The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Automated synthesis of control algorithms from first principles

Henrik Berg, Roland Olsson, Per-Olav Rusås and Morgan Jakobsen

Abstract— **A variety of machine learning techniques have been employed to automatically create control algorithms for autonomous vehicles. Much research has focused on various "black box" approaches, in which the synthesized or learned control algorithms perform well when tested, but are difficult or impossible to analyze and understand. This paper presents the use of the ADATE system to evolve a control algorithm based on a racing car simulator. The system evolved compact and analyzable yet sophisticated control algorithms capable of driving millions of randomly generated tracks at high speeds without ever driving off the road. The approach presented is likely to be applicable to most automatic control problems, given a set of training examples and a suitable software simulator.**

I. Introduction

The design of a fully automatic control system is usually not a trivial task, especially not if an optimal or near-optimal solution is required. This paper examines the use of the ADATE evolutionary system for automatic programming to automatically evolve a control algorithm for a simulated racing car. The goal of the system is to control the speed and steering of a car in order to race as fast as possible on tracks with randomly varying geometry.

The paper is organized as follows. In Section 2, related work on autonomous car control is reviewed. Section 3 gives a brief introduction to the ADATE system for automatic programming, followed by a description of the car simulator used during evolution in Section 4. In Section 5, our experiments are described, their results are given and analyzed in Section 6, followed by a conclusion and directions for further research in Section 7.

II. Related work

Many different autonomous or semi-autonomous subsystems can be found in modern cars, or are likely to appear in commercial cars within few years. Some of these aid the driver in various tasks, e.g., Cruise Control [1], Active Steering [2] and Advanced Driver Assistance Systems [3], whereas others are used to control low-level aspects of the car dynamics, e.g., engine control [4], [5] and suspension control [6]. A notable commercially available example is the newest Honda Accord [7], which can drive autonomously on a highway for up to ten seconds, using a camera to detect road marking in order to keep the car in lane, and a radar system to detect other cars in order to control speed.

H. Berg is with the Norwegian defence research establishment (FFI), Horten, Norway.
R. Olsson is with the Faculty of Computer Science, Østfold University College, Halden, Norway.
P. O. Rusås is with Prediktor AS, Fredrikstad, Norway.
M. Jakobsen is with Kongsberg Norcontrol IT, Horten, Norway.

The use of evolutionary computation [8] and other machine learning [9] techniques have emerged as promising methods for the development of autonomous driver systems. A recent example is given in [10], which describes one of the speed control subsystems of Stanley [11], the winning car of the DARPA 2005 Grand Challenge [12].

Much research has focused on training of neural networks [13] for autonomous vehicle control [14], [15], [16], [17]. In [18], it is suggested that adding fuzzy logic to a neural network makes it more robust and better able to drive safely on unseen tracks. In [19], Neural Fuzzy Networks are constructed using a combination of evolution and training. The focus is on speed control in a highway situation containing numerous other cars, and it is demonstrated that the resulting controller is able to safely interact with the other cars, including cars changing lanes just in front of it, forcing the controller to slow down to maintain its safety distance. A more recent example of the use of fuzzy neural networks is [20], in which a set of fuzzy rules were generated based on a real human driver driving a simulated car on a race track. The rules control the acceleration of the vehicle based on the curvature of the road ahead, and as expected, when driving slowly or when driving on a straight road, more acceleration can be safely be performed than when driving fast or in sharp turns.

Togelius et al. [21], [22], [23], [24] have employed various evolutionary methods to evolve controllers for simulated RC-cars. In [21], different architectures and sensor settings are explored, concluding that the most promising controllers were evolved when using neural networks taking sensor inputs representing a first-person view. The robustness of evolved controllers when tested on tracks not used during training was investigated in [22]. Controllers were evolved using eight different tracks of varying difficulty level, but the performance of these controllers when tested on the remaining seven tracks was not satisfactory. It was found that the most robust controllers were acquired when using *incremental evolution*. In incremental evolution, the evolution of the controllers is initialized on one single track, and repeatedly a new track is added each time the performance of the evolved controllers have reached some predefined value. This is quite similar to our approach described in Section 5.3. It was also shown that once a set of robust controllers had been evolved, they could be further improved and specialized for a given track by evolving them further using only this track for training. This work is further refined in [23], in which Genetic Programming [25] is used to evolve controllers utilizing internal state variables. Reasonable use of the state variables is encouraged by a multi-objective

approach, resulting in controllers clearly superior to simple state-less controllers. Another addition is given in [24], where predictors are added, aiding the control system in deciding what actions to take in situations where sensor inputs may be delayed or completely absent for short periods.

A work slightly similar to that of Togelius et al. is presented by Tanev et al. [26]. Here, a driving style for one single, rather simple track is evolved. However, after a preliminary evolution using a software simulation, Tanev et al. evolve the controller further using a real RC car on a real track for a few more generations. This enables a direct comparison between the evolved controller and a human driver, the results of which show that the performance of the evolved controller is roughly equal to that of the human driver when the human driver is allowed to see the car and the track directly. However, when the human driver is only allowed to see the car and the track through a camera with a latency comparable to that given to the evolved controller, the evolved controller completes the track markedly faster than the human driver.

