

# Learning Drivers for TORCS through Imitation Using Supervised Methods

Luigi Cardamone, Daniele Loiacono and Pier Luca Lanzi *Member IEEE*

**Abstract**—In this paper, we apply imitation learning to develop drivers for The Open Racing Car Simulator (TORCS). Our approach can be classified as a *direct method* in that it applies supervised learning to learn car racing behaviors from the data collected from other drivers. In the literature, this approach is known to have led to extremely poor performance with drivers capable of completing only very small parts of a track.

In this paper we show that, by using high-level information about the track ahead of the car and by predicting high-level actions, it is possible to develop drivers with performances that in some cases are only 15% lower than the performance of the fastest driver available in TORCS. Our experimental results suggest that our approach can be effective in developing drivers with good performance in non-trivial tracks using a very limited amount of data and computational resources. We analyze the driving behavior of the controllers developed using our approach and identify perceptual aliasing as one of the factors which can limit performance of our approach.

## I. INTRODUCTION

Imitation learning is currently regarded as a promising approach to build human-like artificial players [1], [2], [3], [4], [5], [6]. It includes all the methods that can be employed to learn a behavior from the observation of another player such as, (i) supervised learning, which has achieved good results in several games [7], [3], (ii) neuroevolution [5], [6], (iii) rule-based evolution [1], etc. The approaches used for imitation learning can be roughly divided into *direct methods* [4], which apply supervised learning to extract a model from the data logged from a target behavior; *indirect methods* [4], which apply typical methods of computational intelligence (e.g., neuroevolution) and use the data from the target behavior to compute a similarity score used to guide the development of the imitative controller.

In car racing games, direct methods are well-known to produce drivers with dramatically low performances that are usually incapable of completing even one lap (or even just a significant part of a lap) in very simple tracks (see the discussions in [4], [6]). In this work, we focus on The Open Racing Car Simulator (TORCS) [8] and apply supervised learning to develop controllers capable of driving in non-trivial tracks reaching a performance that is in some cases “only” 15% lower than that of the fastest bot available in TORCS. While in [4] the supervised models were trained

from the raw data collected from the target behavior, our approach, inspired by human players, exploits high-level information about the current status of the car and the road ahead. In addition, instead of predicting low-level control actions as done in [9], [10], [4], [5], [6], our approach predicts high-level actions such as a target speed and the car relative position with respect to the track central axis. These high-level actions, predicted by the supervised model, are then mapped to actions on the low-level effectors by a simple control program. This takes the car current speed/position and a target speed/position and produces the low-level actions that are needed to reach the target speed/position.

The results we present show that our approach can reach interesting performance creating drivers which follow as close as possible the trajectories drawn by the target bot, that is the fastest bot available in TORCS. We analyze the behavior of the drivers obtained using our methodology and suggest that the current performance (reported in this paper) can be further improved by (i) limiting the perceptual aliasing that often characterizes large portions of the track and by (ii) improving the reactivity of the basic controller that we use to map the target speed/position to the actual low-level actions on the game effectors.

## II. RELATED WORK

In the literature, there are several works describing applications of imitation learning to computer games. In this context, imitation learning is usually aimed at the development of non-player characters with more realistic and believable human-like behaviors. For instance, Priesterjahn et al. [1] developed a rule-based non player character (NPC) for *Quake III*<sup>®</sup> through a two-step process in which a set of rules was initially derived from the data collected from human players; subsequently, the ruleset was optimized using an evolutionary algorithm. Their final result was an NPC which would behave similarly to the observed human players while also generalizing well to unseen situations. Gorman et al. [2] combined reinforcement learning, fuzzy clustering, and a Bayesian motion-modeling system to develop an NPC for *Quake II*<sup>®</sup> which could imitate a human player. In particular, they decomposed the problem of learning through imitation in two subtasks: the modeling of high-level strategic patterns and the modeling low-level actions. Bryant & Mikkilainen [7] applied neural networks with backpropagation to *Legion II*, a discrete-state strategy game, while Chaperot and Fyfe [3] applied them to *Motocross The Force*. In the field of commercial games, the most prominent example of imitation learning is the X-Box game *Forza Motorsport*

Luigi Cardamone (cardamone@elet.polimi.it), Daniele Loiacono (loiacono@elet.polimi.it) and Pier Luca Lanzi (lanzi@elet.polimi.it) are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy.

Pier Luca Lanzi (lanzi@illgal.ge.uiuc.edu) is also member of the Illinois Genetic Algorithm Laboratory (IlligAL), University of Illinois at Urbana Champaign, Urbana, IL 61801, USA.

(Microsoft), where the player can train his own *drivatar*, a controller that learns the player’s driving style and can take his place in the races.

