Learning the Track and Planning Ahead in a Car Racing Controller

Jan Quadflieg, Mike Preuss, Oliver Kramer, Günter Rudolph

***Abstract*— We propose a robust approach for learning car racing track models from sensory data for the car racing simulator TORCS. Our track recognition system is based on the combination of an advanced preprocessing step of the sensory data and a simple classifier that delivers six types of track shapes similar to the ones a human would recognize. Out of these, establishing a complete track model is straightforward. This model provides an information advantage to controller strategies, as it generally enables *planning*. We demonstrate how such a planning controller can be derived by a mixture of expert knowledge and a simple evolutionary learning approach and give experimental evidence that knowing not only the current conditions but also the big picture of the track is beneficial, as may be expected.**

I. Introduction

Car racing is a popular genre in computer games and an interesting test domain for methodologies from artificial and computational intelligence. In the past couple of years, *The Open Racing Car Simulator* (TORCS) has been used as testbed for artificial intelligent algorithms, and we concentrate exclusively on this game-like testbed in the following. Learning a racing controller is a challenging task and has obtained much attention in the last years. The difficulties comprise handling the special properties of car and track, interaction with other cars, e.g. overtaking, as well as adaptation to policies of the opponents. However, even driving a car on a track on its own is learned easily by a human but not that easy for a computer program if one wants to race at high speed. Humans employ lots of implicit heuristic rules that are fed by experience which the artificially intelligent controllers do not possess. To strengthen this perspective, we will provide a direct comparison showing that even the best currently available controllers are beaten without much effort by a moderately trained human (see Figure 8). It may thus make sense to attempt providing the same level of track knowlege to controllers than humans attain by just looking through the windshield. Of course, this also increases the flexibility of a controller so that it can quickly adapt to virtually any track, known or unseen before.

Current car racing controllers often employ one of two approaches (e.g. [1]):

- Either they use a black box approach by providing all available data to a learning algorithm and let it adapt the controller without any insight concerning *how* the controller works.
- Or, they preprocess the sensor data to discretize them into categories that would also be used by human (e.g. rally) drivers when describing the track, and then use a rule-based system to react to the current conditions.

Taking the second approach surely needs more conceptual work at first, but has several advantages. Data summarization to a human understandable level and controlling itself are separate tasks and may also be interchanged between controllers. Besides this implementation advantage, it is also easier to detect where something is wrong if any problem occurs. This would be almost impossible e.g. for an *artificial neural network* (ANN) controller. We conjecture that partition into 2 steps greatly enhances the possibilities to make the controllers drive human-like (or at least more believable) as humans seem to follow a similar scheme when controlling a car.

However, the first step of distilling useful categorial information concerning the current track state may not be as easy as expected, at least with the sensor data available in TORCS. Simple classification schemes may confuse the entrance of a sharp turn and a corner that can be driven with full speed as they look alike from the point of view of the 19-dimensional sensor model (see figure 1). According to the published video takes of the CEC'09 TORCS competition, our team (Mr. Racer) was the only one to drive through a difficult combination of curves (the corkscrew) in a satisfactory manner, while the other cars usually left the track and often crashed into the wall.

In the following, we first detail the track segment recogition system which enables 'looking into the next curve' and then describe our standard controller that is able to drive appropriately, based on this information. Section IV describes how to construct a full track model out of segment data and Section. V details a *planning* controller employing

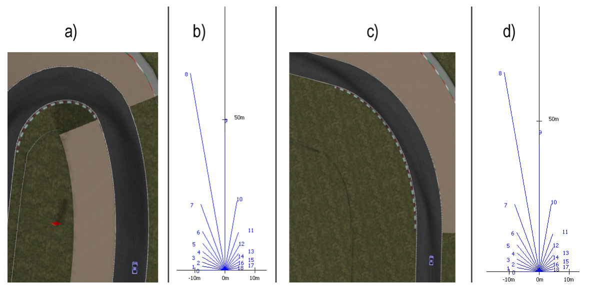


Fig. 1: Misleading sensor information. a) Approaching a tight hairpin, b) sensor data belonging to a), c) approaching a high speed corner, d) sensor data belonging to c) (which is nearly identical in both cases)

Jan Quadflieg, Mike Preuss and Günter Rudolph are with the Chair of Algorithm Engineering, Computational Intelligence Group, Dept. of Computer Science, Technische Universität Dortmund, Germany. E-mail: firstname.lastname@tu-dortmund.de

Oliver Kramer is with the Algorithms Group at the International Computer Science Institute, University of California, Berkeley, CA, USA. E-mail: okramer@icsi.berkeley.edu