III. The ADATE system for automatic programming

ADATE (Automatic Design of Algorithms Through Evolution) [27] is a system for evolving programs in a purely functional subset of the programming language Standard ML [28]. It is a general system that has successfully been applied to a wide variety of programming tasks, like evolving standard textbook algorithms from scratch [29], learning simple natural languages [30], improving existing classification algorithms [31] and evolving better neurons for automatic image segmentation [32]. To evolve a solution to a problem, the system needs a *specification file* that defines data types and auxiliary functions, a number of training and validation input examples, and an *evaluation function* that is used to grade and select potential solutions during evolution. Additionally, the specification file contains an initial program from which evolution will start. This program will be empty when evolving programs from scratch, but it is possible to start the evolution from any given program, for example to search for improvements for the best known program for a given problem.

The evolution in the ADATE system consists of the following overall steps.

1) Initiate the population with the single program given as the start program in the specification file. In addition to the actual programs in the population, the system also maintains an integer value C_P for each program P, called the *cost limit* of the program. For new programs this value is set to the initial value 100.
2) Select the program P with the lowest C_P value from the population.
3) Apply C_P program transformations to the selected program, yielding C_P new programs.
4) Try to insert each of the created programs into the population.
5) Double the value of C_P, and repeat from step 2.

The above loop is repeated until the user terminates the system.

As in all evolutionary systems, individuals with good fitness values are preferred, but in ADATE, the syntactic complexities of the individuals also play an important role, in that smaller programs are preferred over bigger ones. For a program to be allowed to exist in the population, it must therefore be better than all other programs syntactically smaller than it. This approach reduces the problems of overfitting since solutions highly specialized to a given set of training data tend to be syntactically larger than cleaner, more general solutions. As a consequence of this population management strategy, the population of ADATE has a dynamic size, as opposed to the fixed-size approach of most common evolutionary systems.

IV. The car simulator

The motion of the car is modeled by a set of non-linear differential equations based on rigid body dynamics. These are discretized in time and stepped forward by utilizing the second order Runge Kutta method. An implementation is written in Standard ML, and is used by the ADATE system in order to evaluate drivers as they are evolved.

The non-linear dynamical model is based on a single track model [33] which approximates a car by a bicycle like vehicle. The model is two dimensional by neglecting rotation about the horizontal axes, but includes the moment of inertia about the vertical axis. A non-linear tire model [34] is also utilized. It limits the maximum transverse forces on the wheels and lets the wheels slip in curves. The transverse slip combined with transverse forces yields a desirable dissipation of kinetic energy in curves.

The limitations to two dimensional rigid body dynamics and the single track model greatly simplify the dynamical model, but still retain the leading behavior of a race car with a low center of gravity. The simplifications also keep the simulation time down, which is of great importance to make a large number of training runs possible in a feasible time span.

Figure 1 depicts the single track modeled car and shows the parameters of the model. These are the distance between the wheels L, the distance from the rear wheel to the center of gravity L_{cg} and the steering angle ϕ. Three time dependent parameters fully represent the position of the car. These are the spatial coordinates $x_g(t)$ and $y_g(t)$ of the car's center of gravity, and the rotation angle $\theta(t)$.

By adding models of the forces acting on the car, a dynamic model based on rigid body dynamics can be constructed. The forces are the motor and breaking forces, the air drag and the lateral forces on the tires. The latter are considered functions of the tire slip angle α of the wheel (see e.g. the review of Schofield [35]). This is defined as

$$\tan\alpha = -\frac{u_n}{u_s}, \tag{1}$$

where u_n is the lateral velocity of the wheel and u_s is the velocity in the wheel's longitudinal direction. We let α_r

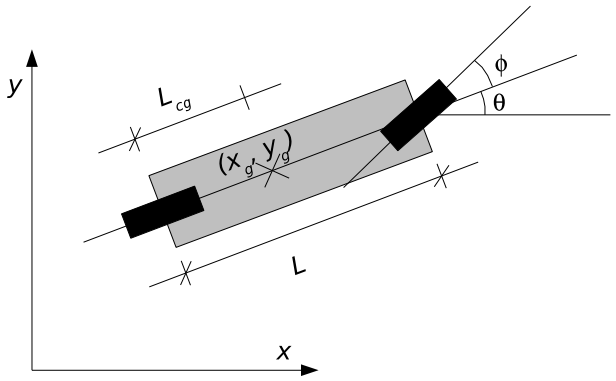

Fig. 1. The single track model. The model and its parameters are described in Section IV.

and α_f denote the slip angle of the rear and front wheel, respectively.

We consider the tire slip angles as expression of the velocity of the center of gravity, and obtain

$$\tan\alpha_r = -\frac{u_{gn} - L_{cg}\dot{\theta}}{u_{gs}}, \quad (2)$$

$$\tan\alpha_f = \phi - \frac{u_{gn} + (L - L_{cg})\dot{\theta}}{u_{gs}}, \quad (3)$$

where the dot represent time derivative, and u_{gs} and u_{gn} are the velocities of the car's center of gravity in the longitudinal and lateral directions, respectively. The tire slip angles are usually linearized by replacing $\tan\alpha$ with α (see e.g. [36], [35]), but we have chosen not to do this in our approach.