In the context of car racing games (which is the focus of this paper), the most relevant body of work was produced by Togelius and colleagues [4], [5], [6] who focused on the learning of a specific driving style using a simple 2D car simulator [4], [5] and The Open Racing Car Simulator [6]. In [4], Togelius, De Nardi, and Lucas applied supervised learning (more precisely, neural networks and k-nearest neighbor) to model driving styles directly, that is, by building models that could accurately predict the player actions based on the current game state. However, direct modeling resulted limited performance [4], accordingly, they moved to *indirect modeling* and used a genetic algorithm to evolve a controller with a driving style similar to a target human player. In [5], the approach based on indirect modeling was improved by using a different fitness function while later, in [6], it was extended by introducing a multi-objective evolutionary algorithm to evolve controllers which could be both robust (in that, they never run out of the track) and could demonstrate a driving style similar to one recorded from a user (in that, they accurately predict the user actions).

### III. TORCS

The Open Racing Car Simulator (TORCS) [8] is a state-of-the-art open source car racing simulator which provides a sophisticated physics engine, full 3D visualization, several tracks and models of cars, and different game modes (e.g., practice, quick race, championship, etc.). The car dynamics is accurately simulated and the physics engine takes into account many aspects of racing cars such as traction, aerodynamics, fuel consumption, etc.

Each car is controlled by an automated driver or *bot*. At each control step, a bot can access the current game state, which includes several information about the car and the track as well the information about the other cars on the track, and can control the car using the gas/brake pedals, the gear stick, and steering wheel. The game distribution includes several programmed bots which can be easily customized or extended to build new bots.

All the experiments reported in this paper have been carried out using the version 1.3.1 of TORCS.

### IV. IMITATION LEARNING IN TORCS USING SUPERVISED METHODS

In this work, we applied supervised learning to develop car controllers for The Open Car Race Simulator from the logs collected from other drivers. Our approach is inspired by the early work of Togelius et al. [4], where it was noted that applying a direct method would lead to unsatisfactory results. In this work we extend the results of Togelius [4] by showing that with a higher-level representation of sensory information and actions it is possible to achieve interesting performance.

We considered two representations of the current state of the car: (i) the set of rangefinder inputs usually employed in simulated car racing competitions [11], [12] and in neuroevolution experiments [9], [10], [6]; and (ii) a high-level, qualitative, representation involving basic lookahead information about the track in front of the car. Instead of predicting the typical low-level actions on the car actuators available in TORCS (namely, the steering wheel, the gas pedal, the brake pedal and the gear change), our approach predicts a target speed and the car position with respect to the track axis.

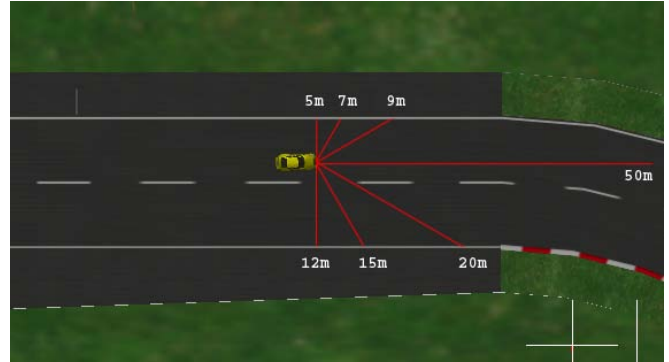


Fig. 1. Rangefinder Sensor.

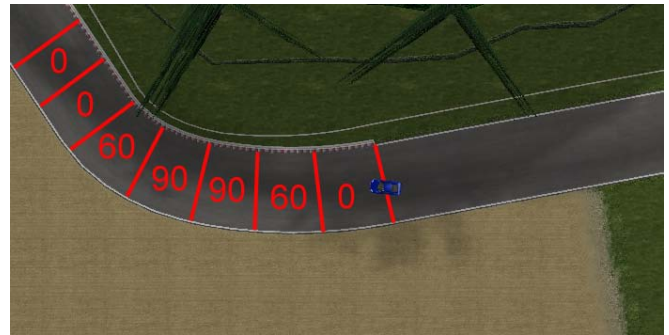


Fig. 2. Lookahead Sensor.

#### A. What Input Sensors?

TORCS provides access to much information about the state of the car, of the opponents, and of the track surroundings. In this work, we considered two types of sensory representations for the driver bot. The first sensory representation is based on a rangefinder and has been widely used in the literature [4], [6], [13], for the simulated car racing competitions (see the competition pages of WCCI-2008, CIG-2008, CEC-2009 and GECCO-2009), and in our previous works on the evolution of driving and overtaking behaviors [9], [10]. The rangefinder casts an array of beams, with a range of 100m, around the car and returns the distance from the car to the track edge (Figure 1).

