Natural Computation Project
Group 17

Giovanni Puzo - 0622701109 - `g.puzo@studenti.unisa.it`
Salvatore Ventre - 0622701343 - `s.ventre@studenti.unisa.it`

ACADEMIC YEAR 2020/2021

Contents

ABSTRACT - The aim of this work is to study and improve the performance of the controller Snakeoil for the simulator TORCS (The Open Source Racing Car Simulator) through the search for the best configuration of its parameters. This work is based on the use of natural computing algorithms, in particular the DE (Differential Evolution) one.

KEYWORDS - Torcs, Snakeoil, Diffential Evolution, Car Racing.

1 Introduction

In the field of artificial intelligence, we often find ourselves comparing the results of different algorithms on games, as these, on the one hand, represent a benchmark and, on the other one, are a stimulus for researchers. In this work we present the **TORCS** racing car simulator and apply natural computing techniques to improve the performance of a given controller, **Snakeoil**, with the aim of arriving in the first positions, minimizing the needed time and the caused damage to complete *two laps* on a circuit, competing against *eight opponents*.

2 Setup

2.1 Simulator

The Open Racing Car Simulator TORCS has often been used as a platform for the development and improvement of intelligent agents for autonomous driving in a competitive environment. It is an ideal platform for optimization algorithms in fact, from one hand, TORCS features a sophisticated physics engine, that takes into account many aspects of the racing car (e.g. collisions, traction, aerodynamics, fuel consumption, etc.) as well as a 3D graphics engine for the visualization of the races. On the other hand, TORCS was not been only conceived as a free alternative to commercial racing games, but it was specifically devised to make it as easy as possible to develop an own car controller.

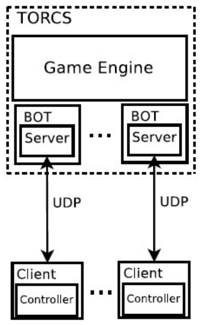


Figure 1: The architecture of the API for controller development.

In order to develop a car controller, the simulator is based on a `Client-Server` model, which communicate through UDP connections: the Server communicates the status of nineteen different sensors (e.g. angle between the direction of the car and the axis of the track, gear inserted) to the intelligent agent, which responds, based on the state in which it is located, through the use of seven different actuators (e.g. accelerator, brake, gear change). In particular, the following table shows some of the used sensors for the proposed optimizations:

Name	**Description**
distRaced	Distance (m) covered by the car from the beginning of the race.
trackPos	Distance (m) between the car and the track axis.
lastLapTime	Time (s) to complete the last lap.
racePos	$[1, \dots, 9]$ Position in the race with respect to other cars.
damage	$[0, +\infty)$ Current damage of the car.

Table 1: Example of used sensors.

2.2 Client

The task was not developed with an agent from scratch, but using the Python client `Snakeoil`, which presents discrete performance on all possible circuits, but not a strong specialization on one in particular. This client has been developed in the first half of the years '10 and written in `Python 2.7`; because of this deprecated version, the code has been converted to `Python 3.x`. The Client is structured to make decisions based on a set of parameters, which are attached to the Python script. In the original version there are 114 parameters: for the optimization problem, the resulting dimensionality is considerable. However, an initial analysis of the client's code and behaviour showed that about half of these parameters were never taken into account, and therefore their number could be reduced to 48, in favour of a lower dimensionality.

2.2.1 Parameters

Below, the table containing the default values of the 48 parameters:

Parameter	**Initial Value**
backontracksx	70.8506
backward	1.6692
brakingpacefast	1.0383
brakingpaceslow	2.2426
carmaxvisib	2.2919
carmin	33.8688
clutchslip	90.3410
clutchspin	50.2910
consideredstr8	0.0100
dnsh1rpm	4013.5036
dnsh2rpm	6313.0396
dnsh3rpm	7018.2574
dnsh4rpm	7430.3105
dnsh5rpm	7483.4711
fullstis	0.8199
fullstmaxsx	20.0708
ignoreinfleA	10.7936
obvious	1.3425
obviousbase	95.3072
offroad	1.0003

Continued on next page

Continued from previous page

Parameter	Initial Value
oksyp	0.0659
pointingahead	2.1964
S2cen	0.5113
S2sen	3.0044
safeatanuspeed	0.0013
seriousABS	27.9135
skidsev1	0.5766
slipdec	0.0180
sortofontrack	1.5041
spincutclip	0.1025
spincutint	1.7450
spincutslp	0.0514
st	689.9554
stC	329.1537
steer2edge	0.9502
str8thresh	0.1438
stst	494.4079
sxappropriatest1	16.0833
sxappropriatest2	0.5520
sycon1	0.6429
sycon2	1.0002
upsh2rpm	9528.4661
upsh3rpm	9519.1839
upsh4rpm	9526.7241
upsh5rpm	9652.7872
upsh6rpm	11914.5027
wheeldia	0.8554
wwlim	4.4870

3 Design of solution

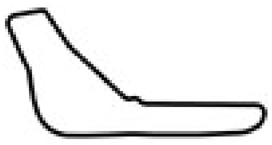
The goal is to train a controller able to race for 2 laps on a set of unknown tracks against other controllers, trying to reach the finish line in the shortest possible time, causing as little damage as possible and arriving in the first positions.