To obtain a realistic behavior at large tire slips, we utilize the so called magic formula [34]. The lateral force on each tire is

$$F_n = \mu F_z \sin\left[C \arctan(B\alpha - E(B\alpha - \arctan(B\alpha)))\right] \quad (4)$$

where μ is the friction coefficient, F_z is the vertical force on the wheel, $B = C_\alpha/(C\mu F_z)$ is a stiffness factor, C_α is called the lateral tire stiffness, and C and E are chosen parameters. To simplify, but still obtain the non-linearity of the model, we let $C = 1$ and $E = 0$. This yields the following non-linear tire model for the rear and front wheel, respectively

$$F_{rn} = \mu F_{rz} \sin\left(\arctan\left(\frac{C_\alpha \alpha_r}{\mu F_{rz}}\right)\right). \quad (5)$$

$$F_{fn} = \mu F_{fz} \sin\left(\arctan\left(\frac{C_\alpha \alpha_f}{\mu F_{fz}}\right)\right). \quad (6)$$

Note that these reduce to the linear model for small values of tire slip angles α_r and α_f, and for large values a pure frictional force μF_z.

We assume the car to have rear-wheel drive. A manipulating parameter $-1 \le q \le 1$ is introduced to model the driver's power control and breaking. A value of $q = 1$ means maximum motor power and $q = -1$ maximum breaking force. The motor is assumed to have a flat torque curve up to a maximum speed V_m. The maximum motor power is denoted by P_{max}. With a continuously variable transmission (CVT) the maximum possible force acting on the rear wheel due to the motor is then P_{max}/u_s for a forward speed of u_s. For velocities smaller than a selected limit, $u_s < V_0$, the force is P_{max}/V_0. The force may however be limited by the maximum friction force μF_z. Thus, a reasonable model for the force due to the motor for $0 \le q \le 1$ is

$$F_m = \begin{cases} \min(\mu F_z, qP_{max}/V_0) & \text{for} \quad u_s \le V_0 \\ \min(\mu F_z, qP_{max}/u_s) & \text{for} \quad V_0 < u_s \le V_m \\ 0 & \text{for} \quad u_s > V_m \end{cases} \quad (7)$$

For $-1 \le q < 0$, which means breaking, the force acting on the rear wheel in the car's longitudinal direction is modeled as $F_m = q\mu F_z$.

We model the longitudinal and lateral slips independently, which is a simplification motivated by the need for a very effective simulator. We could have used a friction circle model or a more advanced combined slip model such as the one in [37], which would have increased the physical realism of our simulation at the expense of longer run times. However, our goal in this paper is to demonstrate the usefulness of automatic programming for control systems in general and to do so with the limited computational resources at our disposal. Therefore, we chose to prefer computational speed over increased realism when modeling friction.

The air drag force on the car is

$$F_d = \frac{1}{2}\rho A C_d u_s^2 \quad (8)$$

where A is the cross section area of the car in the forward direction and C_d is the non-dimensional drag coefficient. For simplicity the drag coefficient is assumed constant.

The front wheel angle ϕ as defined in figure 1, is controlled by the ordinary differential equation

$$\dot{\phi} = \nu \tanh(K(s\phi_{max} - \phi))$$

where $-1 \le s \le 1$ is a manipulating variable, ϕ_{max} is the maximum value of ϕ and K is a chosen factor. The front wheel angle is considered a part of the car's state, and we may now formulate a set of ordinary differential equations for the system by utilizing the equations of rigid body dynamics for the car restricted to two dimensional motion. The mass of the car is in the following denoted by M and the inertia about the vertical axis through the center of mass by I_{zz}. The system reads

$$\frac{d}{dt}\begin{bmatrix} x_g \\ y_g \\ \theta \\ u_{gs} \\ u_{gn} \\ \omega \\ \phi \end{bmatrix} = \begin{bmatrix} u_s\cos(\theta) - u_n\sin(\theta) \\ u_s\sin(\theta) + u_n\cos(\theta) \\ \omega \\ u_n\omega + \frac{1}{M}(F_m(q) - F_d - F_{fn}\sin(\phi)) \\ -u_s\omega + \frac{1}{M}(F_{rn} + F_{fn}) \\ \frac{1}{I_{zz}}(-L_{cg}F_{rn} + (L - L_{cg})F_{fn}) \\ \nu\tanh(K(s\phi_{max} - \phi)) \end{bmatrix}. \quad (9)$$

The system is influenced by the two independent manipulating parameters q and s which reflect the driver's actions.

Parameter	Min value	Max value
Number of segments	1	100
Track width [m]	3	6
Length of segment [m]	100	200
Curve span [radians]	0	π
Initial speed [m/s]	20	40

TABLE I

MINIMUM AND MAXIMUM VALUES FOR GENERATED TRACKS.