The second and novel sensory representation is inspired by the behavior of human drivers whose decisions are usually based on (i) high-level information about the current speed

and trajectory (instead of the car distance from the track edges) and on (ii) the shape for the track ahead (e.g., Is there a sharp turn ahead? Then I should slow down and move to the external part of the track). Accordingly, we developed a new sensor model, dubbed *lookahead sensor*, that represents the car state using *just the bending radius of the track segments ahead*. For this purpose, the next  $h$  meters of the track ahead are considered and divided into segments of the same length (10 meters, in the experiments considered here) and the lookahead sensor returns the bending radius for each segment: a rectilinear segment corresponds to a 0; a positive radius corresponds to a right turn; while a negative radius corresponds to a left turn. Figure 2 shows an example of lookahead sensor: the car is approaching a short right turn thus the sensor will return the vector  $[0, 60, 90, 90, 60, 0, 0, 0]$  representing the next 80 meters ahead (i.e., the 8 segments ahead).

### B. What Actions?

In TORCS, a car is controlled by four effectors: the steering wheel, the gas pedal, the brake pedal and the gear change (see Table I for details). Previous works [4], [5] and our preliminary experiments (not reported here) suggest that it is very difficult to apply supervised learning to compute a driving policy using such a low-level effectors. Accordingly, in this work, we propose to control the car at a higher level: we try to learn the trajectories and the speeds along the track rather than low level commands. A trajectory can be represented as the distance from the track axis in each segment of the track. So the output variables of the controller are the speed and the axis distance in the current segment of the track. During the simulation, a simple programmed policy acts to reduce the error among the actual variable and the predicted one. Algorithm 1 describes the simple policy that controls the steering command to reach the target distance from the track axis. First, the wheel is aligned to the car axis and then a correction is applied to reduce the error. The constant  $\epsilon$  is used to tune the steering policy: the higher is  $\epsilon$  the more reactive will be the response of the controller. Algorithms 2 and 3 describe the policy used to reach a target speed: simply accelerate at full speed when the current speed is low and brake when is too high. We apply a soft transition between brake and accelerator when the actual speed is near to the target speed: in this way we avoid switching from full acceleration to full brake with fluctuations around the target speed. To slightly simplify the problem, the driving behavior does not learn to shift gears, but instead a simple scripted policy based on engine RPM is used.

### C. What Learning Method?

We considered two supervised learning methods, multi-layer neural networks and k-nearest neighbor classifiers, and applied them to compute a model mapping input sensors to actions which could imitate the behavior of an observed driver. The model was then deployed to a driver controller that would map the high-level actions predicted by the

---

#### Algorithm 1 Calculate the Steer value.

---

```

1: procedure GETSTEER(targetpos, actualpos)
2:   angle = getCarAngle();      ▷ computing the car
   angle wrt the track axis
3:   error = targetpos - actualpos;
4:   angle = angle + error ·  $\epsilon$ ;
5:   steer = angle/steerLock;
6:   return steer;
7: end procedure

```

---



---

#### Algorithm 2 Calculate the Acceleration value.

---

```

1: procedure GETACCEL(targetspeed, actualspeed)
2:   accel = 0;
3:   if (targetspeed - speed > 0) then
4:     accel = (targetspeed - speed)/20;
5:   end if
6:   if (targetspeed - speed > 20) then
7:     accel = 1;
8:   end if
9:   return accel;
10: end procedure

```

---

models to low-level actions of the usual TORCS effectors using the algorithms discussed in the previous section.

**k-nearest neighbour** does not involve any training in that the *collected data represent the model*. Accordingly, it has been directly applied during the evaluation, i.e. during a race using TORCS. At each game tic, the logged data are searched to find the  $k$  most similar instances to the current sensory input using either the typical rangefinder representation or the new lookahead representation. The  $k$  similar instances are selected and the respective outputs variable are averaged to predict the target value. The similarity measure used is simply the Manhattan distance among instances.

**Neural Networks.** To avoid the issue of selecting a particular neural network structure, we applied Neuroevolution with Augmenting Topology (briefly, NEAT) [14] which applies a genetic algorithm to evolve both the weights and the topology of a neural network. In particular, we applied NEAT [14] to evolve *two separate neural networks*, one to predict the target speed and one to predict the target position for a given input configuration. The fitness was simply defined as the prediction error computed as the total sum of the absolute error between the true value and the predicted value, for each instance in the training data. The training using a rangefinder sensor produced two neural networks with 20 inputs (19 angle values plus a bias input) and one output, while in the case of the lookahead sensors the networks have only 9 inputs (8 segments plus a bias input). Note that, while in [14], [9], [10], [6] NEAT is applied to learn a controller, in this case, NEAT is used to *learn a predictive model* from the logs collected during several runs of TORCS.

---

**Algorithm 3** Calculate the Brake value.

---

```
1: procedure GETBRAKE(targetspeed,actualspeed)
2:   brake = 0;
3:   if (targetspeed - speed < 0) then
4:     brake = -(targetspeed - speed)/20;
5:   end if
6:   if (targetspeed - speed < -20) then
7:     brake = 1;
8:   end if
9:   return brake;
10: end procedure
```

---

TABLE I  
DESCRIPTION OF AVAILABLE EFFECTORS.

