EVOR : An Online Evolutionary Algorithm for Car Racing Games

Samadhi Nallaperuma
School of Computer Science
The University of Adelaide
Adelaide, Australia

Frank Neumann
School of Computer Science
The University of Adelaide
Adelaide, Australia

Mohomad Reza Bonyadi
School of Computer Science
The University of Adelaide
Adelaide, Australia

Zbigniew Michalewicz
School of Computer Science
The University of Adelaide
Adelaide, Australia

ABSTRACT

In this paper, we present evolutionary racer (EVOR) that is a simulated car dynamically controlled by an online evolutionary algorithm (EA). The key distinction between EVOR and earlier car racing methods is that it considers car racing as a dynamic optimization problem which is addressed by an evolutionary algorithm. Our approach calculates a car trajectory based on a controller decision and adjusts this decision according to the suitability of its resultant trajectory with the current track status. Furthermore, it allows to integrate features such as opponent handling implicitly. Our experimental results show that EVOR outperforms current best AI controllers on a wide range of tracks.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence

Keywords

Dynamic Optimization; Online Algorithms; Simulated Car Racing

1. INTRODUCTION

Simulated car racing is a widely popular computer game. The open racing car simulator (TORCS) race engine [4] provides a good testbed to test car racing algorithms. The simulated car racing (SCR) championship series [9**?**] have enabled further expansion of these capabilities specially for artificial intelligence (AI) approaches. In recent years, there has been a boost of AI based car racing algorithms implemented on the TORCS framework. Rule based controllers [7, 10, 16], fuzzy inference systems [13, 15], and artificial neural networks [5, 6] are some examples of the existing methods for simulated car racing.

GECCO'14, July 12-16, 2014, Vancouver, BC, Canada.

There are already some car racing methods in which the controller learn using an evolutionary algorithm though an off line process [13, 16**?**]. However, due to the optimization capabilities, evolutionary algorithms have great potential to be the core of the controller rather than an offline learning approach. Using an EA method as a car racer not only enables the controller designers to reduce the need for domain knowledge, but also reduces computational overhead in the pre-processing phase (e.g. neural network-based controllers) or parameter tuning phase. Note that preparing domain knowledge is not an easy task specially if there are not enough/accurate information available about the environment. It is well-known that fuzzy systems lead today's general controller design in computer games domain [13] as well as real-life control systems (such as cruise controllers [12]). EA can be a good alternative to this approach.

Controlling a car can be viewed as a dynamic optimization problem and EAs have been widely applied to dynamic optimization [**?**]. In fact a driver needs to apply optimal steer, acceleration, clutch, brake and gear controller values dynamically to drive fast and safe. In case of driving, the problem environment is represented by a diverse set of factors including car position, orientation, speed, current road condition as well as other environmental factors such as wind, rain and obstacles. All these factors are changing over the time deviating the current optimum from the previous optimum (controller values) to a new position in the search space. Accordingly, it would be beneficial to take these dynamic factors into account and try to find optimal solutions during the run. Simply this is the scenario almost any generic dynamic optimization process encounters. In this study, we investigate how dynamism in a car controlling environment can be addressed by an online evolutionary algorithm.

In an EA based car controller, the actuator values can be represented as the genes of the individual. Then the natural optimization problem would be to find the best combination of the controller values under dynamic circumstances. A feasible evaluation criteria is based on how much a car trajectory resultant from the current controller values in an individual would fit the current track. This strategy is simple to realize because it does not rely on heavy domain knowledge. Also, in such an approach, unlike other car racing approaches that need special mechanisms to handle opponents [1, 16], there is no need for such extra specialized modules. Instead, a fitness based strategy could implicitly optimize the trajectory con-

sidering opponents as another level in the fitness criteria. As a proof of concept, we present an implementation of a simulated driver EVOR consisting an evolutionary optimizer for its controller. We conduct a series of experiments to investigate the performance of this new driver on a diverse set of standard benchmark tracks. For the majority of the cases it outperforms existing best AI based drivers.

The organization of the paper is as follows. Section 2 describes strengths and limitations of existing AI approaches and also the widely used mechanisms in physical controllers. Section 3 explains the core of our evolutionary dynamic optimizer and section 4 describes additional features integrated to enhance overall performance. In Section 5 we conduct a series of experiments and analyses of the results. Finally, Section 6 concludes with the highlights and future directions.

2. BACKGROUND

The TORCS framework [4] is one of the most famous frameworks for simulated car racing. TORCS provides a free game engine with standard features such as sensory information, aerodynamics, friction and 3D graphics. The TORCS framework is used for many racing games and competitions. A customized version of TORCS was provided by the simulated car racing (SCR) championship [9] specially aiming for building AI controllers. This version uses a client server architecture where client bots are connected to the related server bots. The communication between clients and the server takes place through the user datagram protocol (UDP) socket connection. Researchers can develop their own drivers with the interface provided by the SCR client.