The parameter values used in this paper are $M = 1500kg$, $I_{zz} = 2500kgm^2$, $L = 3m$, $L_{cg} = 1.5m$, $\phi_{max} = \pi/8$, $\nu = 1rad/s$, $K = 10/rad$, $P_{max} = 150kW$, $V_{max} = 60m/s$, $V_0 = 7.5m/s$, $\rho AC_d = 0.8kg/m$, $\mu = 1.0$, $g = 9.81m/s^2$ and $C_\alpha = 80000N/rad$. The time step used for the second order Runge Kutta method is $0.1s$.

V. EXPERIMENTS

Using the car simulator described above, we employed the ADATE system to evolve general and robust programs for driving a car as fast as possible on racing tracks. Each newly synthesized program was evaluated by letting it control the car in a simulation on a number of tracks, as described in the following sections.

A. Training and testing tracks

All tracks were generated automatically. The tracks have the following characteristics. Each track has a constant random width, and consists of a random number of segments. The first segment is always a straight segment, and the last segment is always a long, straight segment. When the car enters this last segment it is considered to have completed the track. All the other segments are curves with random directions and curve spans. We also use a random initial speed at which the car starts to race.

The parameters used for generating the random tracks are shown in table I. All random numbers are selected uniformly from the given intervals.

As a measure to avoid overfitting, a safety margin of 0.5 meters was used during training. If a driver ever got closer to the edge than this margin during simulation, it was considered to have driven off the track.

As will be explained in Section V-D below, an iteratively increasing set of training tracks was used during the evolution. After the experiments, the evolved drivers were tested on millions of new, freshly generated tracks not used during training.

B. Parameters and return values of the programs to evolve

In addition to specifying the tracks to use for training the drivers, we also needed to specify the sensor inputs available for the drivers, as well as the actions the drivers are allowed to perform.

Table II summarizes the sensor inputs we decided to use. Most of them are self-explanatory, but the following need some additional comments:

- β, the Rotation Slip Velocity, measures the spin of the car in excess of what should be expected based on the current speed and front wheel angle. It may be expressed as
$$\beta = \dot{\theta} - \frac{v}{L}\tan(\phi) \tag{10}$$
- The five α_N values are computed relative to the current heading of the car. The distances to the measurement points are selected as the braking distance of the car when driving at 10, 20, 30, 40 and 50 m/s, respectively.

Sensor input	Description
u_s	The longitudinal speed of the car [m/s]
u_n	The lateral speed of the car [m/s]
w	The road width [m]
d_c	The distance from the center of the car to the center of the road [m]
β	The Rotation Slip Velocity [rad/s]
ϕ	The angle of the front wheels relative to the rest of the car [rad]
	The angle to a point ...
α_{10}	...5.10 meters ahead of the car
α_{20}	...20.39 meters ahead of the car
α_{30}	...45.87 meters ahead of the car
α_{40}	...81.55 meters ahead of the car
α_{50}	...127.42 meters ahead of the car

TABLE II

SENSOR INPUTS AVAILABLE TO THE EVOLVED PROGRAMS

As described in section IV, the car is controlled by two parameters: The throttle and brake control q, and the steering control s. Correspondingly, we decided to evolve programs returning two floating point numbers. The returned values were clipped to the interval $[-1, 1]$, and fed into the car simulator as q and s for the next time step. Thus, at each time step, the program controls the car by directly manipulating the throttle/brake and steering controls.

C. Expressiveness of evolved programs

Even though ADATE has the ability to define new auxiliary functions on-the-fly, the effectiveness of its program synthesis may strongly depend on the set of predefined functions that it is allowed to use. For the experiments reported in this paper, we included addition, multiplication, subtraction and division of floating point numbers in this set and also the hyperbolic tangent function $\tanh$, which is commonly used in neural networks.

Since ADATE is able to effectively introduce and optimize floating-point constants on its own, there was no need to include any special, predefined constants.

The above set of predefined functions is a superset of what is needed to implement standard feed-forward neural networks with any number of hidden layers, which can express quite good approximations to any non-linear function [38]. Therefore, the same result holds for our evolved programs.

In practice, however, the limiting factor for most neural and evolutionary computation techniques is not the theoretical expressiveness of the languages that they employ but their ability to avoid entrapment in local optima in the search space. Another key limiting factor is overfitting. We believe that ADATE excels at both reducing overfitting and avoiding

local optima, but we do not have space here for a discussion of the many mechanisms employed to do so [27].

D. Iterative-deepening of the total training distance

In order to keep program evaluation times short, we started by using only 8 tracks randomly generated with the distribution described in Table I for evaluation of the evolved robot drivers. However, this limited driving experience turned out to be far from sufficient to guarantee generally safe driving among the fastest evolved drivers.

Therefore, we iteratively increased the number of training tracks by a factor of about 1.2 and each time restarted the evolution using the last population obtained during the previous iteration. Thus, the amount of training data for the robots increased gradually as they became more and more advanced drivers. We continued the iterative deepening of the number of tracks until a total of 210 tracks were used for training.

We then used a 0.25m margin and 16384 tracks to select a program from from all so-called parent programs, which are the ones considered good enough to be further transformed during ADATE runs. For our runs, the parent programs were about 0.1% of all generated programs. The selected program was then tested with zero margin by running the simulation continuously for several days on millions of new random tracks without any crash.