Name	Description
steering	Steering value: -1 and +1 means respectively full left and right, that corresponds to an angle of 0.785398 rad.
gas	Virtual gas pedal (0 is no gas, 1 is full gas).
brake	Virtual brake pedal (0 is no brake, 1 is full brake).
gear	Gear value defined in $\{-1,0,1,2,3,4,5,6\}$ where -1 is reverse and 0 is neutral.

#### D. What Data?

To test our approach, we collected the source data by running the *Inferno* bot (the fastest bot available in the TORCS package) on three tracks: (i) *G-track-1* (Figure 3(a)), a simple track which presents interesting trajectories when driven fast; (ii) *Wheel-1* (Figure 3(b)), a more difficult track with many fast turns; and (iii) *Aalborg* (Figure 3(c)), a difficult track with many slow sharp turns. For each track, we run the *Inferno* bot for three laps and recorded the data of the second lap so as to remove inaccurate information about speed and trajectories from the first and the final laps. We logged the data from the rangefinders [11], [12], [9], [10], [6] and from our new lookahead sensors. We also logged the car distance from the track axis and the car speed which we use as outputs. Overall, the process resulted in three datasets, one for each track, respectively containing 1982 examples for the *G-track-1*, 3899 examples for *Wheel-1* and 3619 examples for *Aalborg*. Note that, since the sensory inputs and the target outputs were logged every game tic, the three data sets contain several duplicates since, in some parts of the tracks, the car state and track state do not change between tics (for instance, on long rectilinear sections).

#### V. EXPERIMENTAL RESULTS

In our experimental analysis we applied two supervised algorithms, the basic k-nearest neighbor and neural networks evolved using NEAT, to the datasets containing the information from the rangefinder inputs and from the lookahead inputs. For each combination of sensor model and algorithm, we trained three models to drive on each one of the three tracks used to collect the training data (Figure 3) and we also trained a fourth model by applying the learning algorithm to

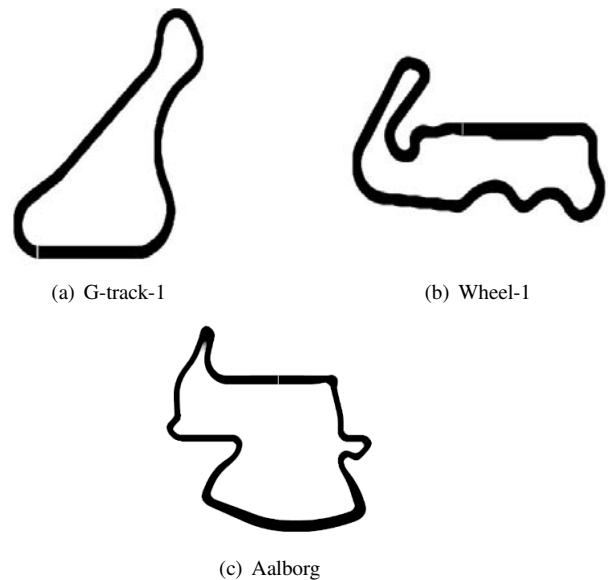


Fig. 3. Tracks used to log the data for training.

a set containing all the data from the three training tracks. Overall, we obtained 16 models (two types of sensors  $\times$  two types of algorithms  $\times$  four possible datasets). *k*-nearest neighbor was applied using a value of *k* empirically set to 20; such a relatively high value of *k* was needed to deal with the many duplicate or very similar instances potentially present in the datasets. NEAT was applied with a population of 100 individuals for 100 generations (all the other parameters were set to their default value). All the experiments were conducted using the version 1.3.1 of TORCS and the car model *car1-trb* [8].

#### A. Evaluation

We evaluated the 16 models by applying them to predict the target speed and position necessary for the simple programmed driver described in Section IV. Each model was evaluated by using it to drive a car on each track and the performance was measured as the distance raced by 10000 game tics (around 3 minutes and 20 seconds of actual race time). The driver was also equipped with the standard recovery policy used in other bots distributed with TORCS to bring the car back to the center of the track when the car crashes. The recovery policy was introduced to provide a better evaluation of the models while not penalizing models which would crash only in very challenging situations but drove well on the other parts of the track. Note that, since the recovery process is very slow, a model that causes the car to crash is still highly penalized so that the performance still distinguishes between a bot that performs well on the whole track and a bot that crashes at some point. The evaluation procedure involved the three tracks used for training (Figure 3) and, to test the generalization capabilities of the models, also two unseen tracks: *G-track-2* (Figure 4(a)), a simple fast track and *E-track-3* (Figure 4(b)) which presents many fast and difficult turns.