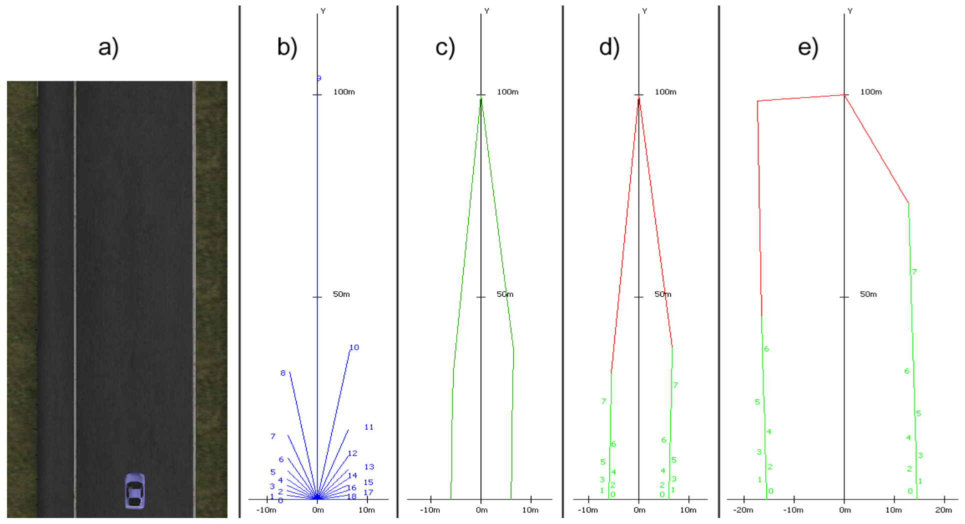


Fig. 2: Visualization of sensor data: a) Screenhot of the game, b) reconstructed vectors belonging to each sensor, c) vectors $\vec{t_i}$, the reconstructed outline of the track, d) shown in red the vectors not used to calculate the measure ($idx_l = idx_r$), e) a wider track, where $idx_l \neq idx_r$.

knowledge of the complete track for deciding e.g. where to position itself on the track at the beginning of a curve. We close with summary and conclusions.

II. Track Segment Recognition System

In order to build an accurate and human-readable track model that makes planning possible, we first need to identify the different track segments, namely different curve types. This must be done in an indirect manner from the incoming sensor data which gives only a rough radar-like scan of the environment and needs to be suitably interpreted. Our approach consists of two steps. At first, we estimate the 'real' angle of the next curve, then we classify the curves into different types according to angle information. Our controller then builds on this type information to adjust car behavior accordingly, especially the targeted speed.

A. *TORCS Sensory Model And Default Segment Recognition*

For assessing its own position on track and perceiving its environment, a TORCS controller for the 2009 car racing championship had access to 72 sensor values of which we further on focus on the 19 track sensors (see Figure 2 b)). These enable recognizing the track shape, and especially sensing curves. In steps of 10 degrees we obtain the 'free' length of the track up to 100 m from -90 to 90 degrees relative to the current heading of the car. The interface also provides some more values as the position on the track and the distance to the finish line which are of minor interest here, see [2] for an overview.

Most controllers we are aware of use a simple heuristic for estimating the curvature of the next track segment as they rely on the longest of the 19 obtained track sensor vectors (e.g. [3]). While this is a useful first guess, it leads to problems for extremely tight curves, the so-called *hairpins*, see Figure 1 a). Driving too fast into a hairpin often results in leaving the track as not enough space remains for braking down to the maximum allowed speed. It is therefore important to recognize this kind of curve early. With a re-mastered heuristic, we attempt to tackle this problem and thus also improve our chances to attain an accurate model of the track later on.

B. *An Angle Based Measure for Curvature*

We first present a preprocessing step to calculate a measure ρ based on the the 19 track sensors s. We then show how this measure can be used to distinguish between six different track segment shapes.

To measure the curvature of the track, we use the sum of the angles of consecutive vectors from the outline of the track. The data needed to calculate measure ρ is reconstructed from the 19 track sensors s_i in the following way. In a first step, the two dimensional vectors $\vec{s_i}$ corresponding to the track sensor values are reconstructed by using a local coordinate system with the car at the origin (see Figure 2 b)).

$$\vec{s_i} = \begin{pmatrix} -\cos(i \cdot \frac{\pi}{18}) \cdot s_i \\ \sin(i \cdot \frac{\pi}{18}) \cdot s_i \end{pmatrix}, \quad i \in \{0, \ldots, 18\} \qquad (1)$$

As shown in Figure 2 c) to e), the vectors $\vec{t_i}$ defining the outline of the track are then reconstructed from the vectors $\vec{s_i}$.

$$\vec{t_i} = \vec{s}_{i+1} - \vec{s_i}, \quad i \in \{0, \ldots, 17\} \qquad (2)$$

Due to the nature of the track sensor, not all vectors $\vec{t_i}$ are useful for calculating ρ as they include the closing cap of the track outline (which stems from the finite range of the sensors and does not relate to the track shape). We employ a heuristic for removing the cap by identifying the indices of the biggest sensor values from the left idx_l and the right