The design specifications requires to evolve the controller on at least two of the 4 circuits provided (Forza, Wheel-1, CG-Track-2, E-track-3), competing with at least 8 specific opponents (damned1, damned3, damned4, damned4, damned6, berniwhist 2, berniw hist6, berniw hist7) in at least 2 laps.

3.1 Track analysis

First of all, the provided circuits has been carefully analyzed in order to understand their characteristics; here you can see the track of each of them with the respective lengths:

Forza



Wheel-1

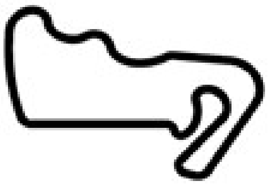
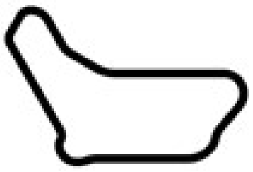


Table 3: Track Map of Forza and Wheel-1.

CG-Track-2



E-Track-3

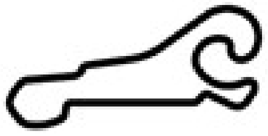


Table 4: Track Map of CG-Track-2 and E-Track-3.

Name	**Length**
Forza	5784.10m
Wheel-1	4328.54m
CG-Track-2	3185.83m
E-Track-3	4208.37m

Table 5: Length of the Circuits.

As you can understand from the images above, you can identify *two* subgroups based on similar characteristics, specifically `Forza` and `CG-Track2` are characterized by rectilinear stretches sufficiently long to allow the achievement of high speeds and wide-radius curves that do not require an excessive decrease in speed; while the other two circuits have tight curves in succession and various chicanes that require sudden changes in gear and trajectory for avoiding to go outside the track.

Based on these considerations, we can divide the circuits in 4 subgroups of complementary characteristics: `[Forza, Wheel-1]`, `[Forza, E-track-3]`, `[CG-Track-2, Wheel-1]`, `[CG-Track -2, E-track-3]`.

The basic idea is to optimize the controller parameters competing in parallel on 2 complementary circuits and then combine the obtained results. This choice is due to the fact that the controller was able to learn the particular pitfalls of each circuit to face them in the most correct way possible, preventing it from specializing on a particular circuit but making it able to generalize, working on a set of information as complete and heterogeneous as possible.

In particular in the first phase, we decide to alternate the various subgroups so that each of them is employed in the 25% of the total number of generations and in two following generations the subgroup is varied; this for avoiding to specialize the controller on the last seen circuits and stimulating it to learn as many features as possible of all the circuits. However, in the second phase, our strategy has been changed due to inclusion of opponents that carries out a lot of randomness: we evolved the controller using only a pair [Forza, Wheel-1] that is the most completed one in our opinion. This aspect will be better discussed in Section 4.1 and Section 4.2 respectively.

3.2 Algorithms

The first phase concerned the choice of algorithms to be used for the evolution of the Snakeoil controller. The problem concerned the optimization of a function, therefore it was decided to tap into the class of evolutionary algorithms because they are designed specifically for this purpose. Among the various evolutionary algorithms proposed, we have decided to restrict the analysis to a single algorithm: **Differential Evolution (DE)**: it works by having a population of candidate solutions (called agents) which are moved around in the search-space according to a fitness function, measuring their ability of adaptation to the proposed problem. If the new position of an agent is an improvement with respect to the previous one, then it is accepted and forms part of the population, otherwise the new position is simply discarded; this process is repeated and by doing so it is hoped, but not guaranteed, that a satisfactory solution will eventually be discovered.

The reasons about the choice of DE are:

- This algorithm is ideal for optimizing functions of real variable, so you did not have to change the representation of the values that the parameters can assume.
- It presents few hyper-parameters: *population size*, *crossover rate (CR)* and *mutation factor (F)*.
- The convergence of Differential Evolution is fast: the strategies of this algorithm take advantage of the best solution found in the parent population and have a faster convergence towards the optimal solution.
- The solution is quite stable, this means that on average it returns the global minimum with different initialization

The fourth point was crucial in our case because, due to a lack of computational power and to time constraints, we were not able to run each experiment many times with different initialization.