Test Track	Inferno	Trained on G-track-1		Trained on Wheel-1		Trained on Aalborg		Trained on 3 Tracks	
		rangefinder	lookahead	rangefinder	lookahead	rangefinder	lookahead	rangefinder	lookahead
G-track-1	10081.37	4673.84	6197.03	4091.91	4480.35	6031.47	<b>6577.08</b>	7757.81	6314.51
Wheel-1	10596.45	3075.00	4951.25	4009.53	7604.33	5815.82	6333.62	7596.44	<b>8247.82</b>
Aalborg	6786.62	3959.58	3571.14	3316.06	3525.84	3220.95	<b>5529.19</b>	3228.57	4334.33
G-track-2	11720.72	5671.18	5600.53	2813.69	<b>9560.65</b>	6300.87	6493.02	8546.6	8064.19
E-track-3	10888.03	3639.98	5864.40	4076.91	<b>7727.66</b>	5584.86	6564.73	7112.01	7075.06

TABLE II

PERFORMANCE OF K-NEAREST-NEIGHBOR BASED ON FOUR DATA SETS WHEN APPLIED TO G-TRACK-1, WHEEL-1, AALBORG, G-TRACK-2, AND E-TRACK-3 USING RANGEFINDER AND LOOKAHEAD INFORMATION.

Test Track	Inferno	Trained on G-track-1		Trained on Wheel-1		Trained on Aalborg		Trained on 3 Tracks	
		rangefinder	lookahead	rangefinder	lookahead	rangefinder	lookahead	rangefinder	lookahead
G-track-1	10081.37	3365.72	<b>8690.18</b>	5243.68	5915.58	2988.09	5990.67	3242.74	8474.46
Wheel-1	10596.45	1194.73	5595.04	3357.93	<b>6764.30</b>	2671.98	5473.36	3758.3	5890.45
Aalborg	6786.62	798.10	3964.74	1877.41	3379.05	1882.22	<b>3992.26</b>	2866.54	3983.82
G-track-2	11720.72	1910.37	<b>9158.23</b>	6666.29	5554.72	4118.37	6039.25	3953.09	9010.53
E-track-3	10888.03	1360.21	5625.56	3024.45	<b>6920.70</b>	2126.86	5020.50	3249.57	4806.56

TABLE III

PERFORMANCE OF NEAT BASED ON FOUR DATA SETS WHEN APPLIED TO G-TRACK-1, WHEEL-1, AALBORG, G-TRACK-2, AND E-TRACK-3 USING RANGEFINDER AND LOOKAHEAD INFORMATION.

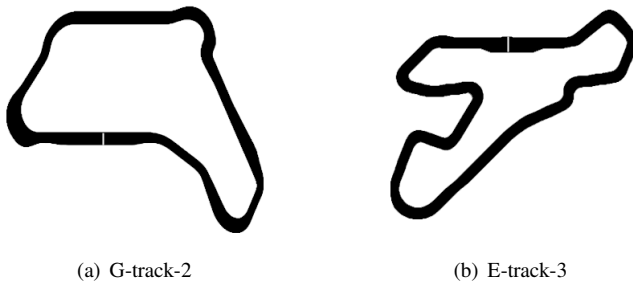


Fig. 4. Additional tracks used for testing.

## B. Results

Table II reports the performance of the bot Inferno and, for each data set, the performance of the k-nearest neighbor using the rangefinder and lookahead information for all the four training data sets and the five test tracks. As can be noted, the models based on rangefinder sensors have a lower performance than the ones using lookahead sensors. In fact, the drivers trained with rangefinders usually cover (with very few exceptions) less distance than those trained using lookahead inputs. This is coherent with the previous findings of Togelius et. al [4], [5], [6] where it is noted that applying supervised learning directly to the raw car sensor data does not result in interesting performance and typically produces controllers unable to finish even one lap (J. Togelius, personal communication).

It is interesting to note however that, since we predict high-level outputs (such as the target speed and the next position on the track) instead of basic control actions (as done in [4], [6], [9], [10]), our approach can achieve interesting performance even with raw sensory data. In fact, the results

in Table II show that the drivers using rangefinders cover around the half of the distance covered by the fastest driver available in TORCS which in terms of laps means that the drivers computed using our approach always cover at least one lap — a result which was basically impossible to obtain by predicting low-level actions (results not reported here). Furthermore, when the data about all the tracks are used for training, k-nearest neighbor reaches a performance that is “only” between 25%-30% lower than that of the Inferno bot (last two columns in Table II), the fastest bot available in the TORCS distribution. For instance, on the G-track-1, k-nearest neighbor achieves a performance of 7757.81 whereas the Inferno bot has a performance of 10081.37. The same holds for the tracks Wheel-1 and G-track-2.

Table III reports the performance of the models evolved by NEAT using rangefinders and lookahead inputs. These results confirm the findings of the previous set of experiments using k-nearest neighbor in that the performance obtained using lookahead inputs is higher than the one obtained using rangefinders (the only exception being when the model is trained using the data for Wheel-1 is applied to G-Track-2). On the G-Track-1, the performance of the models evolved using NEAT is generally higher than the performance of k-nearest neighbor (with the exception of the E-track3). On the other tracks the performance of k-nearest neighbor is usually higher than the performance of the models evolved by NEAT (with some exceptions). Interestingly, there are cases in which the performance of the drivers using the models evolved by NEAT is comparable with that of the Inferno bot. For instance, the model evolved using lookahead inputs on the G-track-1 has a performance that is just the 15% lower than Inferno which to us represents an incredible results since the model was obtained from the data of just one lap.