Current simulated car racing approaches include expert systems [7, 10, 16], artificial neural networks [5, 6] and fuzzy expert systems [13, 15]. Among these approaches Autopia [13] driver has won a great popularity due to its performance in simulated car racing (SCR) championship series [9] in recent years. The significance of Autopia is that it works equally well under different road conditions. The fuzzy controller in Autopia enables its smooth handling under different conditions. Mr.Racer [16], the winner of SCR championship at GECCO 2013 [9] is also a popular driver. This driver is based on an expert system as the core of the controller. Mr.Racer constructs a track model during its initial run on a new road. The runner up ICER IDDFS of the same competition is also an expert system. Some approaches incorporate evolutionary strategies to tune the parameters used in their formulas[16][13] or genetic programming to learn the controller program [**?**]. In both these cases the evolutionary process takes place in off-line mode that is prior to the actual driving/racing.

Although EAs are used for optimizing parameters and for learning the controller program in some current approaches, the core of these controllers are expert systems [16] or other AI approaches like fuzzy inferencing [13] or artificial neural networks [5]. None of these approaches have considered the possibility to use an evolutionary algorithm as the core of the online decision making process rather than to use it only for parameter tuning. By using an evolutionary algorithm as an online decision maker, there is no need for heavy domain knowledge (as for expert systems), pre-processing (as for neural networks), or tedious parameter tuning (as for off line tuning based controllers) any more. Similar kind of evolutionary dynamic optimizers have been used in decision making for other real time games and proves to successful [**?**].

There are simple and efficient control mechanisms used in physical automobiles on top of the raw actuator outputs, such as advanced brake system (ABS) [14] and traction control mechanisms [14] [3]. We can integrate these techniques to the evolutionary controller outputs to enhance the overall performance. Another popular feature is the auto gear system used in real vehicles at present. We can introduce this feature to the simulated environment also by introducing a direct mapping [3] of gear values to the engine rpm.

3. APPROACH

Our controller is based on an evolutionary algorithm having a population of one individual. The single individual of the EA represents the acceleration, brake and steer actuator values. Our driver is designed to have an auto gear system, therefore, we do not optimize gear dynamically. Instead, gear is applied through a direct mapping of the engine rpm (round per minute) to the gear. The clutch is also not included in this initial version of the driver considering its less relevance for the basic performance. Nevertheless, adding another control value is simplified with our algorithm. In fact, a new actuator introduces merely a new gene to the individual. The significance of the individual we use here is that it contains a set of heterogeneous genes having different ranges opposed to the standard binary or real valued vectors generally used in EA's. Hence, the standard crossover operators cannot be applied here. We use uniform mutation as the only variation operator.

As shown in Algorithm 1 the evolutionary optimization process runs forever. Once a new car state is received from the sensors the algorithm adapts the optimization process accordingly. At the end of a certain time interval the data in the current individual is converted to the actual controller values and applied to the car. In computer game environment this interval means the simulation server timeout. This asynchronous behavior is realized through a multi threaded model consisting separate threads for the EA and the input/output (I/O) communication. In practice this algorithm provides robust dynamic car controlling that could either be used in autonomous vehicles or simulated car racing. Let us discuss the components of the evolutionary algorithm in detail.

Algorithm 1 (1+1)EA

```
x ← an individual with a random set of controller values
repeat
    y ← Mutate(x)
    if f(y) ≥ f(x) then
        x ← y
    end if
until forever
```

The individual represents the set of actuators of the car including steer, acceleration and brake. Acceleration and brake are represented by a single floating point variable that has a continuous data range of -1 to +1 in which positive values represent acceleration and negative values represent brake. The floating point variable representing steer lies in a continuous range of -90 to +90 degrees. The actual steering controller values may stay in a more restricted range depending on the car. We consider this wide range for optimal steering. If the value of a variable, generated by the algorithm, exceeded its feasible range then it is mapped into the closest

value in the feasible range. At the timeout of the server, a car controller output is created by converting the data in the individual to relevant actuators. The fitness of an individual is mainly based on how much it suits the current environment. First, We approximate the car trajectory resultant from the control values in the current individual applied to the current car status. Then this trajectory is evaluated based on the track. Here, the trajectory consists of two major factors: (1) the trajectory direction which is dependent on the optimal path and, (2) the velocity that is dependent on the current trajectory direction.

The strength of this fitness function is that the mutual dependency of the applied steer and the applied acceleration is implicitly addressed unlike most of other AI approaches where these actuator values are determined separately. In this model, a combination of these actuators are considered at once as they are contained within the individual. Hence, the acceleration is evaluated dependent on the steer value implicitly as the steer is considered in the trajectory direction for which acceleration is evaluated. These concepts will be discussed in detail within the trajectory evaluation criteria.