The Differential Evolution global optimization algorithm is available in Python via the `Pymoo` package with the `pymoo.algorithms.so_de.DE()` class. It presents various arguments that can be configured: first of all, the *variant* argument that controls the type of differential evolution search that is performed. By default, this is set to *rand1bin* `(DE/rand/1/bin)`, which is a good configuration for most problems: it creates new candidate solutions by selecting random solutions from the population and, for this reason, it has stronger exploration capability but may converge slowly. But there is also an other type of search *best1bin* `(DE/best/1/bin)`: this strategy takes advantage of the best solution found in the parent population and have a faster convergence towards the optimal solution; however, it may become stuck at a local minimum point during multimodal function optimization.

Also, there is the *popsize* argument that controls the size or number of candidate solutions that are maintained in the population; the *mutation (F)* argument controls the number of changes made to candidate solutions each iteration; and finally, the amount of recombination is controlled via the *crossover rate (CR)* argument. Moreover you can choose to use or not the self-adaptive strategy, trough the `dither` argument: this strategy foresees to select F from the interval (0.5, 1.0) randomly for each generation or each individual; it is empirically found that this strategy improves the convergence behavior. Another strategy for adaptive weights can be settled through the `jitter` argument: it foresees to add or subtract a very small value to the weight used for the crossover for each individual.
Another hyperparameter is the *sampling* argument that defines the initial set of solutions which are the starting point of the optimization algorithm. One of the most used is *Latin Hypercube Sampling (LHS)*, a statistical method for generating a near-random sampling distribution. LHS is built as follows: each dimension space, which represents a variable, is cut into n region where n is the number of sampling points, and we put only one point in each region. Taking a single sample from each region increases the likelihood that the full range of the posterior distribution is sampled.

3.3 Parallel computation

We noticed that the time needed to complete a generation is very large. In order to run the algorithm in a reasonable time, we introduced a parallel computation using a network of computers: we split the evaluation of the population on more computers, defining a `master-slave protocol` in which the node master executes the whole DE algorithm and, when the evaluation phase is reached, it splits the individuals among the nodes, sending at each node the individuals to be evaluated by writing them in a file on a shared folder. The slave nodes wait for the own batch of individuals and, when they are received, the evaluation phase is performed; at the end, they write the results in a file on the same shared folder. After all the nodes have completed the evaluation, the master node reads all the results in order, combines them and continues with the next phase.
Using this technique, we were able to drastically reduce the time needed to complete a generation, we were able to finish a generation in about 20 minutes competing on two tracks with opponents, that is the most expensive computation.

4 Experiments

In this section we report the experiments that have been carried out to look for the best possible behavior of the controller. We decided to divide the evolutionary process into two phases:

- **Without Opponents**: in this first phase, the improvement of the parameters has been obtained by competing on the four circuits provided, in the absence of opponents. In fact, the idea was to look for a good level of balance of parameters on all circuits, increasing the maximum speed and decreasing the lap time.
- **With Opponents**: in this second and last phase, the opponents were added, based on the improvements obtained from the previous phase. The presence of opponents has complicated the problem: you have to check the overtaking phase of the opponents and the limit of the damage of the car in any contacts.

4.1 Without Opponents

As we said in Section 3.2, as algorithm we used the Differential Evolution with self-adaptive strategy and the most important aspect was the choice of the appropriate hyper-parameters and also the number of iterations: we consider a population of 240 agents (five-times the number of parameter: $240 = 48x5$) and values of $CR = 0.9$ and $F = 0.8$, that is the classical setting for a wide range of optimization problems. We chose as *variant* argument the *rand1bin*, and for the *sampling*, the *Latin Hypercube Sampling*. Also, during the experiments we used a number of iterations large enough to allow a significant evolution, but at the same time, to avoid the risk of specialization. As we said in Section 3.1, the training was completed by competing the controller in parallel on 2 complementary circuit, alternating the various pairs of tracks.

4.1.1 First Experiment

Initially, the controller was trained for 60 generations and, as fitness function, we chose to consider the average speed of the car and the *distRaced*, *lastLapTime* sensors in order to add a penalty term if car runs more distance than the length of the circuit, that means it is gone outside the track or, after a spin, it is proceeding in the opposite direction. In fact the goal is to increase especially the speed of the car but also the accuracy of trajectories in order to complete the race in the shortest possible time:

$$f = -(V_1 * V_2 - \frac{(D_1 - L_1) * (D_2 - L_2)}{100})$$

where:

- V: Average Speed (m/s), considering distance covered by the car and time to complete this distance.
- D: Distance (m) covered by the car from the beginning of the race.
- L: Length (m) of the circuit. The training is based on two laps, so this value is twice the distance of the single lap of the circuit.

Parameters	**Value**
CR	0.9
F	0.8
Pop Size	240
N. of generations	60
Individuals initialization	*LatinHypercubeSampling*
Tracks	Forza, Wheel-1, E-Track-3, CG-Track-2
Seed	41

Table 6: Summary of the hyperparameters considered.