The initial 8 track iteration was allotted about two weeks on a cluster with 56 CPU cores. Using these 8 tracks, a single core performs about 37 simulations per second for an average driver. This means that about 2.5 billion evaluations of programs were performed altogether on the 56 cores during the two weeks. We then used between two and four days between restarts and altogether expended several months of wall clock time on the cluster.

VI. The evolved drivers

One of the advantages of the ADATE system over numeric approaches like neural networks is the ability for humans to read, understand and analyze the solutions found. The solutions given by the ADATE system are programs in a subset of Standard ML, and with some simple rewriting they can be made quite understandable for humans. In this section we will present two of the drivers evolved by the ADATE system. One of these drivers is quite slow, but has been tested for millions of tracks without driving off the track a single time. The second driver presented is the fastest but still safe driver, also being able to drive millions of tracks without crashing.

A. A simple, slow but safe driver

The control algorithm for one of the simplest but still safe drivers is given by the following equations for throttle control q and steering control s, restructured from the raw ML code created by the ADATE system for better readability:

$$q = 5\frac{w}{20.89 - u_s}$$

$$s = \alpha_{20}$$

This simple evolved driver never crashes, but then it does not drive very fast. It maintains an approximately constant speed of around 20-21 m/s which enables it to drive through all the curves it encounters on our training and testing tracks without skidding.

The driving strategy employed by this driver is very simple. The throttle algorithm simply tries to maintain a fixed constant speed at all times. To do this, it computes the difference between the speed of the car and 20.89, which seems to be its preferred "ideal speed" regardless of the situation. If the current car speed is lower than this, it applies throttle, and if the speed is higher, it applies brakes. The amount of throttle/braking is proportional to the road width w, and inversely proportional to the difference between the current car speed and the ideal speed.

The steering algorithm is also quite simple. It simply aims to steer towards the direction of α_{20}, that is, towards a point approximately 20 meters ahead of the car. Since the driver keeps a constant moderate speed of about 20 m/s, it never looses control of the car, and little or no skidding occurs. Consequently, the driver is able to keep the car near the center of the road at all times.

Figure 2 (a) shows how the front wheel angle ϕ varies during a sharp left turn. Note the slight delay of ϕ when α_{20} starts to increase. This is due to a simulated delay in the transmission from the steering wheel to the front wheels, and results in a slightly exaggerated steering at the beginning and at the end of the turn.

Many of the evolved individuals have found ways to avoid this slightly unstable steering, by using the current angle of the front wheels to adjust the requested steering, for example with the following formula:

$$s = 2\alpha_{20} - \phi \tag{11}$$

At the beginning of the turn, when ϕ is near zero, this formula steers the wheel sharper than the simple formula $s = \alpha_{20}$, and thus compensates for the mentioned delay. However, once the front wheels start to react, the steering is decreased slightly until a stable state is reached. This is illustrated in Figure 2 (b), where the car drives through the same turn, but this time using ϕ to adjust the steering parameter. As we can see, the front wheel angle increases faster in this case than in Figure 2 (a). Consequently, by using ϕ to adjust the steering parameter, the car follows the curve better, and the steering is smoother both at the beginning and at the end of the curve. In the following section we show that such a technique is employed by the steering algorithm of the best driver found by the ADATE system.

B. The best evolved driver

The control algorithm for the best driver found is given by the following equations:

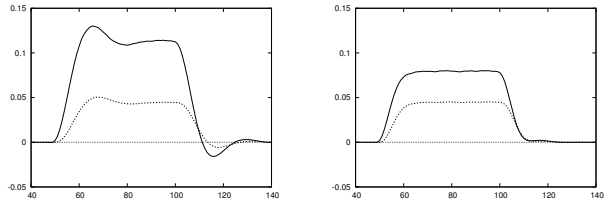

Fig. 2. Plots of α_{20} (solid line) and the front wheel angle ϕ (dotted line) during a turn. In **(a)**, the simple steering algorithm $s = \alpha_{20}$ is used, whereas **(b)** uses the slightly more sophisticated $s = 2\alpha_{20} - \phi$.

$$
\begin{aligned}
q &= \tanh\left(\frac{35.17 - u_s}{100\tanh(\tanh(u_s\alpha_{30}^2))} - (2.515 + d_c)\right) \\
s &= \frac{\alpha_{10} + \alpha_{20} - \phi}{w/20}
\end{aligned}
$$

The evolved driver has been tested on millions of tracks, driving several million kilometers without ever driving off the track. It maintains an average speed of about 30.5 m/s, that is, approximately 110 km/h.

The driver tends to brake down for a brief period in the beginning of curves, thereafter it often applies a high level of throttle and skids through the turn, with the car pointing inwards.

C. An analysis of the best evolved driver

We will now present an analysis of the behavior of the evolved driver, starting with the steering algorithm.

The two sensor inputs α_{10} and α_{20}, denoting the angle to points approximately 5 and 20 meters respectively ahead of the car, are added to guide the direction and magnitude of the steering. Additionally, the current front wheel angle ϕ is subtracted. As explained earlier, this reduces the delay of the front wheels relative to the steering wheel by exaggerating the control of the steering wheel at the beginning and end of turns.