3.1 Car Trajectory Approximation

The car trajectory is estimated using basic physics [14] on motion. The trajectory is the expected path of the car defined by a particular set of controller values and a car state. We have considered two models for the trajectory estimation: the actual circular trajectory based on car dynamics and a linear approximation for the trajectory. For the circular trajectory, the radius of the circle is determined based on the centripetal force resultant from the lateral forces of the front and the rear wheels. The lateral force is formed in the opposite direction to the lateral velocity of the vehicle. The center of the circle is found using the current position of the car, the radius and the direction of the lateral forces. When the car is on a straight line this radius is infinite. Given the centripetal force F, velocity V and mass of the car M, using $F = MV^2/r$ the radius of the trajectory is calculated.

The centripetal force is formed by a combination of the lateral forces for front and rear wheels of the car and is denoted by $F_{centriprital} = F_{lat,rear} + F_{lat,front} \cdot \cos(\delta)$ where δ representing the steering angle. These lateral forces are dependent on the slip angle α of the tires. For smaller angles the force has a linear correlation with the slip angle hence can be calculated by $F = c\alpha$ where c represents the cornering stiffness. Slip angles are calculated based on the lateral and longitudinal velocities of the car V_{lat}, V_{long}, the distances from the front (b) and the rear (c) axles to the center of gravity of the car, the angular motion ω and the steering angle δ. For the front wheels this angle is calculated using the formula $\alpha_{front} = \arctan[(V_{lateral}+\omega \cdot b)/V_{long}] - \delta \cdot sgn(V_{long})$ and for the rear wheels using $\alpha_{rear} = \arctan[(V_{lateral} - \omega \cdot c)/V_{long}]$. This actual trajectory is defined with the lateral and longitudinal velocities of the car, the angular motion, the size and the mass of the car and the steering angle. For the considered simulation platform, the mass and the dimensions of car are not available. Therefore, we use initial assumptions and utilize a low pass filtering technique to update the mapping function of the centripetal force, for a given slip angle. For a comprehensive description on this physics model we refer the interested reader to the textbook of Pacejka [14].

Our second model, the linear trajectory, is an approximation to the actual circular car trajectory. In this model, we find a straight line based on the car steering direction α and the current car position $P(x_p, y_p)$ using $Y - y_p = tan\alpha \cdot (X - x_p)$. This is based on the observation that for a small distance, a fraction of the circumference of a circle is close to a line. This approximation works well because, in practice, only the nearest fraction of the track is critical on the controlling decision. The initial implementation of EVOR performs well with both models although we maintain the second model due to the simplicity, adequacy for the game environment, and the minimal dependency on the domain knowledge.

3.2 Construction of the Track Model

When a driver is set on a track initially it can learn the track model and then use this model to support further driving on the same track. Usually, in both simulated and physical racing competitions, a brief warm up session is provided before the race [8]. When set on a new track, EVOR drives slowly for two laps to build the track model during the warm up phase. EVOR is capable of constructing the track model during the race even if there is no specific warm up phase provided.

The track model is comprised of line segments perpendicular to the axis or the heading direction of the original track, positioned on the track per each meter. This design enables our fitness calculation based on the trajectory intersection with these track segments. For each sensory reading, the segments related to the visible section of the track are calculated. These new segments are then translated and rotated to merge with the existing track segments. To calculate these segments, first the right and left boundaries of the visible track are estimated using the sensor information [1]. Then, relative to the current position of the car, the perpendicular segments connecting the right and left boundaries of the track are identified. This process is repeated for each meter of the visible section of the track, thus, constructing a segment per a meter. Using the information of the current distance from the track start position which is considered as the reference point of the coordinate system and the current rotation angle of the car relative to the axis of our coordinate system, these relative segments are merged to the existing track as absolute track segments. We use a single rotation and a translation to merge a new (or repeating) segment to the existing track model.

Additionally, the track is annotated with the bend information. Straight areas, small and sharp bends are classified based on the curvature of the track. The curvature is calculated by aggregating the heading changes of consecutive track segments. This bend information is useful in determining the optimal acceleration on a turn. A sample track model built using this mechanism is shown in Figure 1

3.3 Trajectory Evaluation

Fitness for the considered trajectory is incremented for each track segment that intersects with the trajectory as shown in Figure 2. For each of these intersecting track segments, the fitness is incremented only if the intersect point lies within the left and right boundaries of the track. As shown in Algo-

[1]Note that the particular sensor distribution can be a property of the physical car or the simulation environment in case of gaming. For example, in the SCR default setting these sensors sample the space in front of the car every 10 degrees, spanning clockwise from -90 degrees up to +90 degrees with respect to the car axis

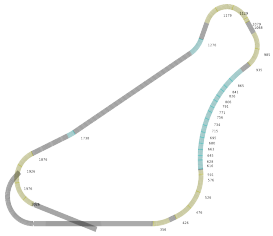

Figure 1: a sample track model

Algorithm 2 fitness function

```
for each visible track segment s do
    if trajectory intersects s at p inside the track then
        fitness += distance from p to the closest track boundary
        if maximum distance possible to stop < safe distance then
            fitness += speed at safe distance
        else
            fitness -= speed at safe distance
        end if
    end if
end for
```

rithm 2, this fitness value is calculated based on the distance from the closest boundary to the intersection point.

