Global to Local for Path Decision using Neural Networks

Dana Vrajitoru
Indiana University South Bend
1700 Mishawaka Ave
South Bend, IN, USA
dvrajito@iusb.edu

ABSTRACT

In this paper, we present a project aiming to improve the path of an autonomous vehicle on a race track in terms of speed and amount of turning. In this part of the study, we use global information about the track to compute an optimized trajectory. This trajectory uses known turning strategies for cars in road curves minimizing the amount of turning. Then we use sample information from the computed path to train a neural network using only local information available to the driver in real-time as input. The goal is to study how well global information can be inferred from local information.

CCS Concepts

• Computing methodologies~Machine learning

Keywords

autonomous vehicles; path optimization, neural networks

1. INTRODUCTION

In this paper, we present a project that is part of developing autonomous vehicles. Our work uses the TORCS system that simulates a car race. Our current aim is to improve the path of the controlled vehicle on a race track in terms of time taken to complete the race and amount of turning that the vehicle does over the whole track.

In this part of the study, we continue the work from [1] where global information about the track's geometry is used to compute an optimized trajectory. Here, we collect local information characterizing the computed trajectory in each frame of the race. Then we use this information to train a neural network (NN) to compute the car's trajectory using only local information as input. The goal of this project is to study whether local information is sufficient in some circumstances to compute a trajectory as well as using global information.

Autonomous vehicles have been generating significant interest in the research community for a good number of years. Recent industrial and academic advances have made it closer now to becoming an everyday reality. Thus, a future where such cars will be commonplace on our roads is not far away. It is important to develop and document good algorithms for these vehicles. We hope that they will contribute to increase traffic safety, reduce the stress of commuting, and increase traffic efficiency in general.

PRAI 2018, August 15–17, 2018, Union, NJ, USA © 2018 Association for Computing Machinery. ACM ISBN 978-1-4503-6482-9/18/08…$15.00 https://doi.org/10.1145/3243250.3243269

Our system extends work started with the EPIC car controller [2] and continued with Gazelle [3]. It was developed in the TORCS (The Open Racing Car Simulator) framework [4] that simulates car races with a variety of available tracks, in a realistic visual environment. The current iteration of our car controller is called Meep.

Several approaches can be found in the literature for track prediction aiming to optimize the performance. For example, in the track segmentation approach, the track is divided into fragments that are classified as pre-defined types of polygons. Then the controller reconstructs a full track model from these polygons, as presented in [5]. Another controller based on the track segmentation principle is proposed by Onieva et al. [6], which includes a fuzzy system working as an opponent modifier unit. A more recent work [7] introduces a driving controller called AUTOPIA, one of the most successful competitors in the simulated racing car competition. These algorithms are also related to map-matching algorithms such as can be found in [15].

Our work on trajectory calculation is related to [8] in that the trajectory for the car is computed for a track situation in both cases, with the goal of optimizing the required time. In their approach, however, they pre-compute safe zones for the car on the road and use them to speed up computation of the trajectory in real time. Incidentally, the shape of the trajectories presented in their paper is consistent with ours. A major difference comes from the fact that in their system, the geometry of the track in known in advance, while in ours, we must infer it from measurements provided at runtime by TORCS.

The driving method employed in Meep uses the pre-computed trajectory as a target and makes decision to steer towards it. This presents some similarity to methods of following a moving target, such as a second car, as presented in [9] and [10], even though the purpose of our research is quite different.

Other approaches to trajectory calculation include the use of evolutionary algorithms. In [11], the authors propose an evolutionary approach called EVOR that computes the trajectory at race time using a fitness aiming to keep the car on the track. This approach uses a pre-constructed model of the track using a method similar to ours. A related approach is presented in [5] where the controller uses an evolutionary learning system to plan the path ahead for the car.

Optimal paths within road constraints can be found in several papers. In this work, we are using the curve shapes computed in [12], which present similarities with the trajectory presented in [8].

Neural networks have been used widely for robot guidance systems and even for pilots trained under the TORCS system. For example, in [13], a NN is used to decide on an appropriate speed for an autonomous car pilot. In [14], the authors use a NN to mimic a trajectory chosen by a human pilot.