To summarize all the results, we computed the average performance for every track and for every possible combination of technique and input representation used and reported the result in Figure 5. As previously discussed, the models using lookahead sensors perform much better than those using the typical rangefinder inputs. Although, the coupling of lookahead sensors with NEAT achieves the most interesting results (leading to a rather low 15% difference between the performance of our model and that of the fastest bot available in TORCS), it is also subject to a high variability and in fact, its average performance is lower than that of k-nearest neighbor using lookahead in three tracks out of five while it is higher only on the G-Track-1.

### C. Execution Time

One very important aspect about the proposed approach regards its simplicity and the consequent very low computational cost. Indirect methods (e.g., [4], [5]), as well as other online learning approaches (e.g., [9], [10]), require several simulations of actual races. In contrast, direct methods, like the one proposed in this work, only need few runs to collect the data and the time for model building which is typically very limited when compared to the time required to perform several simulations.

Table IV compares the CPU time required to train the supervised models using NEAT and the CPU time required to evolve a simple controller using NEAT as done for instance in [10]. As can be noted, our approach needs 30 times less CPU time to obtain a reasonable (although not as performing) driver and this difference becomes more evident in more difficult tracks (namely Wheel-1 and Aalborg) where we stopped the evolutionary learning after 1:30 hours of CPU time.

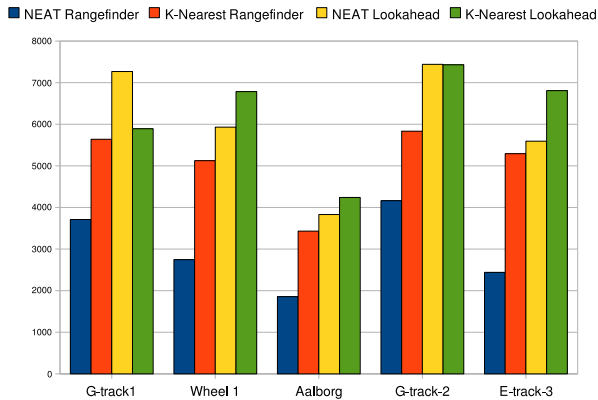


Fig. 5. Comparison among the four combinations of controllers.

### D. Increasing The Lookahead

The previous results suggest that the combination of lookahead sensors and high-level actions might be an interesting approach to represent input data for applying supervised methods to develop drivers through imitation learning. But how much lookahead is useful?

We performed a second set of experiments to compare the performance of k-nearest neighbor when 8 or 16 segments are used as lookahead. Table V, Table VI and Table VII compare the performance of k-nearest neighbor on the three training tracks, i.e., G-track-1, Wheel-1, and Aalborg using the same lookahead sensor used in the previous section (i.e., using 8 segments ahead) and a lookahead sensor using 16 segments ahead. The results shown are somehow predictable in that greater lookahead causes overfitting. In fact, the performance is higher (with a very small improvement on *Aalborg*) when the controller is tested on the same track used to build the model (reported in bold in Table V, Table VI and Table VII) while it is lower when the same model is applied to the other two tracks. This overfitting can be explicitly by looking at Figure 6 where the trajectories of (i) the bot *Inferno*, (ii) the controller using 8 lookahead segments, and (iii) the driver using 16 lookahead segments are compared on the G-track1. As can be noted in Figure 6, the trajectory produced by the driver using a 16 segments lookahead (black line) is closer to the trajectories of the bot *Inferno* than the driver with an 8 segment lookahead (red line).

TABLE V

LOOKAHEAD COMPARISON USING TRAINING TRACK *G-track-1*

	8 segments	16 segments
<b>G-track-1</b>	<b>6032.08</b>	<b>8389.03</b>
Wheel-1	6596.16	5038.08
Aalborg	3965.68	3940.8

TABLE VI

LOOKAHEAD COMPARISON USING TRAINING TRACK *Wheel-1*

	8 segments	16 segments
G-track-1	5141.54	4627.59
<b>Wheel-1</b>	<b>7627.12</b>	<b>8485.97</b>
Aalborg	3437.68	3095.51

TABLE VII

LOOKAHEAD COMPARISON USING TRAINING TRACK *Aalborg*

	8 segments	16 segments
G-track-1	6583.46	4613.46
Wheel-1	6374.1	4586.64
<b>Aalborg</b>	<b>5948.11</b>	<b>5949.35</b>

## VI. DISCUSSION

We performed a qualitative evaluation of the driving behaviors obtained using our methodology. For this purpose, we selected the drivers whose performance (i.e., whose distance raced in 10000 game tic) differs for less than 2000 meters from the performance of *Inferno* and labeled them as *good drivers*. We determined the threshold of 2000 meters empirically, by watching the controllers during an entire lap and comparing their driving behavior with the target bot *Inferno*. The *good* controllers, with a performance less than 2000 meters away from *Inferno*, appear qualitative good on

TABLE IV  
CPU TIME FOR (I) TRAINING THE SUPERVISED MODELS USING NEAT (ACCELERATION AND STEERING) AND (II) FOR EVOLVING A CONTROLLER USING NEAT [10].