As stated in the latter part of the Algorithm 2 we consider the trajectory based fitness for the acceleration also. First we calculate the distance to the intersection point where the current trajectory collides with one of the left or right boundaries of the track (possibly in a bend). Let us call this *safe distance*. This can be approximated using physics formulas on linear motion. Similarly, the *stop distance*, within which the car can be stopped if brake is applied to the current car status, is calculated. If the *safe distance* is larger than the possible *stop distance* this means the car can have a higher speed, still managing to stop before a collision, therefore we prefer to accelerate more. In this case fitness is incremented based on the calculated speed at the *safe distance*(i.e. the higher the speed is, the larger the fitness increment will be). In contrast to this scenario, if we are already too fast and close to the track boundary (possible collision), we prefer to brake and, hence, fitness is decremented based on the speed at the collision point (the higher the speed, the larger the fitness decrement).

3.4 Mutation

Our mutation operator is specialized for the heterogeneous genes in the individual. As explained in the beginning of the section 3, each individual consists of two different genes with varying data ranges. Therefore, we define two variations of the uniform mutation operator specialized for each gene. At a given mutation step the algorithm selects a mutation point uniformly at random. Then the specific mutation operator for the gene is applied. We have considered two standard mutation strategies relevant to our problem representation and the best is selected subsequently.

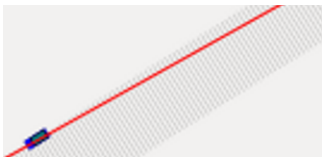

Figure 2: trajectory (long line in red) intersection with the track segments (short lines in grey)

For steer we use uniform mutation that assigns a random value generated within the continuous range from -90 to +90. We have also studied another alternative a nonuniform mutation with a fixed distribution. This mutation changes the current value slightly by adding a small variable drawn from a Gaussian distribution. As observed experimentally, this operator had a tendency to make the algorithm converge quickly to a local optimum without exploring the search space. In contrast, uniform mutation could preserve the global exploration of the search space and enable the algorithm to converge within a reasonable time. For acceleration also we have considered the same uniform mutation operator with a different data range of -1 to +1.

4. ADDITIONAL FEATURES

In this section we describe the additional features introduced to enhance the overall performance of our evolutionary controller.

4.1 Opponents Handling

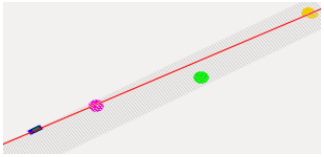

Figure 3: car (rectangle in blue) trajectory (long line in red) intersection with obstacles (circles in colors)

The opponents in front are considered as obstacles with which intersection would reduce the fitness of a trajectory. Based on this obstacle avoidance strategy the trajectories that do not intersect with the obstacles are preferred over the trajectories that intersect with the obstacles. For each intersecting obstacle the fitness is then reduced in terms of the distance from the car to the collision point as shown in Algorithm 3. For static obstacles when they are intersecting with the car trajectory closer to the current position of the car, fitness is reduced for that trajectory more. For example, as shown in Figure 3 the obstacle that intersects with the trajectory closer to the car position would reduce the fitness more than the other obstacle intersects with the trajectory at a greater distance in front. As per the moving obstacles the relative distance is dependent with the opponent's velocity. Hence, only if the opponent velocity is less than ours it is considered as a potential obstacle, otherwise it will not be colliding anyway.

4.2 Fitness Adjustments for More Efficient Turning

The basic approach on fitness evaluation for acceleration discussed in section 3 is safe although it is not very efficient. Therefore, we introduce a slight adjustment to the fitness evaluation criteria for acceleration to increase the considered safe distance further where the actual distance is large or decrease it further where the actual distance is already small.

Algorithm 3 fitness based on opponents

```
for opponent o detected do
    if o intersects and o's relative speed is less than 0 then
        dist := find the distance to collision point from current car position
        fitness -= (dist / max opponent car size)
    end if
end for
```

This adjustment can mitigate any error due to track model building. We use two linear functions to adjust the safe distance based on the sharpness of the bend at the intersection point (in Algorithm 4) and the calculated safe distance value (in Algorithm 5). This adjustment changes the safe distance used in the original fitness function (in Algorithm 2). Then the last two lines of the basic fitness function is replaced by the Algorithm 6

Algorithm 4 angle based adjustment factor

$m := ((1.0 - 1.5)/(\pi/4))$
$c := 1.5$
turn angle α at wall := find turn angle from track
angle based adjustment factor := $m \cdot \alpha + c$

Algorithm 5 distance based adjustment factor

$m := (0.5/100)$
$c := 0.5$
distance based adjustment factor := $m\cdot$ distance to wall $+c$

4.3 Stuck Management

Since EVOR always tries to find an effective trajectory, a stuck scenario is rather rare. Nevertheless, we have taken stuck management into account as it is still probable in the dynamic environment specially when racing against offensive opponents. Considering the small frequency of this scenario it is not covered in the basic EA used for driving. Alternatively, we incorporate stuck management into our evolutionary optimization model through another EA to optimize actuators on stuck situations. This EA is asleep usually, once stuck situation is observed it is awakened and the program control switches from the regular optimization process (driving EA) to this specialized EA. Once again, when the car is in safe position, this EA goes back to sleep mode saving its current stuck recovery controller values in its current parent individual. Then the regular EA runs and once again may call the stuck EA upon a stuck situation. Then the stuck EA resumes the optimization process from this saved individual.