The paper is organized in the following way. Section 2 starts with definitions and general purpose settings for our research. Section 3 describes our model, the environment, and the methodology used in our research. Section 4 presents experimental results with the neural network. The paper ends with conclusions.

2. General Settings

In this section, we introduce notations and definitions that are necessary in the description of our model. We also describe the TORCS application and the general settings of our project.

2.1 The TORCS Environment

TORCS attracts a wide community of developers and users, and it is the platform for popular competitions organized every year as a part of various international conferences [2]. The program is organized as a server which implements races combining multiple cars on a variety of tracks. A client module can be written by the user returning the actions of a controlled car. The server provides information about the track, the car, and the opponents [7]. The client can control the steering wheel ([-1, +1]), the gas pedal ([0, +1]), the brake pedal ([0, +1]); and the gearbox (-1 through 6) [2].

The application provides multiple race tracks of various length and difficulty. We have chosen four tracks for this particular research: Alpine 2, E-Road, E-Track4, and E-Track5. The first one is the most difficult track, with several points of sharp turns presenting a challenge even at low speeds. E-track 5 is the shortest and easiest, and chosen initially to calibrate the path generating algorithm. Figure 1 shows a screenshot of the Alpine 2 track at a difficult point where the road enters a tunnel at a very sharp right angle. We can also see the TORCS dashboard with various indicators and the mini-map in the top right corner.

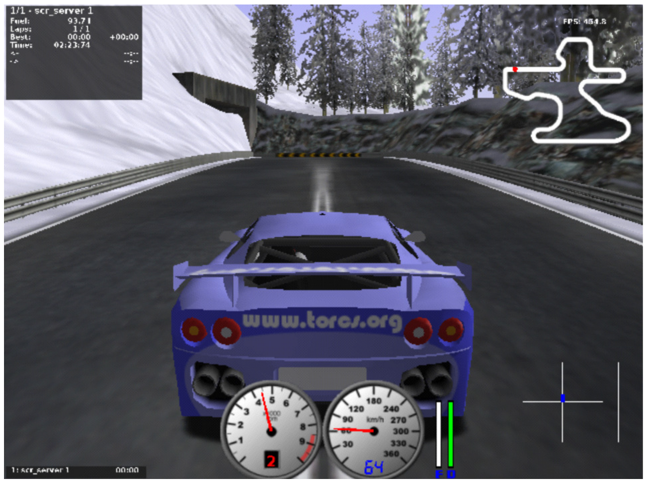

Figure 1. The Alpine 2 track in TORCS

2.2 Notations and Definitions

TORCS provides information about the car state, including a lateral position of the car on the road. If we draw a perpendicular to the centerline of the road from the car center, that we call a *transversal line* or vector, the lateral position shows how far we are on this line from the center of the road. Thus, a value of 0 represents the center of the road, while positions of 1 and -1 represent the borders of the road on each side, as shown in Figure 2.

We define a *trajectory* for the car as a function r(*t*) returning a value in the interval [-1, 1] for each point of the track's centerline *t*, representing the target lateral position of the car on a perpendicular line to the centerline. Figure 2 illustrates this definition.

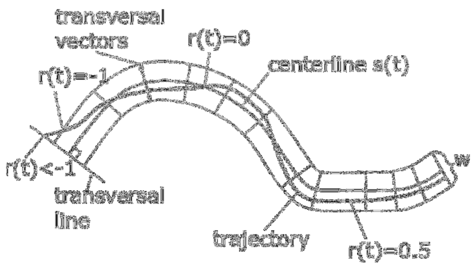

Figure 2. Road and trajectory definition

The function r(*t*) can take values out of the interval [-1, 1] if the car exits the road, but we assume that this is not a desirable trait in such a target function.

Theoretically, the trajectory is a continuous function. Practically, it is discretized on the set of points of the centerline where the car is found in each frame of the application. One constraint that arises from the particular setting of this problem is that the trajectory should be fast to compute. Thus, it must allow the car's pilot to communicate with the race server quickly enough for the car not to exit the road or crash against a hard shoulder.

A trajectory computed on the spot by a NN of modest size is not computationally expensive and can be used directly in the race. However, a pre-computed trajectory that is too dense can both require too much memory and slow down the pilot too much through the search process. To alleviate these issues, we only store a reasonably dense trajectory and interpolate the values in between key points.