Moreover, the penalty term subtracted to speed is scaled by a factor of 100 for giving more importance to the speed because this means less time to complete the lap; in fact, running the baseline from which we have to start, we observed that, applying this scaling, there was 1 order of magnitude between the two terms. However, from the logs during the training, we noticed that the adopted scaling wasn't sufficiently and the speed was decreasing too much with respect to the one of the baseline; so we decided to stop the training after 40 generations and to test it immediately in graphic mode for effectively understand if we were right.

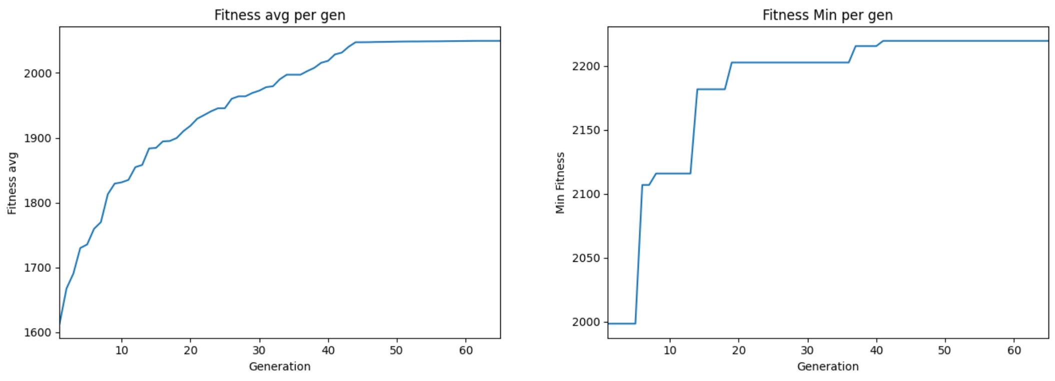


Table 7: Evolution plots for first experiment.

In Figure 7, you can see the plots regarding the minimum and the average of fitness value of all individuals for each generation. As you can see, the average fitness in the last about 20 generations is substantially stable, meaning that evolution is reached a point in which the best individual has difficult to change anymore; this is another reason for which we decided to stop this experiment and test it graphically. After the graphic test, our considerations turned out to be correct, so we don't resume this training but we passed to the next experiment.

4.1.2 Second Experiment

For the second experiment, we tried to resolve the problem mentioned before, that is giving more weight to the speed w.r.t the penalty term, in order to improve the lap time. For this reason, we chose 1000 as scaling factor for the penalty term, setting 2 orders of magnitude between the two terms composing our fitness:

$$f = -(V_1 * V_2 - \frac{(D_1 - L_1) * (D_2 - L_2)}{1000})$$

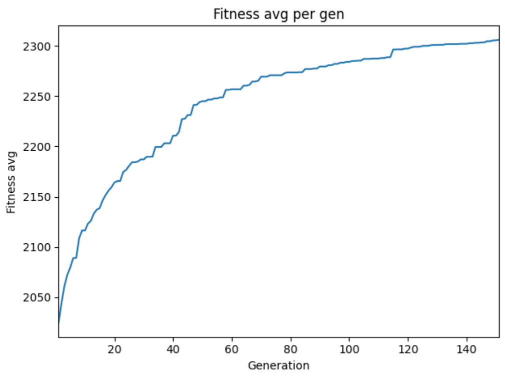
where:

- V: Average Speed (m/s), considering distance covered by the car and time to complete this distance.
- D: Distance (m) covered by the car from the beginning of the race.
- L: Length (m) of the circuit. The training is based on two laps, so this value is twice the distance of the single lap of the circuit.

Parameters	**Value**
CR	0.9
F	0.8
Pop Size	240
N. of generations	152
Individuals initialization	from *first experiment*
Tracks	Forza, Wheel-1, E-Track-3, CG-Track-2
Seed	41

Table 8: Summary of the hyperparameters considered.

We stopped the training at 152_{th} generation because, in addition to the fact that the average fitness is quite stable in the last generations as you can see in Figure 9, we did not have so much time to train the algorithm without the opponents. Moreover, seeing the obtained results reported in Section 5, we were satisfied considering our final purpose to compete with opponents, so we finally decided to not resume this experiment and to pass to training the algorithm with the opponents.



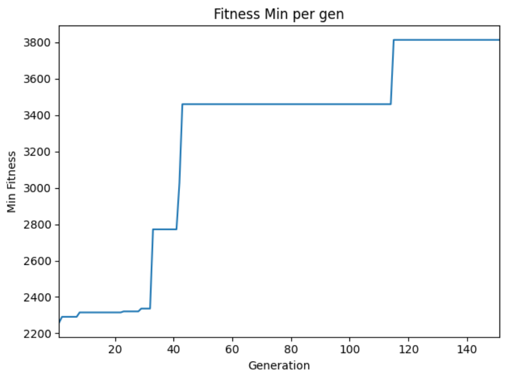


Table 9: Evolution plots for second experiment.