If the car is at the same position without a velocity for a specified time we consider the car is stuck. For stuck management we use a simple fitness strategy. Here, the gear is fixed to be on reverse. The fitness is evaluated based on how much the car position is close to the track center and the car heads straight front. The design of this EA is similar to the basic EA used for regular driving except for this fitness criteria.

4.4 Auto Gear System, Traction Control and Advanced Brake System (ABS)

Algorithm 6 fitness adjustment for efficient turning

```
safe distance d := find distance to wall
d *= angle based adjustment factor
d *= distance based adjustment factor
if distance to stop < d then
    fitness += speed at d
else
    fitness -= speed at d
end if
```

We incorporate some expert knowledge on standard vehicle controller design to our evolutionary driver where appropriate. The auto gear system is an example, that we can use the popular mapping on optimal gearing based on engine rpm [3]. For traction control also we use a few standard rules used in car physics [3, 14]. Similarly we apply ABS on top of the already chosen brake value from the optimizer. The details about these techniques are not discussed here, since its not of the major interest of this study. We refer the interested reader to the text books [3] and [14] for more information regarding these techniques.

4.5 Convergence and Runtime of the Algorithm

As described earlier, the EA runs forever. Based on our implementation the number of possible fitness evaluations between two consecutive status updates is around 2000. While a global convergence is not achieved within this time interval, a solution of close approximation from the global optima is always achieved. Note that a dynamic change does not cause re-optimization but only a reevaluation of the current individual immediately after the change. A typical run of the algorithm is observed to be effective. Furthermore, rigorous theoretical analysis is essential to obtain a deeper understanding on the underlying dynamic optimization process and its effectiveness.

5. EXPERIMENTAL ANALYSIS

As a proof of the concept that a simple EA works well as a car controller we have implemented a prototype version of EVOR to run on a simulated environment. We use a customized version of the TORCS [4] simulator with limited sensor information that is used in SCR competition [9]. We believe this setting is more realistic because in physical driving environment also, we have access to sensor information on the visible area of the track only. We developed EVOR as a client communicating with a SCR-server bot [8].

We have conducted a series of experiments to compare the performance of EVOR against other AI approaches. For that, we have chosen 3 other best competitors of SCR championship 2013 [9] Autopia [13] the all time winner, Mr Racer [16] the winner of SCR championship at GECCO 2013 and the runner up ICER IDDFS of the same competition. We have analyzed the behavior of these drivers for several groups of tracks oval, road and dirt provided by the TORCS simulator and also for a set of random tracks generated using the track generator by Cardamone et al. [2]. The size and the physical factors of the tracks in the first three classes have major differences.

We have evaluated our controller based on an established experimental set up inspired by the SCR competition [9]. This

competition set up consists of 3 stages the warm-up, the individual race and the final race. Similar to the competition, for our experiments also 100000 game ticks are provided for the warm up session. During this stage drivers can build track models or tune any parameters. Then the cars race individually for 10000 game ticks. This time is approximately equal to 200 seconds as a game tick is around 20ms. In the final stage the drivers race together on each track for 10 laps. The starting positions of the competitors are determined according to the results of the individual race stage. Running on several laps means the experiment is repeated for several times, thus guarantees the fairness of the results like any other standard experimental setup.

Experiments were conducted in a Ubuntu Linux 12.04 PC with 64 bit Intel i5 $2.40GHz$ quad-core processor and $3.8GB$ memory. The experiments were run on SCR [8] patched version of TORCS 1.3.4 under the default mode. In the default mode the damage, the lap-time and the fuel limits are enabled.

5.1 Oval Tracks

Oval tracks represent speedways. The shapes and slopes slightly differ in the oval tracks provided with the TORCS simulator. Road friction, aerodynamic and other physical effects are quite similar in all oval tracks. The results of the four drivers raced individually on the oval tracks are shown in Figure 4. These results show that for 4 out of 5 oval tracks EVOR has bridged the largest distance within the given time budget of 10000 game ticks. It is evident that EVOR's trajectory optimization has provided the better path and the speed for these tracks than other drivers.