The notion of *curvature* of a curve is used extensively through the paper. For a continuous curve s(*t*) defined in a space with more than one dimension, the first derivative s'(*t*) is the tangent to the curve, while the second derivative s''(*t*), is called the curvature, and is normal to the curve. If the curve represents the motion of an object, the first derivative is the velocity, while the second one is the acceleration. For the current project, the curvature is not of great interest as a vector. The relevant parts, in our case, are the norm of the vector and the orientation.

In our model, the centerline of the road can be used as the curve, as well as either of the two borders of the road. However, since this is a simulated environment, the curve is not continuous, but composed of a sequence of small line segments. We have used the change of slope between adjacent line segments as a measure of the curvature of the road. As the segments are not directly available, we compute the angle between observed adjacent road segments through the sensors available in TORCS.

3. Algorithms Description

In this section, we will describe our approach to the problem and the different stages in the implementation of our model.

3.1 The Meep and Gazelle Pilots

Let us first introduce the outline of the autonomous pilots used in this research.

The Gazelle pilot [3] continues and improves the ideas used for the Epic controller [2]. It has two modes: the first one is procedural, using the measurements from the TORCS sensors to compute a target direction and speed, and the second one uses a trained NN for the direction unit. In this study, we have only used the procedural part of Gazelle to compare with the current model, and from it, only the direction unit. The experiments presented in this paper all use constant speed.

The procedural pilot that we wrote for Epic and subsequently improved for Gazelle contains several control units. We start by determining the target angle for steering, which is the first element of driving, as it establishes the trajectory. An opponent modifier unit may change this target angle based on opponents' presence. Next, a target speed is chosen so that the car can achieve a turn by the target angle, while also trying to maximize the speed based on a given limit. This speed change is translated into acceleration or braking and can also be changed by the opponent modifier unit. The gear is adjusted in the last place.

TORCS provides 19 distance sensors in each frame of the race, centered on the car's direction of movement and spawning in both directions by increments of 10 degrees. Each of them returns the free distance ahead to the border of the road in their respective direction. These measurements are capped to a maximum of 100m.

The target direction in Gazelle is obtained starting from the central sensor that points in the car's direction of movement, and scanning left or right using the sensors provided by TORCS, towards the direction where the distance is increasing. The last sensor before the free distance starts decreasing is returned as the target direction. Based on it, the steering value is equal to the target angle divided by *maxAngle*, the maximum turning angle available for the vehicle. Figure 3 illustrates this idea.

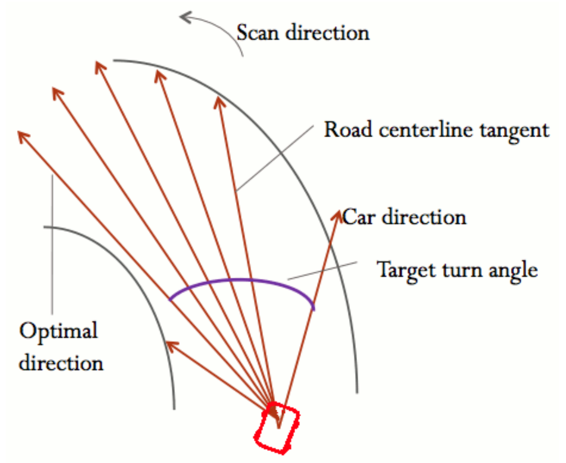

Figure 3. Gazelle steering unit

The driving method employed in Meep uses a target trajectory that can be either a pre-computed trajectory, or one calculated on the spot, and makes a decision to steer towards it. This presents some similarity to methods of following a moving target, such as a second car, as presented in [9] and [10], even though the purpose of our research is quite different. In addition, Meep inherits the Gazelle recovery behavior for cases where the car has exited the road, or got stuck at an inconvenient angle after bumping against the road shoulder.

The target direction unit of Meep is used primarily in this research. The pilot requires a target value for the trajectory in the interval [-1, 1], representing an objective value for the lateral position of the car on the road in each frame of the race. The steering value is computed using two measurements: the current lateral position of the car on the road, and the current angle between the car's axis and the centerline of the road. Both of these are provided by TORCS.