4.2 With Opponents

The first phase of the project was characterized by the training of the controller without the opponents. This phase gave us a good level of the controller, bringing out features such as maximum speed in straightaway and good speed in wide-radius curves. In this second phase our goal was to encourage the car to overtake the opponents in order to get the first positions, causing as little damage as possible; the fitness functions used are characterized by simplicity for favouring the research of the optimum: in general, we worked with *racePos*, *lastLapTime* and *damage* sensor.

We focused also on the study of the opponents: the number should be at least 8 and should include: *damned 1*, *damned 3*, *damned 4*, *damned 5*, *damned 6*, *berniw hist 2*, *berniw hist 6* and *berniw hist 7*. We noticed that the group "damned" has a better performances that "berniw hist" group and this was a good hint to create an homogeneous starting grid and and not to compromise experiments.
The first thing we did at the beginning of this second phase was to test the controller obtained during the last experiment with the opponents. This was useful to understand the main short-comings, problems and the starting level of the controller with the presence of the opponents.

An important aspect to underline, to better understand this second phase of experiments, is about the *racePos* sensor. In fact, we considered the final position of the car as one of the crucial points on which base our next experiments, in particular we wanted to give a sort of "*weight*" to each position with the goal of collecting good positions in each circuits.

For these reasons, we thought about the actual points scoring system of *Formula One (F1)*, that gave us a good adaptation for our aim.

This system is described in the following table:

Position	**Points**
1	25
2	18
3	15
4	12
5	10
6	8
7	6
8	4
9	2

Table 10: Points Scoring System used.

In particular, we denote as $\mathcal{F}(P)$ the function that takes in input the final position of the car ($P = racePos$ sensor output) and gives as output the relative points. For example, if the car arrives in 6_{th} position: $\mathcal{F}(6)$, the car collects 8 points.

We adopt this mechanism for giving more difference to arrive in a certain position rather than in another one. In fact, using directly the position, we have an improvement of 1 for each gained position in the race, which is less considered in the fitness function w.r.t the improvement obtained with the F1 points scoring system, which is at least equal to 2; obviously this argument holds for the same scaling factor by which we can multiply the term depending on the race position.

Another aspect that characterizes the following experiments is about the circuits on which we trained the controller. In fact, all the experiments were performed on only two tracks: `Forza` and `Wheel-1`. The reason of this choice is that the introduction of the opponents carried out a certain degree of randomness in the races generations by generations because their behaviour isn't always the same and the evolution of our controller depends on them; for a better evolution, we wanted to have the clearest possible comparison between the best individual at each generation and to judge the experiments with as little randomness as possible, so we decided to limit the randomness as far as possible avoiding to change the couple of circuit along the evolution. However, we maintained the idea explained in Section 3 of considering, as pair of circuits on which perform the fitness evaluation, two complementary circuits; for this reason, we chose the subgroup with *Forza* and *Wheel-1* as tracks, which we considered the most complete ones for its characteristics.

Finally, regarding the hyperparameters, we continued to choose the setting on the same line of the previous experiments: a population of 240 agents and values of $CR = 0.9$ and $F = 0.8$. Also for the *variant* and the *sampling* we chose respectively *rand1bin* and *Latin Hypercube Sampling.*

4.2.1 Third Experiment

In the first experiment with the opponents, we decided to use only a subgroup of opponents, taking the 5 most competitive ones between the 8 provided, that are the ones belonging to the *"damned"* group; this because in this phase we wanted only to explore the ability of the controller to overtake so, using less opponents, we thought that its features can be most highlighted because five opponents generate less randomness than eight opponents and controller behaviour depends mostly from its ability.

As regards the fitness function, we concentrated our attentions to make the controller able to improve its final position, so we took into account its final position through the scoring system presented in Section 4.2 and the average speed because, increasing it, our car is more stimulated to overtake slowly opponents and to take the best positions, eventually getting the better in a final sprint. However, increasing a lot the average speed, could have brought to cause a lot of damage, hurting with other cars or with the barriers due to a out of track; for this reason a penalty term is added considering the total damage made by the car. Moreover we have multiplied points and speed by a factor of 10 for making all the quantities of the same order of magnitude. Finally the training is based only on two circuits, `Forza` and `Wheel-1`, as indicated in Section 4.2, and our car started in the last (5_{th}) position.