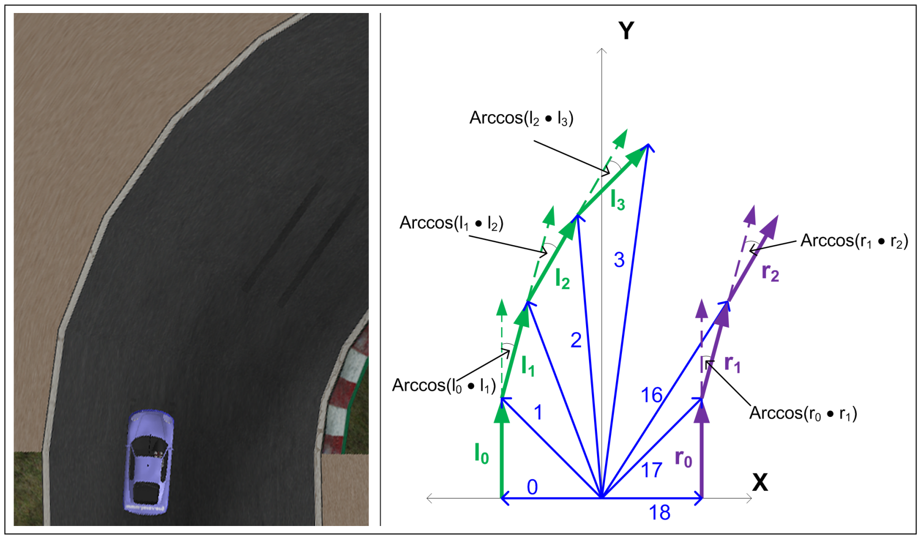


Fig. 3: Left: Screenshot of the game, right: Illustration of the computation of the sum of angles. Vectors $\vec{s_i}$ of track sensor s are colored blue, the normalised vectors $\vec{l_i}$ refering to the left side of the track outline are colored green. The purple vectors $\vec{r_i}$ (also normalised) give the right side of the track outline. Not all vectors $\vec{s_i}$, $\vec{l_i}$ and $\vec{r_i}$ are drawn for the sake of readability.

idx_r, respectively (see figure 3). The two indices usually only differ on wide tracks, whereas on narrow tracks $idx_l = idx_r$. Based on these two indices, two normalized sets of vectors are computed:

$$\vec{r_i} = \frac{\vec{s}_{i+1} - \vec{s_i}}{||\vec{s}_{i+1} - \vec{s_i}||}, i \in \{0, \ldots, idx_l - 1\} \tag{3}$$

$$\vec{l_i} = \frac{\vec{s}_{17-i} - \vec{s}_{18-i}}{||\vec{s}_{17-i} - \vec{s}_{18-i}||}, i \in \{0, \ldots, 17 - idx_r\} \tag{4}$$

where $\vec{l_i}$ and $\vec{r_i}$ are the normalized vectors defining the left and the right edge of the track, respectively. Note that one of the sets gets empty if either $idx_l \leq 2$ or $idx_r \geq 16$. The measure ρ is then calculated by summing up the angles between two consecutive vectors:

$$\begin{aligned}\rho \quad = \quad & \sum_{i=0}^{idx_l-1} [\arccos(\vec{l_i} \cdot \vec{l}_{i+1}) \cdot \frac{180°}{\pi} \cdot \\ & \operatorname{sgn}(\vec{l}_{i,x} \cdot \vec{l}_{i+1,y} - \vec{l}_{i,y} \cdot \vec{l}_{i+1,x})] \\ & + \\ & \sum_{j=0}^{16-idx_r} [\arccos(\vec{r_j} \cdot \vec{r}_{j+1}) \cdot \frac{180°}{\pi} \cdot \\ & \operatorname{sgn}(\vec{r}_{j,x} \cdot \vec{r}_{j+1,y} - \vec{r}_{j,y} \cdot \vec{r}_{j+1,x})] \end{aligned} \tag{5}$$

Figure 3 illustrates Equation 5: The first sum $\sum_{i=0}^{idx_l-1} \ldots$ refers to the sum of the angles between two subsequent vectors on the left side of the track (shown in green in figure 3), the second sum $\sum_{j=0}^{16-idx_r} \ldots$ accordingly for the right side of the track (shown in purple). The equation uses the well known relation between the dot product of two vectors and the cosine of the enclosed angle: $\vec{l_i} \cdot \vec{l}_{i+1} = \cos(\vec{l_i} \angle \vec{l}_{i+1}) \cdot |\vec{l_i}| \cdot |\vec{l}_{i+1}|$. Since the vectors are normalized, this reduces to $\vec{l_i} \cdot \vec{l}_{i+1} = \cos(\vec{l_i} \angle \vec{l}_{i+1})$ (and accordingly for $\vec{r_j}$ and $\vec{r}_{j+1}$). For left hand corners, ρ is positive and for right hand corners, ρ is negative. This is achieved by the terms

$$\operatorname{sgn}(\vec{l}_{i,x} \cdot \vec{l}_{i+1,y} - \vec{l}_{i,y} \cdot \vec{l}_{i+1,x})$$

and

$$\operatorname{sgn}(\vec{r}_{j,x} \cdot \vec{r}_{j+1,y} - \vec{r}_{j,y} \cdot \vec{r}_{j+1,x})$$

in equation 5. These calculate the z-coordinate of the cross product of the 2-dimensional vectors, interpreted as 3-dimensional vectors in the x/y plane, which provides the orientation of the 2 vectors. This property also directly results from the orientation of a basis in $\mathbb{R}^n$ which is defined as the sign of the determinant of a matrix m constructed from the basis vectors as column vectors [4].

$$m = \begin{pmatrix} \vec{l}_{i,x} & \vec{l}_{i+1,x} \\ \vec{l}_{i,y} & \vec{l}_{i+1,y} \end{pmatrix} \tag{6}$$

Vectors $\vec{l_i}$ and $\vec{l}_{i+1}$ are a basis in $\mathbb{R}^2$, and $\operatorname{sgn}(\det(m)) = \operatorname{sgn}(\vec{l}_{i,x} \cdot \vec{l}_{i+1,y} - \vec{l}_{i,y} \cdot \vec{l}_{i+1,x})$. Accordingly, this applies to the second term and vectors $\vec{r_j}$ und $\vec{r}_{j+1}$. We need this information as differentiating between left and right curves is not possible from angles computed by means of the scalar product (these are always positive).