Thus, the target steering angle is calculated first by reversing the current angle with the road centerline, to align the car with the road. Then, we add the difference between the current lateral position on the road and its target value, multiplied by a parameter. To avoid turning too sharply when the road position is far off, we use the square root of the difference if its absolute value is larger than 1. For the same reason, we cap this difference at 2.5. The resulting steering value is computed the following way, then clamped to the interval [-1, 1]:

```
$deltaTraj = \max(-2.5, \min(2.5, targetTraj - currentTraj));$
if ($|deltaTraj|>1$)
    $deltaTraj = deltaTraj / \sqrt{(deltaTraj)};$
$steer = (roadAngle + k_t * deltaTraj)/maxAngle;$
```

where *targetTraj* is the target trajectory, *currentTraj* is the current lateral position of the car on the road, *roadAngle* is the current angle of the car with the road centerline, and k_t is an adjustable parameter.

3.2 Computing the Global Trajectory

To compute an optimized trajectory based on global information, we started by mapping the curvature of the tracks. The curvature is computed locally based on sensor data provided by TORCS. We ran the pilot on a constant trajectory at a constant speed through each of the tracks and recorded the values of the curvature at regular intervals.

The first step is to compute the curvature of the road using the distance sensors in TORCS. Figure 4 illustrates this process. First, we draw a line from the car's position C perpendicular to the road centerline until it crosses the road's border in A. The direction of this line is based on the angle of the car with the road. The angle between this line and the car's direction is equal to 90^o-car angle. Then, we use the closest values from the sensors, marked in the figure as distance probes, to map the triangle ABC, connecting the car with A and with the intersection point B of the next sensor line with the border of the road. This allows us to compute the curvature angle as marked in the figure.

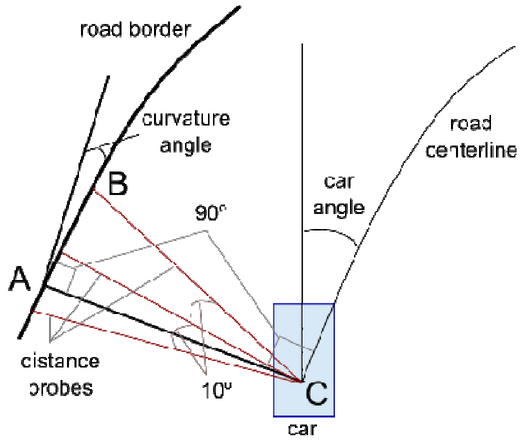

Figure 4. Curvature mapping

Once the curvature is mapped, a trajectory is computed using these stored values. We used the shape of curves that optimizes the overall time it take to go through a curve, as computed in [12] and shown in Figure 5 on the left.

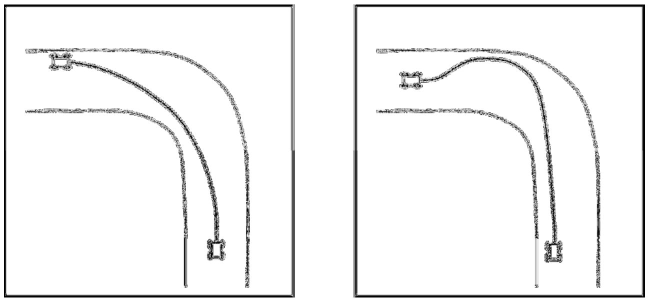

Figure 5. Optimal trajectory through a curve minimizing the time (left) and maximizing the exit velocity (right)

The trajectory is computed in three phases which are described more in detail in [1]. Phase one goes through the entire track and assigns values of 0 to all the long stretches where the track is almost straight, and 0.8 and -0.8 respectively to all the long stretches where the road turns in the same direction. The value 0.8 was chosen based on the width of the car, so that the side closest to the border is still inside the track. These values are assigned leaving a buffer zone of a few meters at the beginning and end of each continuous stretch.

Phase 2 goes through the entire trajectory again and patches the pieces of the track between those assigned during Phase 1. This part uses a linear interpolation of the trajectory values between the end of one stretch and the beginning of the next one. Phase 3 smooths the resulting trajectory even more by using anti-aliasing techniques.