5.2 Road Tracks

The 15 road tracks shown in Figure 4 are diverse. They are significantly different to each other with regard to many aspects unlike oval or dirt tracks that share similar conditions within the group. The selected set of road tracks represent all different sub groups within the road group provided with the TORCS simulator. The road tracks can be of various shapes, widths, as well as of road friction. For example, the road friction may vary significantly from paved roads like *forza* to snowy slippery roads like *alpine-2*. And also the curvature of bends varies from mild bends in *forza* and *g-tracks* to large bends in *alpines* or sharp bends in *corkscrew*. EVOR has traveled the largest distance for the majority of the road tracks. For some tracks like *forza*, *wheel-2* and *g-track-2* having good road friction and moderate bends Mr.Racer was better. The major reason is that unlike Mr.Racer, EVOR was not specially tuned for specific friction parameters. Furthermore, for two tracks with a very sharp bend, *corkscrew* and *street-1* Autopia was the best. In this kind of sharp triangle shaped bends EVOR's race line optimization becomes weak. However, this kind of bends are rather unlikely in natural tracks. For all other road tracks including real tracks like *alpines*, *ruudskogen* etc., EVOR has reported the longest traveled distance.

5.3 Dirt Tracks

Dirt tracks are different from the other two groups with their sandy road conditions and bumps. The results of the four drivers raced individually on dirt tracks are shown in Figure 4. Results show that Autopia has bridged the longest raced distance for individual races for all dirt tracks while EVOR was catching up closely and the other competitors stayed reasonably behind. EVOR currently addresses the slipperiness of the tracks well with its traction control. We believe that by incorporating bump detection feature to EVOR, we can improve its performance on dirt tracks further.

5.4 Random Tracks

These tracks are generated from the TORCS compliant random track generator presented in Cardemone et al. [2]. These tracks consist of a combination of road and dirt conditions. Results show that EVOR has traveled the longest distance for the half of the random tracks (see Figure 4).

5.5 Race with Opponents

For all the tracks considered in the individual race round, we have then tested the racing performance with opponents also. For each track all the competitors were set to race together for 10 laps. The starting grid was based on the performance in the individual race. The results are shown in Table 1. The end results are quite similar to the previous experiment that generally the winners of the individual race are the winners of the race with opponents too. However, there are some interesting scenarios due to opponents observed during these experiments. Due to the strengths of opponent handing and stability EVOR could win three additional races in the final round than the previous individual race round. These were on tracks *forza* and *wheel-2* previously won by Mr.Racer and *torcello-mountain-snow* won by Autopia in the individual race round. These effects are depicted in the results shown in Figure 4 and Table 1. Although, EVOR has lost one track, namely *ole-road-1* that won in the previous individual race round. ICER with good bump detection capabilities won this race where as other drivers had problems with high damage rate in this high bumpy road thus could not complete the race. This was not observed in the individual race round as its only held for 10000 ticks within which most of the drivers could stay in the TORCS damage limit of 10000. Nevertheless, EVOR could win 19 out of 30 races in the opponents round that is even better than the 17 out of 30 in the individual race round. From the rest of 11 races EVOR could become the first runner up reporting a small time gap from the winner's time for most of the cases.

6. CONCLUSION AND FURTHER WORK

This study has remodeled the controlling of a vehicle as a dynamic optimization problem. The problem was then addressed by a simple evolutionary optimizer to optimize the actuator outputs in a dynamic environment. As a proof of the concept EVOR, a simulated driver implementation based on this approach is presented. Experiments were conducted to benchmark this driver on a diverse set of tracks and to compare the performance against existing best AI approaches. EVOR has outperformed current best AI controllers for the majority of the tested tracks. More broadly, this study reveals the potential of the evolutionary optimizer as a physical car controller specially in areas like cruise control where only fuzzy rule based controllers are currently being investigated. Future work will be concentrated on introducing bump detection and noise handing capabilities to EVOR to perform better on dirt tracks and in noisy environment.