On a straight, $|\rho| \approx 0°$. Entering a corner, $|\rho|$ increases, until it reaches its maximum at the apex. The highest value observed was $|\rho| \approx 90°$ at the apex of a hairpin corner.

C. Recognizing the current situation

We deliberately decided to map the value of ρ to discrete, human understandable, *types*:

- **Straight** - The car is on a straight and no corner can be seen. $|\rho| < 3°$ is mapped to this class, when the biggest value of the 19 track sensors exceeds 90m.

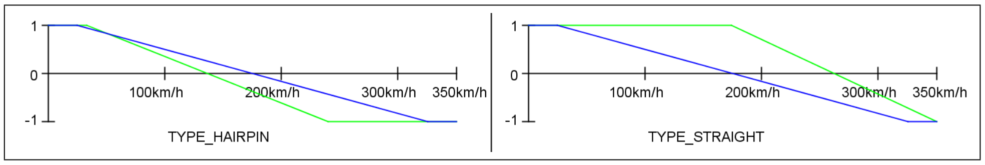


Fig. 4: Visualization of the linear functions for the two situations Straight and Hairpin. Shown in blue is the start individual, in green the best evolved individual out of ten runs of the es. Note that compared to the start inidividual, the evolved behaviour drives slower in hairpin corners (full acceleration only for a speed up to 40km/h, smaller gradient) and a lot faster on straights (full acceleration for a speed up to 175km/h).

- **StraightAC** - Straight, approaching a corner. $|\rho| < 3°$ is mapped to this class, when the biggest value of the 19 track sensors is smaller than 90m.
- **Full** - Full speed corner, which can be driven flat out. $3° \leq |\rho| < 20°$ is mapped to this class.
- **Medium** - Medium speed corner. $20° \leq |\rho| < 35°$ is mapped to this class.
- **Slow** - Slow corner. $35° \leq |\rho| < 45°$ is mapped to this class.
- **Hairpin** - A very tight and therefore slow corner. $|\rho| > 45°$ is mapped to this class.

The boundaries have been derived from a set of preclassified test samples. The current situation consists of one of the six types defined above and the *direction* of the track. In the case of Straight and StraightAC, the direction is *Forward*. For corners, the direction depends on the sign of ρ: Positive values indicate a left hand corner (direction *Left*) and negative ones a right hand corner (direction *Right*).

III. Standard Controller Based on Sensory Data

For our standard controller, we assume that for all six situations defined above, the value for acceleration and brake is a simple one dimensional linear function over the current speed of the car, independent of the current direction. Positive values of the functions stand for acceleration and negative for braking. The gradients of these linear speed matching functions are limited to negative values and the functions' values are limited to the intervall $[-1, 1]$. See Figure 4 for a visualization of two of the learned functions. Each of the six functions can be parameterised by setting the gradient and the location of the zero point.

For steering the car, a heuristic has been borrowed from Kinnaird-Heether (described in [5]) and slightly modified by tripling the steering angle for hairpin bends while doubling it for slow bends.

We learned the speed matching functions using a simple (1+1) evolution strategy [6] on the track Aalborg. This track was chosen because it represents a worst case scenario, with straights and fast corners followed by sharp and therefore slow corners. Starting from a rather conservative setting with full acceleration only set for low speeds, the functions are evolved by driving a fixed time (10,000 gameticks, around one to three laps, depending on the speed) for each newly generated individual and measuring the distance covered. Since two parameters (gradient and zeropoint) for each of the six functions have to be optimized, the search space has 12 dimensions. The evolution strategy used a fixed normalized step size of 2.5% and 500 individuals were evaluated during each run.

As a consequence of the unforgiving nature of the chosen test track, the evolved driving behaviour is very defensive and the controller only reaches a top speed of ca. 200km/h. Nonetheless, the fitness of the start individual (2719m) could be significantly improved by the evolution strategy to 6297m for the best individual out of ten runs (5940m for the worst run, median 6146m).