So the used fitness function is the following one:

$$f = -((\mathcal{F}(P_1) + \mathcal{F}(P_2))) * 10 + (\frac{D_1}{T_1} + \frac{D_2}{T_2}) * 10 - (H_1 + H_2))$$

where:

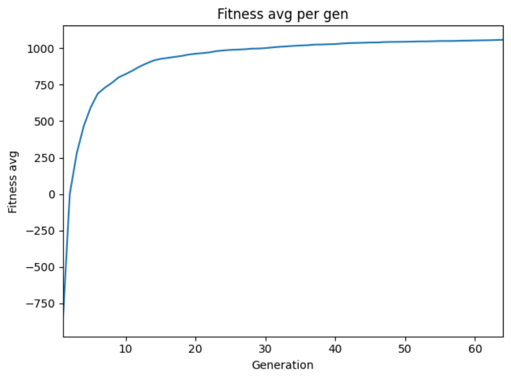
- P: Position $[1, \ldots, 9]$ in the race with respect to other cars.
- T: Time (s) to complete the two laps.
- D: Distance (m) covered by the car from the beginning of the race.
- H: $[0, +\infty)$ Current damage of the car.
- $\mathcal{F}()$: Function of points scoring system.

Parameters	**Value**
CR	0.9
F	0.8
Pop Size	240
N. of generations	64
Individuals initialization	from *second experiment*
Tracks	Forza, Wheel-1
Seed	41

Table 11: Summary of the hyperparameters considered.

As anticipated before, this wasn't a real experiment because the requirements included the race with 8 opponents, not only 5; it serves for testing the main features of the controller evolved with opponents, as regards the overtaking skills, so we stop the evolution after only 64 generations.

First of all, we had encouraging results by the chosen fitness since the earliest generations, as you can see from Table 12; for this reason we decided to continue with the same fitness in the next experiment. Moreover, we noticed that the controller, due to its implementation, exhibits a behaviour for which it tends to follow other opponents at a certain distance, if they are on its trajectory. Distance is also maintained in the turn when, for example, it is doing a tight turn while the opponents ahead it has approached the turn with a wider radius; so the controller is doing an overtake but, reading the sensors output, notices the opponent and widens its trajectory, going out of the track, losing grip and risking to hurt. This behaviour has been already emerged when we have tested with the opponents the controller evolved without them, so this test represents a validation of that consideration because in this case we have added an evolution taking into account the opponents.



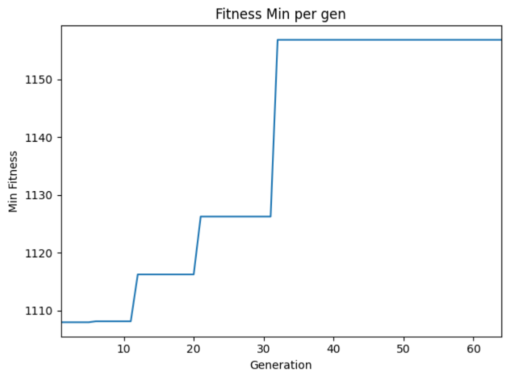


Table 12: Evolution plots for third experiment.

4.2.2 Fourth Experiment

In this experiment, we decided to start the car not at the end of grid but in the middle, in the 4_{th} position, for testing the controller in a more complicated situation in which there are cars ahead and behind it; moreover we thought that this can stimulate better it to overtake the opponents because, starting closer to the head of the race, it doesn't lose immediately contact with the first positions due to a slower start with respect to the other cars.

For this purpose, we reuse the fitness fitness of the previous experiment:

$$f = -((\mathcal{F}(P_1) + \mathcal{F}(P_2))) * 10 + (\frac{D_1}{T_1} + \frac{D_2}{T_2}) * 10 - (H_1 + H_2))$$

where:

- P: Position $[1, \ldots, 9]$ in the race with respect to other cars.
- T: Time (s) to complete the two laps.
- D: Distance (m) covered by the car from the beginning of the race.
- H: $[0, +\infty)$ Current damage of the car.
- $\mathcal{F}()$: Function of points scoring system.

Parameters	**Value**
CR	0.9
F	0.8
Pop Size	240
N. of generations	140
Individuals initialization	from *second experiment*
Tracks	Forza, Wheel-1
Seed	41

Table 13: Summary of the hyperparameters considered.

The Table 14 shows the behaviour of the fitness function for this fourth experiment, highlighting an immediate improvement up to 60_{th} generation and then a stall up to 110_{th} generation. We decided to stop to 140_{th} generation because we noticed a new situation of stall and an weaker improvement in the average.