Additionally, the steering algorithm depends on the width of the road, in that on narrow roads, the magnitude of the steering parameter s is higher than on wider roads. This seems like a natural behavior: On narrow roads, the steering must be more aggressive than on wider roads in order to keep the car on the track. On a wide road, the steering can be more relaxed.

The throttle algorithm is slightly more complex. The throttle to apply is computed by subtracting two terms, and then applying the $\tanh$ function on the result.

Let us first look at the second term, $(2.515 + d_c)$. This term is subtracted from the first term, thus, the driver will accelerate whenever the first term is greater than this second term. Note the addition of d_c, the current distance between the car and the center of the road. The higher this distance, the greater will the second term be. Consequently, the driver is more reluctant to accelerate and more willing to brake when the distance to the center of the road is high.

The first term,

$$\frac{35.17 - u_s}{100\tanh(\tanh(u_s\alpha_{30}^2))}$$

depends on the current velocity u_s of the car and on the angle α_{30} to a point approximately 45 meters ahead of the car. In the numerator, u_s is subtracted from the constant 35.17. As a result, if ever the speed exceeds this value, the value of the first term of the throttle algorithm, and thus the value of the entire throttle expression, will be negative. Thus, the car will always brake if its speed exceeds 35.17 m/s.

At speeds below this limit, the value of the first term decreases quickly when α_{30} increases. Additionally, the value decreases when u_s increases. Thus, at low speeds and relatively straight roads, the value of this term will be high, resulting in the driver accelerating. If the speed is high, or if the road ahead turns, the value of the first term will be lower, and if it is lower than the second term, the driver will brake.

Recall that the parameter α_{30} denotes the angle to a point on the road 45 meters ahead of the car. This angle is given relative to the current direction of the car. When testing our driver, we noted that in most curves, the driver oversteers the car, that is, the car points inwards throughout the curve. The more oversteer the driver applies, the lower the value of α_{30}, thus the first term of the throttle algorithm will be higher. When testing the driver, we have seen that this makes the driver apply throttle throughout most of the curves. When the car points inwards throughout the curve, the effect of this throttle is to generate a centripetal force in excess of what the friction of the tires is capable of, thus allowing the car to be kept on the road despite the high speed.

D. About the robustness of the evolved driver

As in all reinforcement learning contexts, the particular evolved solution is only trained and tested on situations that actually occur when using the driver to drive around a track. For example, the solution given in Section VI-B above tends to keep the car not too far from the middle of the road. Therefore, it may not be suitable for handling situations in which the car is close to the edge of the road. To test this, we ran two small tests on a single straight 6 meter wide track: One in which the car was initially placed 1 meter from the edge of the road, and another one in which the car was placed only 0.3 meters from the edge. The initial speed was set to 10 m/s in both cases.

The results are presented graphically in Figure 3. As can be seen in these figures, when starting the car 1 meter from the edge, the driver immediately steers rightwards towards the center of the road. As a result, the car crosses the center of the road, but the driver is quickly able to stabilize the car by applying a modest left steer.

In Figure 3 (b), however, the robot driver is driving only 0.3m from the edge of the road. It steers towards the center, resulting in the car crossing the center at a somewhat sharper angle than in the previous case. Thus, to stabilize, the driver needs to apply a sharper left steer, but this causes the car

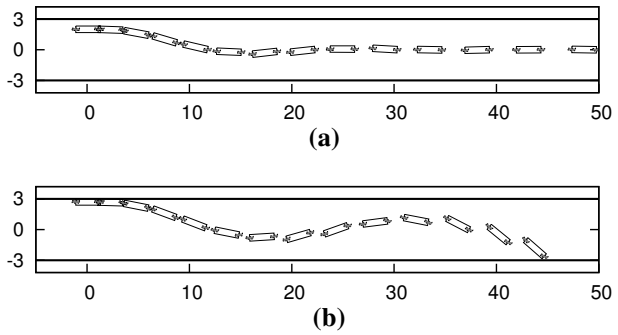

Fig. 3. Using 10m/s and **(a)** 1 meter and **(b)** 0.3 meters from the edge of the road as the initial state. The position, orientation and steering of the car is plotted every 2 iterations (that is, every 0.2 seconds)

to cross the center of the road with a too sharp angle again. Thus the driver needs to apply a sharp right steer, resulting in a rightwards spin After crossing the center again and approaching the right edge of the road, the driver is not able to regain control of the spinning car before it is too late and the car crosses the edge.

This behavior is somewhat similar to a so-called pilot induced oscillation (PIO) that is a well-known cause for accidents with fly-by-wire fighter aircraft.

In order to evolve a driver able to handle situations that do not occur during normal driving, such situations must be added as training examples. For example, our evolved drivers would probably be more robust if the lateral starting position of the car relative to the road was varied in the training examples. Additionally, other difficult situations could be added, like, e.g., entering a curve while the car is spinning in the wrong direction or approaching a curve at very high speeds. Such an approach is further examined in [15].