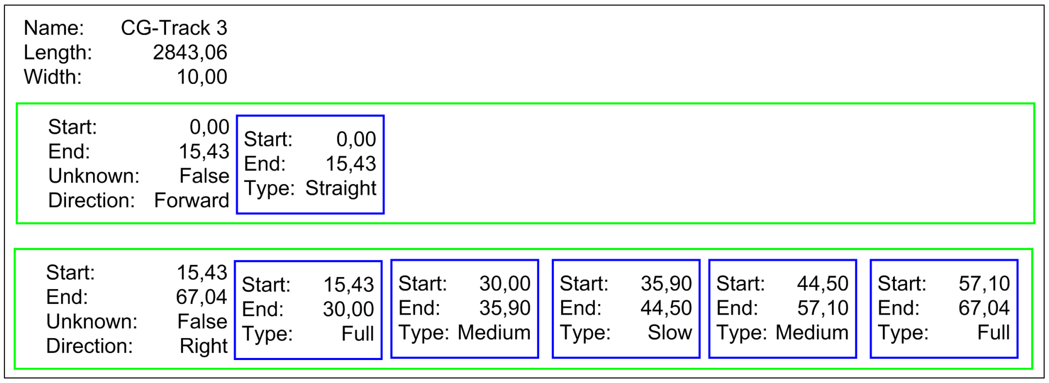
IV. Learning the Track Model

The track model should only serve as an abstract representation of the track, to provide information about the locations of straights and corners and the types of the latter. To achieve this, we simply join parts of the track in which the type of the preprocessed track sensor data has not changed, to form *track subsegments*. On a higher level, track subsegments with the same direction are joined to form *track segments*. Figure 5 shows an example of the datastructure. In the track model, the types Straight and StraightAC are merged to Straight, because the information that the car is approaching a corner (indicated by the type StraightAC) is now implicitly contained in the track model already.

The track model is learned online: At the start of the first lap, it consists of only one track segment with the state *unknown*. While the car is driving around the track, this unknown segment is gradually replaced by the information of the preprocessed track sensor data until the model is finally complete and describes the whole track.

In the case of segments describing a corner, the corner's type is simply the type of the slowest subsegment. The apex of a corner is then defined as the center of the slowest subsegment.

To demonstrate the quality of the learned track model, we compare our model of the track wheel2 with the orginal data structure of Torcs. Figure 6 shows a bird's eye view of the track, the numbers belong to the segments of our track model. The right part of Table I lists the data of each segment, the left part of the table contains the segments derived of the XML files contained in the Torcs distribution. Note that for corners, the XML files do not contain the length explicitly but instead the radius and arc of the corner. The length of a corner has been derived from this data and might not be 100% accurate, since some corners have a varying radius. The table contains the smallest radius of a corner.



Name: CG-Track 3
Length: 2843,06
Width: 10,00

Start: 0,00 End: 15,43 Unknown: False Direction: Forward	Start: 0,00 End: 15,43 Type: Straight

Start: 15,43 End: 67,04 Unknown: False Direction: Right	Start: 15,43 End: 30,00 Type: Full	Start: 30,00 End: 35,90 Type: Medium	Start: 35,90 End: 44,50 Type: Slow	Start: 44,50 End: 57,10 Type: Medium	Start: 57,10 End: 67,04 Type: Full

Fig. 5: Data structure of the track model, using the track CG-Track 3 as an example. Only the first two track segments are shown here in green colour. Within the track segments, the track subsegments are shown in blue. The first track segment represents a short straight, the second one a right hand corner, whose slowest track subsegment is of type *slow*. Therefore the whole corner is classified as being slow.

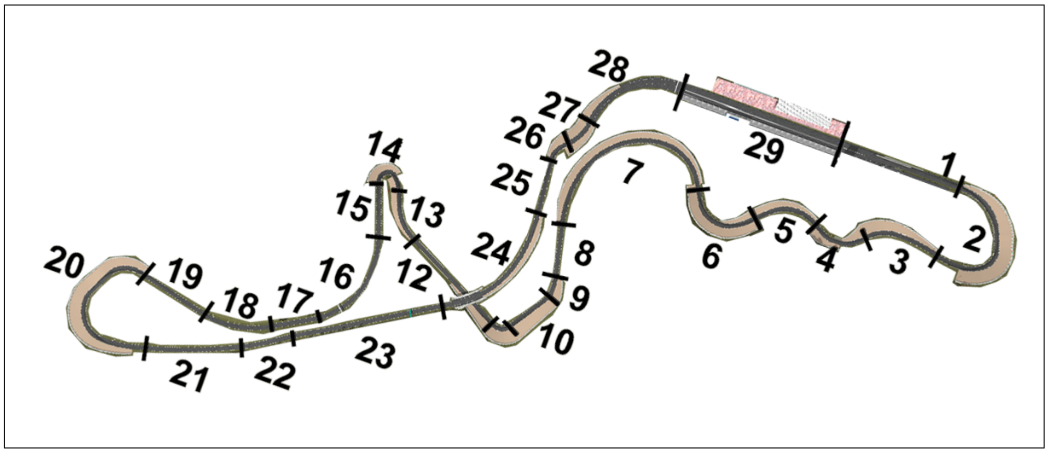


Fig. 6: Birds eye view of the track Wheel 2.

We generally observe a very good matching of learned and game-specified track model, and in this respect, the given example is a good representative of other tracks tested which cannot be shown here due to space limitations.

V. Employing the Track Model

In the last section, we have proposed a technique to learn a track model. Now, we demonstrate that this model can be exploited to achieve a significant improvement of the controller proposed in section III.