Track	Train Acceleration	Train Steering	Evolutionary Learning
G-track-1	1m24s	1m25s	47m
Wheel-1	2m36s	3m03s	>1h30m
Aalborg	2m28s	2m28s	>1h30m

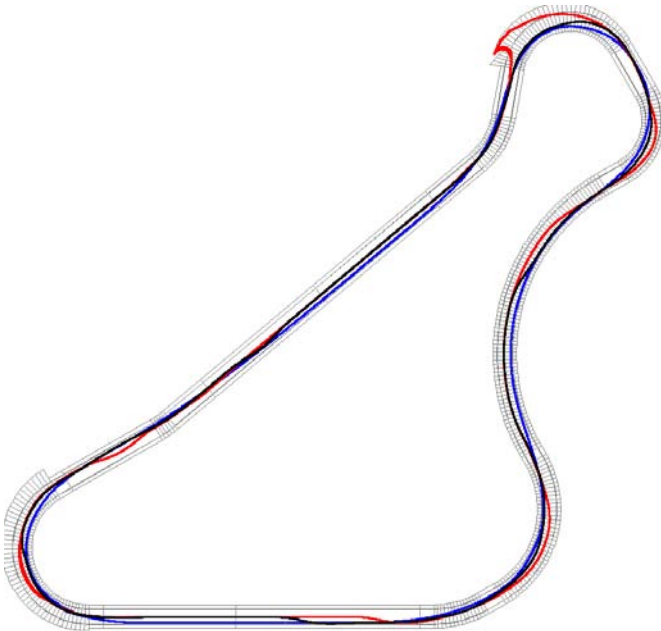


Fig. 6. Trajectories comparison in track *G-track-1*: bot *Inferno* (blue line), 8 segments lookahead (red line), 16 segments lookahead (black line).

screen: they are not as fast as the programmed bot, but their trajectories are as close as possible to the trajectories taken by the target bot (as can be noted in Figure 6 by comparing the blue line of *Inferno* with the red line produced by our bot). The only limitation we observed in *good* controllers is that they are likely to run out of track in very difficult turns (a typical place where this can happen is the last turns of track *Wheel-1*). An additional analysis of the drivers' behavior we performed suggests that this happens because the bot enters the turn with a speed near to the maximum possible and a small error in the prediction or a low reactivity to apply the steer command (Algorithm 1) can result in a crash. Accordingly, planned future works include the improvement of the controller responsible to map the high-level actions to the actual low-level effectors (Algorithm 1) to increase the driver's reactivity and reliability.

Controllers that we do not consider good (i.e., whose performance measure is much lower with respect to the bot *Inferno*) presents very poor behaviors. The main problem of these controllers is the presence of many discontinuities in the prediction of the trajectories which for instance can cause the car to move quickly from one side of the track

to the other one. This behavior can be experienced several times along the track and when it happens during a turn, it can cause a crash. For instance, a discontinuity is visible in Figure 6 in the first straight part of the track. Our analysis suggests that such discontinuities usually happen when the car drives through track configurations never seen during the training process.

#### A. Perceptual Aliasing

Finally, a general problem that can affect all the drivers is the perceptual aliasing which the driver can experience when in the training data there are instances that present similar track configurations but different values of speed and position. Perceptual aliasing is typically present in the straight parts of a track, where many segments present the same sensory information but correspond to different values of speed and position. An example of perceptual aliasing can be noted in the trajectories in Figure 6: in the lower horizontal straight part of the track, the *Inferno* bot stays near the right side of the track (the driver moves counter clockwise), whereas in the diagonal straight part of the track (on the left side of Figure 6) the *Inferno* bot tends to stay on the left side of the track. These two parts of the track cause perceptual aliasing to the controller in that they require different trajectories and speeds but their representation in terms of drivers' sensory inputs are basically identical. Accordingly, it is impossible for a driver (derived from the log data) to learn separate policies for the two track parts. In fact, the trajectories for the learned bot in Figure 6 are similar for most of the straight parts of the track. The behavior changes only near the turns when the perceptual aliasing is finally solved by the sensory information arriving from the forthcoming turn.

When perceptual aliasing is present, the driver cannot learn to reach the top speed specific to each part of the track, but it usually learns a target speed that is the average of the speeds in each one of the aliased segments. This is also one of the causes of the lower performance of the learned bots with respect to the target bot *Inferno*. A simple way to remove the perceptual aliasing in a straight part, would consist of treating in a special way the input configuration that identifies a long straight so as to force the car to accelerate at full speed. Aliasing can also be reduced increasing the lookahead distance. In the experiments with an increased lookahead the controllers correctly accelerate for a large part of the straight. It is interesting to underline that, in the track *Aalborg*

the performance improvement between the simple and the extended lookahead is very small, as reported in Table VII. This happens because, in track *Aalborg*, each turn is very different from the others, and there is almost no perceptual aliasing, so that the extended lookahead controller achieves only a little improvement.