7. ACKNOWLEDGEMENTS

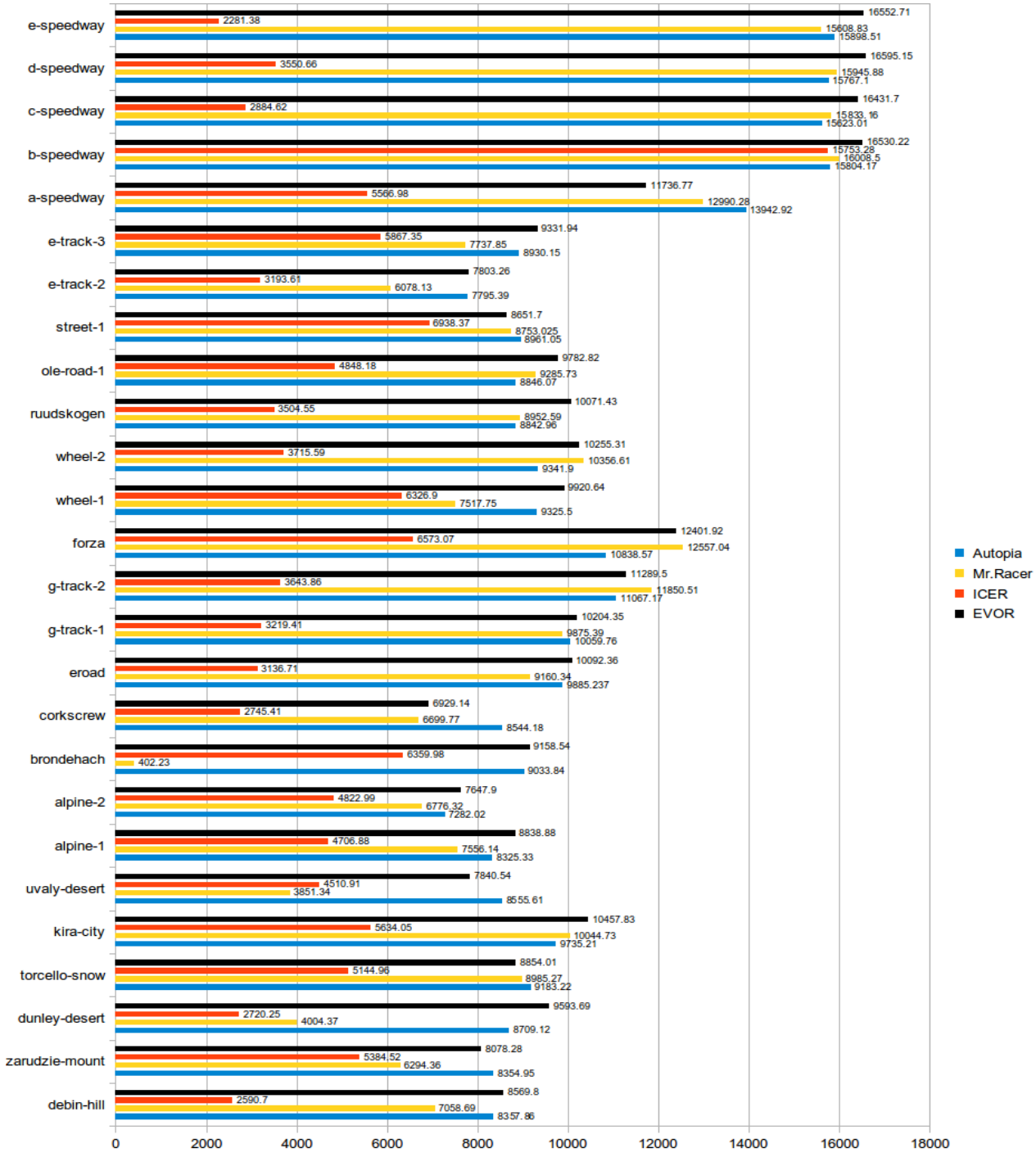

Figure 4: distance raced (in meters) for 10000 game ticks for oval, road, dirt and random tracks

We thank the organizers of SCR championship for the motivation and the feedback for the preliminary version of EVOR and also for sharing the sources of TORCS special version used for championship contest to conduct our experiments similar to SCR setting.

References

[1] L. Cardamone, P. L. Lanzi, D. Loiacono, and E. Onieva. Advanced overtaking behaviors for blocking opponents in racing games using a fuzzy architecture. *Expert Systems with Applications*, 40(16):6447 – 6458, 2013.

[2] L. Cardamone, D. Loiacono, and P. L. Lanzi. Interactive evolution for the procedural generation of tracks in a high-end racing game. In N. Krasnogor and P. L. Lanzi, editors, *GECCO*, pages 395–402. ACM, 2011.

[3] C. Edmondson. *Fast Car Physics*. The Johns Hopkins University Press, 1st edition, 2011.