Multiple experiments were done with the trajectory calculating algorithm and variations of parameters such as the value of the curvature for which the road can be considered almost straight or the length of the buffer zones between the different stretches of constant trajectory. These results are presented in [1], and summarized in Section 4.1.

3.3 Neural Network Training

We proceeded to collect training data for a neural network to learn to compute the car trajectory. For this, we ran the Meep pilot using the pre-computed trajectory based on curvature mapping on each of the four selected tracks. For each frame of the race, we recorded as potential input values, the current values of the distance sensors, the current road angle or the car, current lateral position (or trajectory value), and the current value of the curvature, and as output value, the computed trajectory for the location.

In terms of organizing the data into training and testing sets, we used the *cross-validation* method. Thus, for each individual track, we used the data collected from the other three tracks for training purposes, and the data collected from the track itself for testing purposes. This way, we also examine whether data collected from one or more tracks can be used on new tracks.

The first set of experiments we did involve using the curvature only as input for the NN. We experimented with two settings, of one hidden layer of 3 neurons, and of two hidden layers with 5 and 3 neurons respectively. After extracting the curvature and trajectory coordinates from the data and removing duplicates, the size of the data turned out small enough to be used for training without further filtering. The data set was shuffled after each training pass. Experiments where the data set is not shuffled have led to a lower performance of the NN.

For the second set of experiments, we added one more variable as input. To choose the second parameter, we computed a statistic of how much the input values for each of the distance sensors vary when the values of the curvature are constant but not the value of the trajectory. Thus, we computed the sum of the absolute difference of the value of the sensor coordinate for all the pairs of data x and y for which the value of the curvature is the same but the value of the trajectory is different:

$$\Sigma_{\{(x,y)|curv(x)=curv(y),\ traj(x)\neq traj(y)\}} |sensor_i(x)-sensor_i(y)|$$

This measure is meant to show which of the sensors' values is the most significant in describing why the output varies if the input used so far does not. The central sensor, following the direction of the car's axis, returned the highest value for this measure for all the tracks. Thus, for this experiment, we used two input variables, the curvature and the central sensor, and one output variable. For this part, we used a network topology with two hidden layers, of 5 and 3 neurons respectively. Experiments with larger networks did not improve the performance.

For the second set of experiments, the data turned out to be of a too large size to be used efficiently for training directly. For this part, we have filtered the data using intervals of 0.1 size for the output value, and selecting a representative subset of data in each interval. Through the filtering process, for each such interval we capped the selected number of data points to a constant. Through this process, we aimed to have a uniform distribution of the data points over the interval of output values. This technique is similar to the data filtering described in [3].

For these experiments, the testing has been done on unfiltered data, for a better comparison with the other results. Even so, the numbers are not directly comparable because the test sets are larger when we extract two input parameters instead of one.

Table 1 and Table 2 show the results of training and testing the NN for each of the tracks. We performed 5000 training iterations in each case and selected the best average error obtained in each case.

Table 1. Best training error for the NN

Track	1Input 1L	1Input 2L	2Input 2L
Alpine2	0.2004	0.3269	0.3961
E-Road	0.2189	0.2669	0.3915
E-Track4	0.1733	0.2020	0.3846
E-Track5	0.2093	0.2580	0.3911

Table 2. Best test error for the NN

Track	1Input 1L	1Input 2L	2Input 2L
Alpine2	0.0025	0.0023	0.0042
E-Road	0.0039	0.0038	0.0080
E-Track4	0.0043	0.0030	0.0055
E-Track5	0.0042	0.0041	0.0113

These tables show that the network with one input and two hidden layers generally performs better than the one with one single layer. The testing error is generally smaller than the training one because the data set it is calculated on is smaller. We also note from these results that a lower training error does not always correspond to a lower test error due to the fact that the data sets used in these cases come from different tracks. However, there is an overall correlation of the results.

As we will see in Section 4.2, the network with two input parameters and two hidden layers consistently converges towards calculating a constant trajectory. This network is minimizing the error by producing values in a very small interval around a median value for the trajectory. It would be interesting to study for future research why the extra parameter causes this to happen and if a larger network might solve this problem. Another avenue for future research consists in finding different methods of choosing the extra parameter(s) and seeing how that choice can influence the outcome.