## VII. CONCLUSIONS AND FUTURE WORKS

In this work, we applied supervised learning to develop driving behaviors for TORCS through imitation. To develop drivers that are robust to prediction error, we proposed an approach that focuses on the control of high-level aspects of the driving, namely speed and trajectory, rather than the low-level effectors typically used in previous experiments reported in the literature [9], [10], [6]. To model complex driving human-like behaviors accurately, we introduced a novel way to represent the driver sensory inputs using high-level, qualitative, information about the track ahead using basic lookahead information. We performed an empirical analysis and compared four combinations of two sensory input representations and two supervised learning techniques (namely, k-nearest neighbor and neural networks) on five tracks. The reported results suggest that even a very simple technique such as k-nearest neighbor, when coupled with our new lookahead high-level representation, can produce a reasonably good driver for a specific track. The driving performance however decreases when we try to generalize and use the same driver on a different track.

None of the controllers developed using our methodology performs better than or similarly to the bot *Inferno* which was used to log the data. Our analysis suggest that perceptual aliasing is one of the critical factors which may cause such a lower performance of our learned bots. Another cause of reduced performance is the controller used to map the high-level actions to the low-level effectors. In fact, we noted that in abrupt turns, the controller (Algorithm 1) reacts too slowly and can cause the car to run off track.

Although this work is preliminary, the presented results are promising in that they are actually better than other ones reported in the literature for direct methods (see the discussion about direct methods in [5], [6]). Therefore, they represent a good starting point for further improvements. Future works will aim at improving generalization for instance by exploiting structural symmetry on the track, or by applying the knowledge acquired on the right turns also to control the car during the left turns presenting similar shapes, and vice versa. Generalization could also be improved by introducing better supervised methods and training data covering several representative tracks. In addition, to increase the robustness to noise and to reduce the computational costs, we plan to introduce data reduction techniques to clean the data sets from useless duplicates and to eliminate less informative attributes. Finally, to improve the driving performance, the programmed policy that calculates the steer command from the target position, could also be improved so as to react quicker to abrupt direction changes, i.e. sharp turns, while remaining robust to noisy or discontinuous predictions.

## REFERENCES

- [1] S. Priesterjahn, O. Kramer, A. Weimer, and A. Goebels, "Evolution of reactive rules in multi player computer games based on imitation," in *ICNC (2)*, 2005, pp. 744–755.
- [2] B. Gorman, C. Thurau, C. Bauckhage, and M. Humphrys, "Bayesian imitation of human behavior in interactive computer games," in *Proceedings of the International Conference on Pattern Recognition (ICPR '06)*, vol. 1, IEEE, 2006, inproceedings, pp. 1244–1247.
- [3] B. Chaperot and C. Fyfe, "Improving artificial intelligence in a motocross game," in *Proc. IEEE Symposium on Computational Intelligence and Games*, 2006, pp. 181–186.
- [4] J. Togelius, R. De Nardi, and S. M. Lucas, "Making racing fun through player modeling and track evolution," in *Proceedings of the SAB'06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, 2006.
- [5] J. Togelius, R. De Nardi, and S. Lucas, "Towards automatic personalised content creation for racing games," in *Proc. IEEE Symposium on Computational Intelligence and Games CIG 2007*, 2007, pp. 252–259.
- [6] N. van Hoorn, J. Togelius, D. Wierstra, and J. Schmidhuber, "Robust player imitation with multiobjective evolution," in *Proceedings of IEEE Congress on Evolutionary Computation*, 2009.
- [7] B. Bryant and R. Miikkulainen, "Exploiting sensor symmetries in example-based training for intelligent agents," in *Proc. IEEE Symposium on Computational Intelligence and Games*, 2006, pp. 90–97.
- [8] "The open racing car simulator website." [Online]. Available: <http://torcs.sourceforge.net/>
- [9] L. Cardamone, D. Loiacono, and P. Lanzi, "On-line neuroevolution applied to the open racing car simulator," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, May 2009, pp. 2622–2629.
- [10] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Evolving competitive car controllers for racing games with neuroevolution," in *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2009, pp. 1179–1186.
- [11] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated car racing championship 2009: Competition software manual," Dipartimento di Elettronica e Informazione, Politecnico di Milano, Tech. Rep., 2009.
- [12] D. Loiacono, J. Togelius, and P. L. Lanzi, "Software manual of the car racing competition." [Online]. Available: <http://mesh.dl.sourceforge.net/sourceforge/cig/CIG2008-Manual-V1.pdf>
- [13] N. Kohl, K. Stanley, R. Miikkulainen, M. Samples, and R. Sherony, "Evolving a real-world vehicle warning system," in *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2006, pp. 1681–1688.
- [14] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.