We know from oberservations that the evolved driving behaviour of the basic controller is very good at slowing down for slow corners but lacks speed in faster parts of the tracks. Using the knowledge of the track model, the advanced controller is able to plan ahead towards the next corner. The derived plan contains four distinct phases; during each phase of the plan, the evolved driving behaviour of the basic controller is replaced by the planned driving behaviour:

- The first phase of the plan starts at the last apex of a corner. At this point, the car starts to accelerate with full throttle.
- At a certain distance from the next corner, the car is positioned at the outside of the track (left side for right hand corners and vice versa), while still driving flat out.
- Depending on the type of the next corner the car then slows down to a certain target speed (not necessary for full speed corners).
- Between the end of the braking phase and the first apex of the next corner, the car uses a predetermined acceleration value.

The planning module is not used in sections of a track with a number of corners in succesion without straight segments or full speed corners in between (e.g. the S-Curves in Suzuka / wheel2). In this case, the evolved speed matching functions of the basic controller are used for slow and hairpin corners. For medium speed corners, the corresponding speed matching function has been further optimized together with the parameters of the planning module.

Table II lists the various parameters of the planning module. The set of parameters contains the target speeds for the different types of corners (except for full speed corners, which can be driven at full speed, hence no target speed is

Wheel 2									
Torcs XML File					Learned model				
Segment #	Direction	Radius	Start	Length	Segment #	Direction	Type	Start	Length
1	Forward	-	0m	330m	1	Forward	Straight	0m	309m
2	Right	100m	330m	156m	2	Right	Medium	309m	387m
3	Forward	-	486m	35m					
4	Right	40m	521m	188m					
5	Left	70m	709m	192m	3	Left	Medium	696m	190m
6	Right	27m	901m	95m	4	Right	Medium	886m	131m
7	Forward	-	996m	21m					
8	Right	50m	1017m	34m	5	Left	Medium	1017m	178m
9	Forward	-	1051m	30m					
10	Left	90m	1081m	168m					
11	Right	40m	1249m	192m	6	Right	Medium	1195m	195m
12	Forward	-	1441m	50m	7	Left	Full	1390m	546m
13	Left	120m	1491m	518m					
14	Forward	-	2009m	130m	8	Forward	Straight	1936m	109m
15	Right	25m	2139m	74m	9	Right	Medium	2045m	86m
16	Foward	-	2213m	120m	10	Forward	Straight	2131m	93m
17	Right	25m	2333m	42m	11	Right	Slow	2224m	69m
18	Forward	-	2375m	150m	12	Forward	Straight	2293m	276m
19	Right	115m	2525m	96m	13	Right	Medium	2569m	109m
20	Left	15m	2621m	136m	14	Left	Hairpin	2678m	135m
21	Forward	-	2757m	140m	15	Forward	Straight	2813m	107m
22	Right	160m	2897m	59m	16	Right	Full	2920m	317m
23	Forward	-	2956m	30m					
24	Right	150m	2986m	204m					
25	Forward	-	3190m	100m	17	Forward	Straight	3237m	87m
26	Right	200m	3290m	176m	18	Right	Full	3324m	191m
27	Forward	-	3466m	180m	19	Forward	Straight	3515m	146m
28	Left	40m	3646m	99m	20	Left	Medium	3661m	423m
29	Forward	-	3745m	70m					
30	Left	50m	3815m	227m					
31	Left	720m	4042m	375m	21	Forward	Straight	4084m	213m
					22	Left	Full	4297m	130m
32	Right	950m	4417m	99m	23	Forward	Straight	4427m	459m
33	Left	950m	4516m	50m					
34	Forward	-	4566m	300m					
35	Left	180m	4866m	338m	24	Left	Full	4886m	338m
36	Forward	-	5204m	140m	25	Forward	Straight	5224m	105m
37	Right	40m	5344m	46m	26	Right	Medium	5329m	86m
38	Left	60m	5390m	94m	27	Left	Medium	5415m	83m
39	Right	150m	5484m	279m	28	Right	Full	5498m	281m
40	Forward	-	5763m	412m	29	Forward	Straight	5779m	426m

TABLE I: Segments of Wheel 2. The left part of the table contains the segments of Torcs internal data structures, derived from the XML file belonging to the track. The right part shows the learned track model.

needed) and the acceleration values for the approach of the corner's apex (once again no special value is needed for full speed corners). For medium speed corners, we distinguish between normal and "far" corners. The latter are corners which apex is farer away from the beginning of the corner than $5 * trackWidth$. We assume that for this kind of corner, the planned target speed can be higher because the car has more time to slow down within the corner (this assumption is actually dead wrong and the opposite is true, as the optimized parameter set shows). The last two parameters belong to the linear function for medium speed corners of the basic controller, which is used when the planning module is not active.

Starting with rather conservative values and the linear function's parameters from the basic controller, the ten parameters have been optimized with a (3,15) evolution strategy on the track wheel2. The ES used discrete recombination of two uniformly chosen parents and a fixed, normalized step size of 5% for mutation. 450 individuals (30 generations) were evaluated during each of the three runs.