4. Experiments in Race Setting

In this section, we present the experiments performed with the trained neural network as used to pilot the car in race conditions and compare it with other pilots previously used in our project.

4.1 Benchmark Data

Let us start by introducing the pilots that we will use for benchmark in these experiments, as well as their performance on the chosen tracks. Thus, in this subsection, we present some of the results that we will use to compare with the performance of the new model. More elaborate results are presented in [1]. We ran the experiments on the four selected tracks in TORCS, as mentioned in Section 2. The curvature of the road was mapped for each of them and an optimized trajectory was computed based on this information. The shape of the trajectory was chosen based on known trajectories minimizing the amount of turning done while traversing the curves, such as found in [8] and [12].

For each of these tracks, we summarize the best results obtained by the computed trajectories in terms of time (seconds) required to complete one lapse of the track and total amount of turning (radians) done by the car. Since the object of this research is optimizing the trajectory, we kept the speed constant in these experiments. We performed three runs for each setting, of speeds 50 km/h, 80 km/h, and 120 km/h respectively.

We also present here the results obtained by the "simple pilot" provided by TORCS, named here the "constant pilot" because its aim is to keep the car in the middle of the track at all times. In our notations, this corresponds to a target trajectory equal to 0 over the entire track. Finally, a third benchmark pilot is used, which is the pilot called Gazelle, presented in [3], for which we have used the steering component only.

Table 3 through Table 6 show the results of the benchmark pilots on the four tracks. The pilots are listed in the order: constant pilot, the pilot using the trajectory computed using global information, and Gazelle. For each of them, we show the total time in seconds to complete one lapse and the total amount of turning in radians done by the car in that situation. For the global trajectory pilot we used the results obtained with the best settings for each track, as presented in [1].

From these results we can see that the global trajectory outperforms both the constant trajectory pilot and Gazelle in most situations and for all the tracks. It consistently finishes the race in a shorter time than both of the other pilots. The total amount of turning that the pilot does is also almost always lower than for the other pilots.

Table 3. Benchmark pilots for Alpine 2

Speed		Const	Global Trj.	Gazelle
50	Time	278.01	271.39	271.97
	Turn	*821.28*	*795.67*	*814.24*
80	Time	201.21	171.61	172.23
	Turn	*624.12*	*592.56*	*611.51*
120	Time	233.27	231.31	260.17
	Turn	*1794.01*	*2920.52*	*2227.01*

Table 4. Benchmark pilots for E-Road

Speed		Const	Global Trj.	Gazelle
50	Time	241.31	232.35	232.63
	Turn	*936.92*	*794.01*	*824.96*
80	Time	151.97	146.27	146.55
	Turn	*605.13*	*521.01*	*527.34*
120	Time	164.65	100.15	112.10
	Turn	*975.09*	*698.24*	*522.33*

Table 5. Benchmark pilots for E-Track4

Speed		Const	Global Trj.	Gazelle
50	Time	512.91	505.30	507.81
	Turn	*590.45*	*558.36*	*621.24*
80	Time	321.81	317.07	318.57
	Turn	*388.82*	*370.76*	*414.44*
120	Time	216.11	212.92	213.87
	Turn	*304.96*	*294.73*	*327.26*

Table 6. Benchmark pilots for E-Track5

Speed		Const	Global Trj.	Gazelle
50	Time	121.87	115.93	116.13
	Turn	*337.62*	*288.68*	*429.43*
80	Time	76.79	73.42	73.60
	Turn	*231.17*	*199.73*	*262.63*
120	Time	52.67	50.43	50.65
	Turn	*182.18*	*155.58*	*172.72*

For some of the tracks, we notice that it often takes more time to complete the track at 120 km/h than at 80 km/h Alpine 2. The reason why this happens is because the tracks have several locations that are hard to go through safely at high speed, resulting in the vehicle crashing into the shoulder and exiting the road. This causes a momentary loss of speed and requires extra time for recovery maneuvers.

4.2 Experimental Results with the NN

We used a classic NN with perceptron neurons using the tanh function and trained it with backpropagation. In all cases, we ran 5000 iterations of training the NN with the entire set of data, and we selected the set of weights minimizing the error on the test data. We used the best result obtained from multiple runs with different seeds for the random number generator. This is used in assigning the initial weights in the network randomly.