road	winner		1st runner up		2nd runner up	
	name	time (m:s:ms)	name	winner+time	name	winner+time
a-speedway	Autopia	04:58:82	Mr.Racer	+09:24	**EVOR**	+26:79
b-speedway	**EVOR**	08:08:96	Mr.Racer	+21:85	Autopia	+22:65
c-speedway	**EVOR**	07:03:39	Autopia	+10:14	Mr.Racer	+19:16
d-speedway	**EVOR**	07:30:52	Autopia	+00:09	Mr.Racer	+00:17
e-speedway	**EVOR**	08:37:11	Mr.Racer	+17:80	Autopia	+1 lap
alpine-1	**EVOR**	23:35:53	Mr.Racer	+1 lap	ICER	+2 laps
alpine-2	**EVOR**	17:30:21	Autopia	+23:96	ICER	+ lap
brondehach	**EVOR**	14:12:96	Autopia	+08:19	Mr.Racer	+10:09
corkscrew	Autopia	14:39:02	**EVOR**	+1 lap	Mr.Racer	+2 laps
eroad	**EVOR**	11:10:32	Autopia	+07:69	Mr.Racer	+2 laps
g-track-1	**EVOR**	07:07:88	Autopia	+05:92	Mr.Racer	23:51
g-track-2	Mr.Racer	09:14:58	**EVOR**	+20:79	Autopia	+51:17
forza	**EVOR**	16:34:48	Mr.Racer	+01:12:59	Autopia	+10 laps
wheel-1	**EVOR**	15:12:70	Mr.Racer	+2 laps	ICER	+3 laps
wheel-2	**EVOR**	20:10:45	Mr.Racer	+04:16	ICER	+2 laps
ruudskogen	**EVOR**	11:25:40	Mr.Racer	+27:29	Autopia	+1 lap
ole-road-1	ICER	26:25:65	**EVOR**	+5 laps	Mr.Racer	+8 laps
street-1	Autopia	14:27:73	Mr.Racer	+53:78	**EVOR**	+01:08:67
e-track-2	**EVOR**	13:57:01	Autopia	+03:46	Mr.Racer	+2 laps
e-track-3	**EVOR**	15:48:82	Autopia	+16:57	Mr.Racer	+1 lap
dirt-1	Autopia	05:46:54	**EVOR**	+23:81	ICER	+4 laps
dirt-3	Autopia	10:53:26	**EVOR**	+08:74	ICER	+5 laps
dirt-4	Autopia	13:37:33	**EVOR**	+07:84	ICER	+1 lap
dirt-5	Autopia	05:35:30	**EVOR**	+46:95	ICER	+1 lap
debin-hill	**EVOR**	13:02:90	Autopia	+32:90	Mr.Racer	+1 lap
zarudzie-mountain	Autopia	14:32:58	**EVOR**	+17:06	ICER	+7 laps
dunley-desert	**EVOR**	11:53:22	Autopia	+16:63	ICER	+1 lap
torcello-mountain-snow	**EVOR**	16:10:45	Mr.Racer	+19:44	ICER	+1 lap
kira-yoshida-city	**EVOR**	13:12:28	Autopia	+01:00:92	Mr.Racer	+01:17:49
uvaly-desert	Autopia	13:36:75	**EVOR**	+55:71	Mr.Racer	+5 laps

Table 1: racing times of the winners of the final race with opponents on 30 tracks for 10 laps on each track (run under TORCS default settings, in visual mode and the starting grid is based on the results of the individual race round)

[4] E. Espie and C. Guionneau. Torcs the open racing car simulator. `http://torcs.sourceforge.net/`.

[5] D. Galanopoulos, C. Athanasiadis, and A. Tefas. Evolutionary optimization of a neural network controller for car racing simulation. In I. Maglogiannis, V. P. Plagianakos, and I. P. Vlahavas, editors, *SETN*, volume 7297 of *Lecture Notes in Computer Science*, pages 149–156. Springer, 2012.

[6] K.-J. Kim, J.-H. Seo, J.-G. Park, and J. C. Na. Generalization of torcs car racing controllers with artificial neural networks and linear regression analysis. *Neurocomput.*, 88:87–99, July 2012.

[7] T. S. Kim, J. C. Na, and K. J. Kim. Optimization of an autonomous car controller using a self-adaptive evolutionary strategy. *Int J Adv Robot Syst*, 73(9), 2012.

[8] D. Loiacono, L. Cardamone, and P. L. Lanzi. Simulated Car Racing Championship: Competition Software Manual, Apr. 2013.

[9] D. Loiacono and P. L. Lanzi. Simulated car racing championship. `http://scr.geccocompetitions.com/`.

[10] J. Muñoz, G. Gutierrez, and A. Sanchis. A human-like torcs controller for the simulated car racing championship. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 473–480, 2010.

[11] T. T. Nguyen, S. Yang, and J. Branke. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6:1–24, 2012.

[12] E. Onieva, J. Godoy, J. Villagra, V. Milanés, and J. Pérez. On-line learning of a fuzzy controller for a precise vehicle cruise control system. *Expert Syst. Appl.*, 40(4):1046–1053, 2013.

[13] E. Onieva, D. A. Pelta, J. Godoy, V. Milanés, and J. Pérez. An evolutionary tuned driving system for virtual car racing games: The autopia driver. *Int. J. Intell. Syst.*, 27(3):217–241, 2012.

[14] H. Pacejka. *Tire and Vehicle Dynamics*. Elsevier, 3rd edition, 2012.

[15] D. Perez, G. Recio, Y. Saez, and P. Isasi. Evolving a fuzzy controller for a car racing competition. In *Proceedings of the 5th International Conference on Computational Intelligence and Games*, CIG'09, pages 263–270, Piscataway, NJ, USA, 2009. IEEE Press.

[16] J. Quadflieg, M. Preuss, and G. Rudolph. Driving faster than a human player. In *Proceedings of the 2011 international conference on Applications of evolutionary computation - Volume Part I*, EvoApplications'11, pages 143–152, Berlin, Heidelberg, 2011. Springer-Verlag.