E. Comparing ADATE with an approach based on Evolution Strategies

In order to compare our results with those obtainable using other evolutionary optimization methods, we ran experiments using Evolution Strategies (ES) [39]. We used the same simulation and the same method for creating training and validation tracks as above, and used an ES to determine a set of coefficients for the following linear model.

$$q = c_0 + \sum_{i=1}^{11} c_i p_i \quad (12)$$

$$s = d_0 + \sum_{i=1}^{11} d_i p_i \quad (13)$$

As above, q and s are the throttle and steering control output of the controller. The parameters $p_1, \ldots, p_{11}$ are the same 11 inputs to the controller as those used in the ADATE experiments. The task of the ES is to optimize the c_i and d_i coefficients, that is, a total of 24 coefficients.

We used a standard $(\mu + \lambda)$ ES, with uniform discrete crossover and individuals encoding standard deviations used for mutations. Each individual in the initial population was initiated with all standard deviations set to 0.2, and coefficients randomly drawn from a normal distribution using this standard deviation.

μ	λ	Number of training tracks	Number of generations	Result
1000	7000	400	120	26.601
1000	7000	800	120	27.214
1000	7000	1600	140	26.880
1000	7000	3200	160	27.323
10000	70000	400	120	27.352

TABLE III
RESULTS FROM VARIOUS ES EXPERIMENTS. THE RESULTS GIVEN ARE THE AVERAGE SPEEDS OF THE BEST VALIDATED DRIVER IN EACH RUN.

We used a variety of population sizes and number of training tracks, as summarized in Table III, with a total run time of more than two weeks on a cluster with 84 CPU cores. After each 20. generation, the current population was validated using 16384 validation tracks.

Table III summarizes the results of our experiments, showing the average speed of the best validated individual on the 16384 validation tracks. As shown in the table, our ES experiments were not able to produce safe drivers driving faster than about 27 m/s, even when using very large populations.

Comparing these results with the results obtained by the ADATE system, we see that the controller evolved by the ADATE system drives the car 10% faster than the best ES-evolved controller. The total number of evaluations performed in the ES experiments were significantly lower than the number of evaluations performed during the ADATE experiments though, but we have no reason to believe that spending more computation time by e.g. using even larger populations would yield significantly better results. We therefore conclude that the ADATE system is capable of evolving controllers with significantly better performance than our ES-based approach.

VII. CONCLUSIONS AND FURTHER WORK

In this paper, we have shown that if a simulation and some method for creating training examples are available, the ADATE system is capable of evolving sophisticated, robust and effective control algorithms. Our domain has been the evolution of automatic drivers for a racing car, but our methods are likely to be applicable to most automatic or semiautomatic control problems.

Section VI-E above presented a comparison between our approach and an ES based approach. The ES based approach was only able to evolve linear drivers, whereas our ADATE system is not restricted like this to any predefined solution structure. An important area for future research is to compare our results with those achieved using other, more general evolutionary paradigms, e.g., Genetic Programming.

As shown in Section VI-D above, our evolved driver is not generally able to recover from critical situations that did not occur during training. One possible area of further research

may be to evolve a more robust control algorithm by using a large number of such critical situations as training inputs. In the near future, such an approach may be able to evolve general and robust semi-autonomous systems for avoiding accidents in commercial cars.

To have any practical use, an autonomous driver should also be able to control the car in the presence of stationary and moving obstacles in the road, based on inputs from inaccurate or noisy sensors. In the near future we believe it should be possible to evolve drivers with these abilities using our approach, by adding obstacles to the tracks used for training, and by adding random noise to the sensor inputs given to the evolved individuals.

One may also start the evolution from some already existing control system in order to optimize the system by evolution. We showed the potential use of this approach in [32], where neurons used for image segmentation were automatically evolved based on a standard neuron model from the literature. It is likely that such an approach would also yield good results for problems within the field of automatic control.

REFERENCES

[1] P. A. Ioannou and C. C. Chien, "Autonomous intelligent cruise control," *IEEE Transactions on Vehicular Technology*, vol. 42, no. 4, pp. 657–672, 1993.

[2] J. Ackermann, T. Bnte, and D. Odenthal, "Advantages of active steering for vehicle dynamics control," in *Proceedings of the International Symposium on Automotive Technology and Automation*, 1999.

[3] E. Bertolazzi, F. Biral, and M. D. Lio, "Future advanced driver assistance systems based on optimal control: The influence of "risk functions" on overall system behavior and on prediction of dangerous situations," in *IEEE Intelligent Vehicles Symposium*, 2004, pp. 386–391.

[4] A. Balluchi, L. Benvenuti, M. D. di Benedetto, C. Pinello, and A. L. Sangiovanni-Vincentelli, "Automotive engine control and hybrid systems: Challenges and opportunities," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 888–912, 2000.

[5] R. D. Filippi and R. Scattolini, "Idle speed control of a f1 racing engine," *Control Engineering Practice*, vol. 14, no. 3, pp. 251–257, 2006.

[6] J. Marzbanrad, G. Ahmadi, H. Zohoor, and Y. Hojjat, "Stochastic optimal preview control of a vehicle suspension," *Journal of Sound and Vibration*, vol. 275, no. 3-5, pp. 973–990, 2004.

[7] Honda, "The all-new honda accord," Press-release, published at `http://world.honda.com/news/2008/4080211all-new-Accord-for-Europe/`, 2008.

[8] H.-P. Schwefel, *Evolution and Optimum Seeking*. New York: Wiley and Sons, 1995.