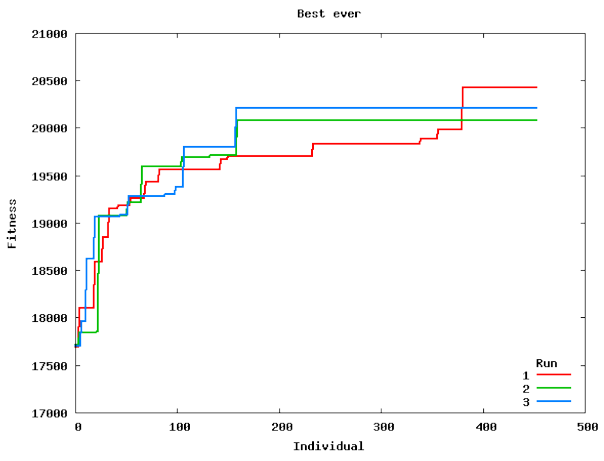
We compared the basic controller and the advanced controller using the initial and the optimized parameter sets with the two best controllers from the 2009 simulated car racing championship: Cobostar [3] and the controller submitted by Onieva & Pelta [7]. Furthermore, we drove the car manually by ourselves, to see if the best controllers fully use the cars potential or if there is still room for improvements. Each controller drove alone for five laps on various tracks. Figure 8 shows the times needed for each controller to finish the five laps.

Compared to Onieva & Pelta and Cobostar, our basic controller is way too slow to be competetive. The advanced controller with the optimized parameter set however is able to close the gap and sometimes outperfom the other controllers on the tracks Wheel2, Forza and C-Speedway. Thus, the incorporation of a track model and the according behavior adaptation of the basic controller has led to an enormous improvement.

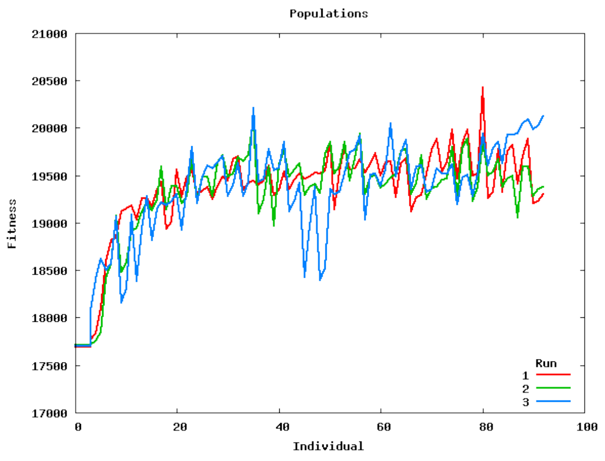
The evaluation on Street 1 produced some interesting results: Cobostar is clearly the fastest controller here, since its driving behaviour has partly been learned on this track [3]. Although the best lap of the advanced controller is faster than

Parameter	lb	ub	Initial value	Optimized value	Meaning
mediumTS	50km/h	350km/h	125km/h	189.03km/h	Target Speed for Medium Corner
slowTS	50km/h	350km/h	80km/h	152.9km/h	Target Speed for Slow Corner
hairpinTS	50km/h	350km/h	60km/h	76.19km/h	Target Speed fr Hairpin Corner
mediumFarTS	50km/h	350km/h	125km/h	157.45km/h	Target speed for far medium corner, Apex $>$ 5 * TrackWidth
mediumTIAcc	0.1	1	0.2	0.21	Acceleration approaching the apex of a medium corner
slowTIAcc	0.1	1	0.2	0.23	Acceleration approaching the apex of a slow corner
hairpinTIAcc	0.1	1	0.2	0.18	Acceleration approaching the apex of a hairpin corner
mediumTIFarAcc	0.1	1	0.2	0.19	Acceleration approaching the apex of a medium far corner
Medium LF Zeropoint	50km/h	350km/h	143.53km/h	193.84km/h	Value of a linear function
Medium LF Gradient	-0.1km/h	-0.01	-0.03	-0.01	Value of a linear function

TABLE II: Parameters of the advanced controller, including the target speeds for different types of corners and the acceleration when approaching the apex of a corner. Lb and ub are the lower and upper bound for each value. The optimized values belong to the second best individual of the first run, since the best individual did not generalize well to other tracks.



(a) Best Ever



(b) Populations

Fig. 7: Fitness development (in m) of three runs for optimizing the parameters of the advanced controller, left: current best fitness, right: fitness values of the consecutive 3 parent individuals. The best obtained solution results in very fragile behavior of the controller

the one of the basic controller, the overall time for the five laps is slower. This is due to a so far undiscovered bug in the stuck detection of our recovery behaviour: The advanced controller loses control in the second to last corner and it takes five seconds until it switches to reverse to get back on the track.

On CG-Speedway Nr 1 the advanced controller loses some time because it drives too slow in the twisting part of the track. On Spa-Franchorchamps, all our controllers are once again slowed down by the slow respond of the stuck detection.

Except for the easy (not to say dumb) tracks, namely CG-Speedway Nr 1 and C-Speedway, none of the controllers is able to match the speed of a human driver. So despite the impressive performance of the current generation of controllers, there remains a lot of room for further improvements on complicated racetracks like Wheel 2 or Spa-Franchorchamps.