Table 7 through Table 10 show the results of the pilot using a trajectory with a trained NN. The models we have used are marked in the tables by the number of input variables and the number of hidden layers (L). The speed is in km/h, the time in seconds, and the turn in radians.

For all three models, the current or local curvature of the road is used as an input variable. For the third model, we added a second input variable with the distance to the road border measured in TORCS by the central sensor, which is aligned with the center axis of the car. The model with one hidden layer uses three neurons in this layer. The models with two hidden layers use 5 and 3 neurons in these layers respectively. The output of the NN in all cases is a single value representing the target trajectory. This value should ideally be in the interval [-1, 1].

Table 7. NN trajectory pilots for Alpine 2

Speed		1Input 1L	1Input 2L	2Input 2L
50	Time	275.23	273.53	277.87
	Turn	*1692.44*	*1156.77*	*816.15*
80	Time	221.69	173.01	206.97
	Turn	*2105.53*	*936.32*	*651.32*
120	Time	280.71	236.49	231.81
	Turn	*1706.88*	*1261.78*	*1219.14*

Table 8. NN trajectory pilots for E-Road

Speed		1Input 1L	1Input 2L	2Input 2L
50	Time	234.53	234.97	240.69
	Turn	*1245.47*	*1180.16*	*951.56*
80	Time	148.17	148.31	151.59
	Turn	*911.35*	*841.97*	*619.22*
120	Time	102.53	103.96	138.35
	Turn	*1029.69*	*917.16*	*837.24*

From these tables, we can see that the NN topology with one input parameter (the curvature) and two hidden layers performs the best of the three models we've tried. For the Alpine 2 and E-Track4 tracks, the time taken by this pilot to complete the race is the lowest of the three pilots except at the highest speed, where bumps against the road shoulder caused it to lose some time. For E-Road and E-Track5, the two topologies with one input performed very similarly to each other and better than the third pilot. These two tracks are in fact easier than the other two, and the pilots are able to complete them without incident even at the high speed.

Table 9. NN trajectory pilots for E-Track4

Speed		1Input 1L	1Input 2L	2Input 2L
50	Time	511.05	507.91	512.01
	Turn	*1015.09*	*1115.39*	*613.13*
80	Time	320.61	318.87	321.25
	Turn	*724.24*	*831.91*	*411.36*
120	Time	215.63	223.99	215.75
	Turn	*661.12*	*867.01*	*328.33*

Table 10. NN trajectory pilots for E-Track5

Speed		1Input 1L	1Input 2L	2Input 2L
50	Time	116.70	116.79	119.93
	Turn	*389.57*	*388.19*	*313.33*
80	Time	73.97	74.02	75.95
	Turn	*276.68*	*275.75*	*207.69*
120	Time	50.83	50.87	52.13
	Turn	*233.59*	*235.00*	*158.72*

The performance of the model with two input parameters and two hidden layers is very similar to the benchmark pilot called constant, which keeps the car in the center of the road through the entire race. This is because the NN with this topology had consistently converged to a setting of weights giving an output in a very small range, making it almost constant.

Comparing the performance of the pilots using a NN with the benchmark pilots, we did not expect the new system to perform better than the pilot based on the computed trajectory. The goal is to obtain a close performance. For all the tracks and most of the settings, the best NN pilot was faster than the constant pilot and almost as good as Gazelle and the computed trajectory pilot. In these cases, it is only about half a second to two seconds behind the benchmark pilots. This means that the NN can learn to replicate pretty close the behavior of the pilot using the precomputed trajectory.

As a future direction of research, we intend to apply trajectory optimization methods to improve the pre-computed path even more. Then a NN can be used to learn to mimic such a trajectory on new tracks without mapping them beforehand.

5. Conclusions

In this paper, we presented a model where we first computed an optimized trajectory based on a mapping of the road, then we collected local data describing this trajectory, and then trained a NN to compute a trajectory from local track data. We used cross-validation to examine whether the system can be trained from data collected from some tracks and then applied successfully to a new track.

Our experiments show that with a careful choice of parameters, the NN can be trained to choose a car trajectory similar to the precomputed one and display a performance close to it. This means that the global information used to compute the trajectory can be translated with close accuracy into the local information used by the NN to compute the trajectory.