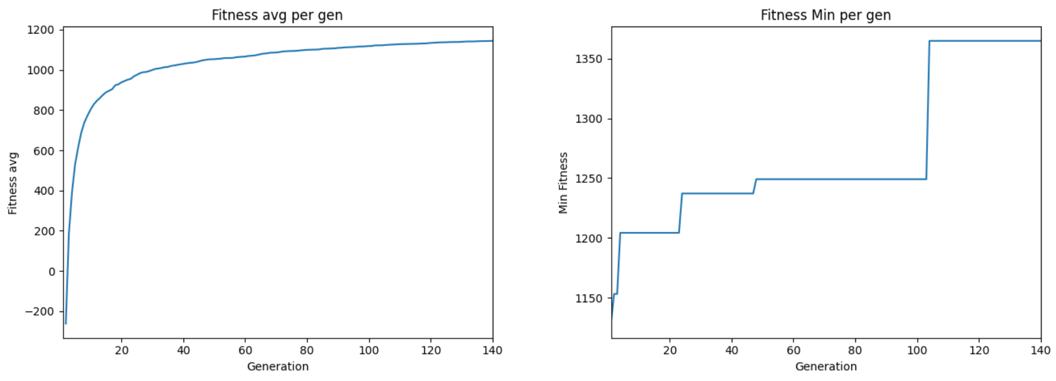


Table 14: Evolution plots for fourth experiment.

4.2.3 Fifth Experiment

The fifth experiment is a sort of complementary test of the previous one, because the training is characterized by the presence of all opponents (8), but we decided to start in last position (9_{th}). This because we have split the evolution goal in two parts: in the first one, we have mainly focused on the ability of overtaking in fact, starting in the first positions, our car could better face the opponents avoiding that the most competitive one accumulates a greater advantage immediately because we noticed that we are one of the slower car as regards the start. In the second part, we wanted to enrich the features of the controller with the ability of completing the lap in the shortest possible time, in fact, starting from the last position, it is out-marched at the starting of the race and it needs to recover time for reaching the other cars and overtaking them, otherwise it receives a lower fitness; so it is more stimulated to improve the best lap.

Moreover, the training is based only on two same circuits, *Forza* and *Wheel-1*, and uses the fitness of the previous experiment:

$$f = -((\mathcal{F}(P_1) + \mathcal{F}(P_2))) * 10 + (\frac{D_1}{T_1} + \frac{D_2}{T_2}) * 10 - (H_1 + H_2))$$

where:

- P: Position $[1, \dots, 9]$ in the race with respect to other cars.
- T: Time (s) to complete the two laps.
- D: Distance (m) covered by the car from the beginning of the race.
- H: $[0, +\infty)$ Current damage of the car.
- $\mathcal{F}()$: Function of points scoring system.

Parameters	**Value**
CR	0.9
F	0.8
Pop Size	240
N. of generations	100
Individuals initialization	from *fifth experiment*
Tracks	Forza, Wheel-1
Seed	41

Table 15: Summary of the hyperparameters considered.

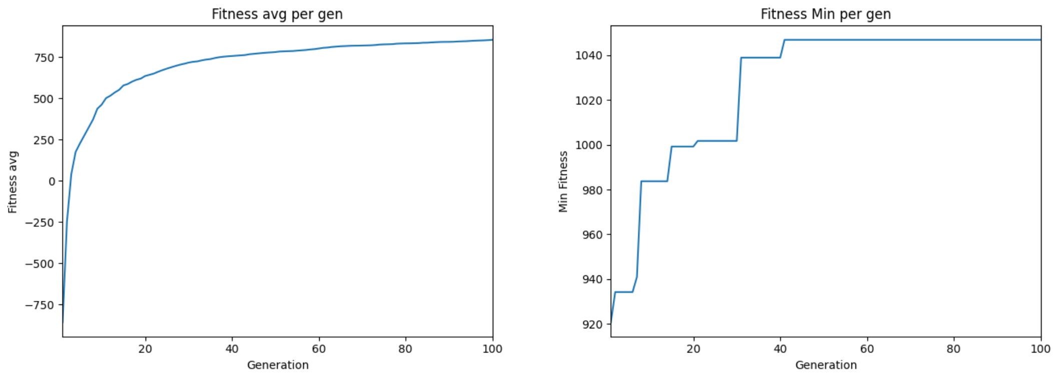


Table 16: Evolution plots for fifth experiment.

As you can see in Table 16, best fitness is increased a lot except in the last 45 generations, where there has been a sort of flattening also in the plot of the average fitness that did not allow the best fitness to improve anymore; for this we decided to stop the training and to pass to the next experiment.

4.2.4 Sixth Experiment

This was the last experiment we did with the opponents and we have tested a new fitness trying to improve the car behaviour obtained in the previous phase. We focused about the lap time instead of the average speed, because we want give more importance to design a controller that runs the circuits in the shortest possible time, according to the requirements. In fact, our idea is that probably the controller is more affected to go out of track or to cause incident if we give more importance to the average speed (as we done in previous experiment) and, accordingly, the lap time increased; on the other hand, minimizing the time required to complete the two laps, we improve also the average speed but until this improvement doesn't invalidate the shortest possible time.