VI. Summary and Conclusions

In this work, we propose and demostrate two techniques that together enable to generate a complete and accurate track model. These techniques separately deal with estimating the curvature of the still invisible next part of the track in order to recognize track segments and with aggregating this information into a model which in turn can be employed to plan ahead. Indeed, the rather discrete model data allows to add features that imitate a human driving style as entering a curve with a position very near to the outer track limits to maximize the speed the car may have without leaving the track. There is surely a measurable effect in using such techniques, but additionally, it also makes the controller appear as ‘more intelligent’, where it basically only acts more human-like. Consequently, when driving against such opponents, the race gets an additional push towards realism.

Although we have stated several advantages of applying our techniques, there are also downsides. Discretizing segment curvatures into segment classes requires to setup or learn several different controller behaviors, each of which corresponds to a segment type. Additionally, one has to be aware that learning speed matching functions can lead to overfitting (as documented in figure 7) so that obtained best solutions must be validated concerning their ability to drive well on many tracks. This problem could most likely be reduced by means of learning a speed function that directly

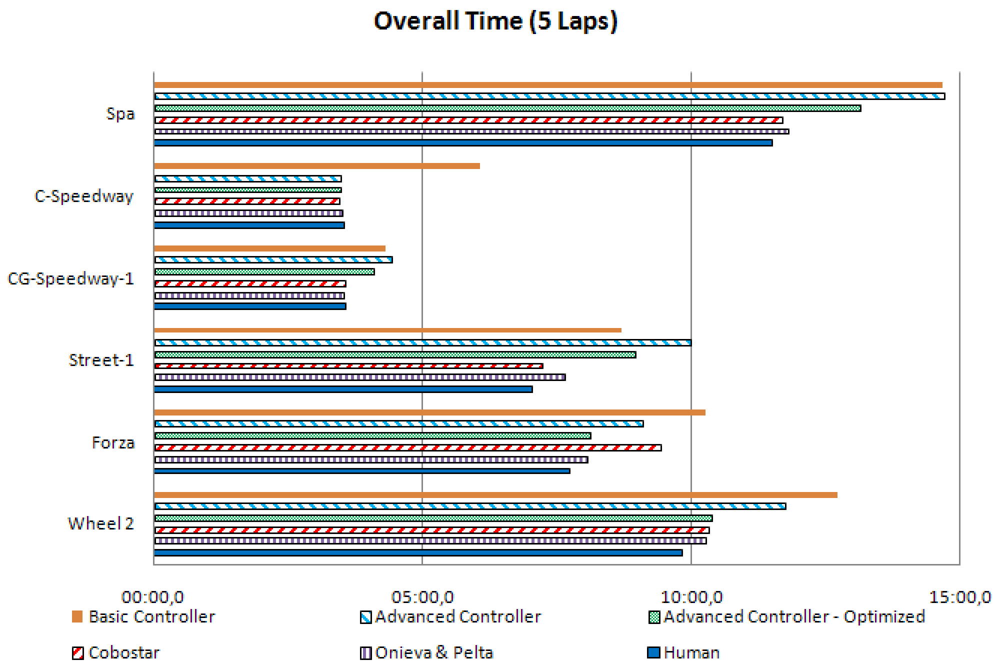


Fig. 8: Overall time for five laps on the various tracks in min:sec. Shorter times are better.

works on the curvature measure and not on its discretization.

It remains to be said that the advanced controller proposed here is still at an early stage of development and its performance is sincerely hindered by the simple steering heuristic and the remaining bug in the recovery behaviour. On some tracks, the evolved target speeds for certain types of corners are too low but the introduction of a warm-up stage for the 2010 simulated car racing championship offers the possibility to further adapt the target speeds for every corner on a specific circuit. Together with a better steering module, this should lead to further improvements of the controller's performance.

Another neat aspect of our planning module is the fact that the handling of opponents could be elegantly incorporated into the planned behaviour, using information from the track model: Either the target speed could be adjusted to avoid a crash or, when on a straight, the car could be re-positioned on the track in order to pass an opponent. We hope to implement these enhancements for the first round of the 2010 championship.

References

[1] D. Loiacono, P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lönneker, L. Cardamone, D. Perez, Y. Saez, M. Preuss, and J. Quadflieg, "The 2009 simulated car racing championship," *IEEE Transactions on Computational Intelligence and Games*, vol. ?, p. ?, 2010, (accepted).

[2] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated car racing championship 2009: Competition software manual," Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, Tech. Rep. 2009.4, 2009.

[3] M. Butz and T. Lönneker, "Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge," in *Proceedings of the IEEE Symposium On Computational Intelligence and Games, 2009*, 2009, pp. 317–324.

[4] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice 2nd edition*. Addison Wesley, 1996.

[5] D. Loiacono, J. Togelius, P. L. Lanzi, L. Kinnaird-Heether, S. M. Lucas, M. Simmerson, D. Perez, R. G. Reynolds, and Y. Saez, "The wcci 2008 simulated car racing competition," in *Proceedings of the IEEE Symposium On Computational Intelligence and Games, 2008*, 2008, pp. 119–126.

[6] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies - a comprehensive introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, March 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1015059928466

[7] E. Onieva, D. A. Pelta, J. Alonso, V. Milanés, and J. Pérez, "A modular parametric architecture for the torcs racing engine," in *Proceedings of the IEEE Symposium On Computational Intelligence and Games, 2009*, 2009, pp. 256–262.