As future research, we intend to focus on trajectory optimization methods to improve the pre-computed trajectory before training the NN to replicate it.

6. REFERENCES

[1] Vrajitoru, D. 2018. Optimal Curves in Trajectory Calculation for Autonomous Cars. Submitted to the *International Journal of Modelling and Simulation*.

[2] Guse, C. and Vrajitoru, D. 2010. The Epic adaptive car pilot, Proc. *Midwest Artificial Intelligence and Cognitive Science Conference*, South Bend, IN, 30–35.

[3] Albelihi, K. and Vrajitoru, D. 2015. An application of neural networks to an autonomous car driver, *Proc. The 17th International Conference on Artificial Intelligence*, Las Vegas, NV, 716–722.

[4] Wymann, B., Dimitrakakis, C., Sumner, A., Espié, E. Guionneau, C., and Coulom, R. 2013. TORCS, The Open Racing Car Simulator, v1.3. http://www.torcs.org.

[5] J. Quadflieg, J. and Preuss, M. 2010. Learning the track and planning ahead in a racing car controller, *Proc. IEEE Conference on Computational Intelligence and Games (CIG10)*, Copenhagen, Denmark, 395–402.

[6] Onieva, E., Pelta, D. A., Alonso, J., Milans, V., and Prez, J. 2009. A modular parametric architecture for the TORCS racing engine, *Proc. IEEE Symposium on Computational Intelligence and Games*, 256–262.

[7] Onieva, E. and Pelta, D. A. 2012. An evolutionary tuned driving system for virtual racing car games: *The AUTOPIA driver, International Journal of Intelligent Systems*, 27(3), 217–241.

[8] Liniger, A. and Lygeros, J. 2015. A viability approach for fast recursive feasible finite horizon path planning of autonomous RC cars, *Proc. 18th International Conference on Hybrid Systems: Computation and Control, (HSCC '15)*, New York, NY, USA, 1–10.

[9] Nippold, R. and Wagner, P. 2012. Calibration of car-following models with single- and multi-step approaches, *Proc. Winter Simulation Conference*, (WSC '12), 410:1–410:11.

[10] Zeyu, J. et. al, 2016. Calibrating car-following model with trajectory data by cell phone, *Proc. Second ACM SIGSPATIAL International Workshop on the Use of GIS in Emergency Management, (EM-GIS '16)*, New York, NY, USA, 12:1–12:6.

[11] Nallaperuma, S., Neumann, F., Bonyadi, M. R., and Michalewicz, Z. 2014. EVOR: An online evolutionary algorithm for car racing games, *Proc. Genetic and Evolutionary Computation Conference (GECCO '14)*, New York, NY, USA, 317–324.

[12] Velenis, E. and Tsiotras, P. 2005. Minimum time vs maximum exit velocity path optimization during cornering, *Proc. 2005 IEEE International Symposium on Industrial Electronics*, Dubrovnik, Croatia, 355–360.

[13] Kim, K.-J., Seo, J.-H., Park, J.-G., Na, J.C. 2012. Generalization of TORCS car racing controllers with artificial neural networks and linear regression analysis. *Neurocomputing* 88 (2012) 87–99.

[14] Muñoz, J., Gutierrez, G., Sanchis, A. 2010. A human-like TORCS controller for the Simulated Car Racing Championship. *Proc. of IEEE Symposium on Computational Intelligence and Games (CIG),* Dublin, Ireland, 473-480.

[15] S. S. Rathour, S.S., Boyali, A., Zheming, L., Mita, S., and John, V. 2017. A Map-based Lateral and Longitudinal DGPS/DR Bias Estimation Method for Autonomous Driving. International Journal of Machine Learning and Computing, Vol. 7, No. 4, August 2017, 67-71.

Authors' background

Your Name	Title*	Research Field	Personal website
Dana Vrajitoru	Associate professor	Intelligent systems, computer graphics	www.cs.iusb.edu/~dvrajito

*This form helps us to understand your paper better, the form itself will not be published.

*Title can be chosen from: master student, Phd candidate, assistant professor, lecture, senior lecture, associate professor, full professor