 2621–2628.

[18] Y. Shichel, E. Ziserman, and M. Sipper, "GP-robocode: Using genetic programming to evolve robocode players," in *Proceedings of the 8th European Conference on Genetic Programming, Lausanne, Switzerland, March 30*, M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, and M. Tomassini, Eds. Berlin: Springer-Verlag, 2005, pp. 143–154.

[19] D. Doherty and C. O'Riordan, "A phenotypic analysis of GP-evolved team behaviours," in *Genetic and Evolutionary Computation Conference. Proceedings of the 9th annual conference on Genetic and Evolutionary Computation, July 7-11, London, England*. New York, NY: ACM, 2007, pp. 1951–1958.

[20] A. Agapitos, J. Togelius, and S. M. Lucas, "Evolving controllers for simuated car racing using object oriented genetic programming," in *Genetic and Evolutionary Computation Conference. Proceedings of the 9th annual conference on Genetic and Evolutionary Computation, July 7-11, London, England*. New York, NY: ACM, 2007, pp. 1543–1550.

[21] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Volume 1: Foundations*. Cambridge, Massachusetts: The MIT Press, 1986.

[22] J. Togelius, P. Burrow, and S. M. Lucas, "Multi-population competitive co-evolution of car racing controllers," in *IEEE Congress on Evolutionary Computation September 25-28, Singapore*, D. Srinivasan and L. Wang, Eds. IEEE Press, 2007, pp. 4043–4050.

[23] A. Agapitos, J. Togelius, S. M. Lucas, J. Schmidhuber, and A. Konstantinidis, "Generating diverse opponents with multiobjective evolution," in *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games, December 15-18, Perth, Australia*. IEEE, 2008.

[24] I. Tanev and K. Shimohara, "Evolution of agent, remotely operating a scale model of a car through a latent video feedback," *Journal of Intelligent Robotic Systems*, no. 52, pp. 263–283, 2008.

[25] D. Loiacono, J. Togelius, and P. L. Lanzi, *Car Racing Competition WCCI2008. Software Manual*, Apr. 2008.

[26] D. Loiacono, J. Togelius, P. L. Lanzi, L. Kinnaird-Heether, S. M. Lucas, M. Simmerson, D. Perez, R. G. Reynolds, and Y. S. (2008), "The wcci 2008 simulated car racing competition," in *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games, December 15-18, Perth, Australia*. IEEE, 2008.

[27] T. Tiede, *Evolution eines Algorithmus zur Steuerung eines virtuellen Fahrzeugs in einer Rennsimulation*. Studienarbeit, Eberhard Karls Universität Tübingen, Wilhelm-Schickard-Institut für Informatik, Lehrstuhl Rechnerarchitektur, Jun. 2009.

[28] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 2nd ed. Reading, Massachusetts: Addison-Wesley Publishing Company, 1989.

[29] V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology*. Cambridge, Massachusetts: The MIT Press, 1984.

[30] M. V. Butz and T. D. Lönneker, "Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, Milano, Italy*. IEEE, 2009, p. (in print).