[9] T. M. Mitchell, *Machine Learning*. Singapore: McGraw-Hill, 1997.

[10] D. Stavens, G. Hoffmann, and S. Thrun, "Online speed adaptation using supervised learning for high-speed, off-road autonomous driving," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2007.

[11] S. T. et al, "Stanley: The robot that won the darpa grand challenge," *Journal of Robotics Systems*, vol. 23, no. 9, pp. 661–692, 2006.

[12] "The DARPA Grand Challenge '05," Web-page: `http://www.darpa.mil/grandchallenge05`, 2005.

[13] S. Haykin, *Neural Networks - A Comprehensive Foundation (2nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, 1999.

[14] D. Pomerleau, "Knowledge-based training of artificial neural networks for autonomous robot driving," in *Robot Learning*, J. Connell and S. Mahadevan, Eds., 1993.

[15] M. Jakobsen, "Learning to race in a simulated environment," Master's thesis, stfold University College, 2007.

[16] L. D. Pyeatt and A. E. Howe, "Learning to race: Experiments with a simulated race car," in *Proceedings of the Eleventh International Florida Artificial Intelligence Research Society Conference*, 1998, pp. 357–361.

[17] M. Carreras, J. Yuh, J. Batlle, and P. Ridao, "A behavior-based scheme using reinforcement learning for autonomous underwater vehicles," *IEEE Journal of Oceanic Engineering*, vol. 30, no. 2, pp. 416–427, 2005.

[18] B. Freisleben and T. Kunkelmann, "Combining fuzzy logic and neural networks to control an autonomous vehicle," in *Proceedings of the IEEE International Conference on Fuzzy Systems*, 1993, pp. 321–326.

[19] S. Huang and W. Ren, "Use of neural fuzzy networks with mixed genetic/gradient algorithm in automated vehicle control," *IEEE Transactions on Industrial Electronics*, vol. 46, no. 6, pp. 1090–1102, 1999.

[20] D. Partouche, M. Pasquier, and A. Spalanzani, "Intelligent speed adaptation using a self-organizing neuro-fuzzy controller," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2007, pp. 846–851.

[21] J. Togelius and S. M. Lucas, "Evolving controllers for simulated car racing," in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, 2005, pp. 1906–1913.

[22] ——, "Evolving robust and specialized car racing skills," in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, 2006, pp. 1187–1194.

[23] A. Agapitos, J. Togelius, and S. M. Lucas, "Multiobjective techniques for the use of state in genetic programming applied to simulated car racing," in *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, 2007, pp. 1562–1569.

[24] H. Marques, J. Togelius, M. Kogutowska, O. Holland, and S. M. Lucas, "Sensorless but not senseless: Prediction in evolutionary car racing," in *Proceedings of the 2007 IEEE Symposium on Artificial Life*, 2007, pp. 370–377.

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

[26] I. Tanev and K. Shimohara, "Evolution of human competitive driving agent operating a scale model of a car," in *Proceedings of the SICE Annual Conference 2007*, 2007, pp. 1582–1587.

[27] R. Olsson, "Inductive functional programming using incremental program transformation," *Artificial Intelligence*, vol. 1, pp. 55–83, 1995.

[28] S. Weeks, "Homepage of the MLton Standard ML compiler," Webpage: `http://www.mlton.org`, 2007.

[29] R. Olsson, "ADATE — Automatic Design of Algorithms Through Evolution," Web-page: `http://www-ia.hiof.no/~rolando/adate_intro.html`, 2006.

[30] R. Olsson and D. Powers, "Machine learning of human language through automatic programming," in *International Conference on Cognitive Science*, 2003.

[31] S.-E. Hansen and R. Olsson, "Improving decision tree pruning through automatic programming," in *Proceedings of the Norwegian Conference on Informatics (NIK-2007)*, 2007, pp. 31 – 40.

[32] H. Berg, R. Olsson, T. Lindblad, and J. Chilo, "Automatic design of pulse coupled neurons for image segmentation," *to be published in Neurocomputing - Special Issue on Neurocomputing for Vision Research*, 2008.

[33] P. Riekert and T. Schunck, "Zur fahrmerchanik des gummibereiften kraftfahrzeugs," *Ingenieur Archiv*, vol. 11, pp. 210–224, 1940.

[34] H. B. Pacejka, *Tyre and Vehicle Dynamics*. Butterworth Heinemann, 2002.

[35] B. Schofield, "Vehicle dynamics control for rollover prevention," Ph.D. dissertation, Lund University, 2006.

[36] M. Egerstedt, X. Hu, and A. Stotsky, "Control of a car-like robot using a dynamic model," in *IEEE International Conference of Robotics & Automation*, 1998.

[37] J. Svendenius and M. Gfvert, "A semi-empirical tire-model for transient combined-slip forces," *Vehicle System Dynamics*, vol. 44, no. 2, pp. 189–208, 2006.

[38] T. M. Mitchell, *Machine Learning*. McGraw-Hill Companies, Inc., 1997.

[39] H.-P. Schwefel, *Evolution and Optimum Seeking*. John Wiley and Sons, Inc., 1995.