So, the fitness function used was characterized by final position, lap time and the damage:

$$f = -((\mathcal{F}(P_1) + \mathcal{F}(P_2))) * 10 - (T_1 + T_2) * 10 - (H_1 + H_2))$$

where:

- P: Position $[1, \dots, 9]$ in the race with respect to other cars.
- T: Time (s) to complete the two laps.
- H: $[0, +\infty)$ Current damage of the car.
- $\mathcal{F}()$: Function of points scoring system.

We used the same *CR* and *F* values from the previous experiment and, obviously, we started the training with the parameter of the second experiments in order to make a comparison with the previous experiment. Also, we decided to train the controller on only two circuits of the requirements, as done before.

Parameters	**Value**
CR	0.9
F	0.8
Pop Size	240
N. of generations	154
Individuals initialization	from *second experiment*
Tracks	Forza, Wheel-1
Seed	41

Table 17: Summary of the hyperparameters considered.

As you can see in Table 18, evolution proceeded in a good way: the average decreased continuously, in fact the best doesn't present the same value for a lot of generations as in previous experiments; moreover, at the end we register a big improvement in the best fitness, this because population is still evolving as you can see by the average fitness. However we have to stop the evolution due to time constraints.

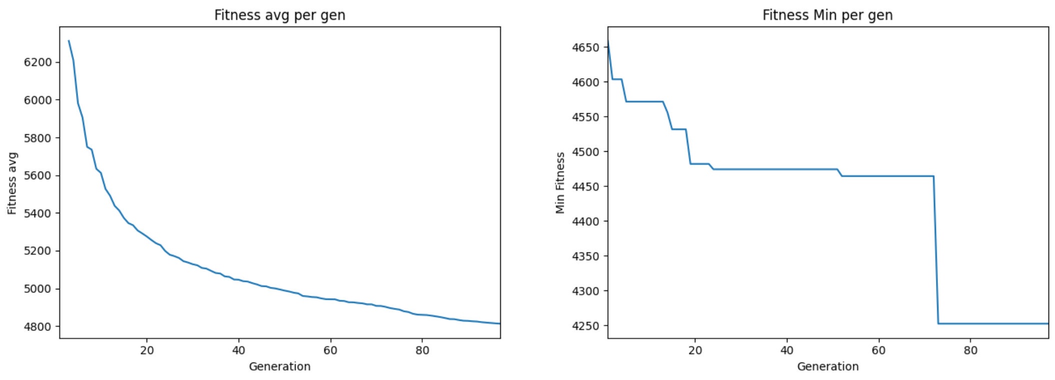


Table 18: Evolution plots for sixth experiment.

5 Results

In this section we report the analyses of the results we obtained from the experiments. In fact, these were useful to understanding which direction to follow during the development of the whole project. For a better understanding, as the previous section, we have divided the results in two subsections: without the opponents and with opponents.

5.1 Without Opponents

To have a real measure of the improvements of our model, it was necessary to compare the values obtained from the first two experiments with the default controller. So, first, we measured the timing of Snakeoil Default Controller on the four circuits of the specifications and, also, in three circuits the controller has never seen during the training: `E-Track-4`, `Wheel-2` and `E-Road`; this for better testing the ability of the controller to generalize, because it will face new track features. In particular, we chose that 3 circuits for their characteristics, almost complementary: `E-Track-4` presents long rectilinear stretches in which you can reach an high speed, `E-Road` presents tight curves in which you can test the ability of remaining inside the track and `Wheel-2` is a very difficult circuit which combines the previous features so we thought that it could represent a complete test.

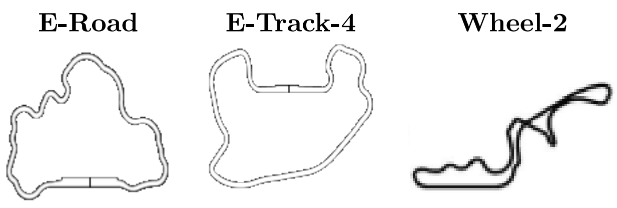


Table 19: Track Map of E-Road, E-Track-4 and Wheel-2.

With the *Snakeoil* controller using the default parameters, we obtained these results:

Track	**Tot. Time**	**Best Time**	**Top Speed**	**Damage**
Forza	240.46 s	116.61 s	276 km/h	829
Wheel-1	273.74 s	134.72 s	226 km/h	2068
E-Track-3	265.65 s	128.91 s	227 km/h	1778
CG-Track-2	131.07 s	63.02 s	250 km/h	0
E-Track-4	253.55 s	124.36 s	277 km/h	979
Wheel-2	264.35 s	129.96 s	270 km/h	26
E-Road	198.46 s	91.34 s	227 km/h	760

Table 20: Results of Snakeoil Default Controller.

Instead, in the next table we report the results obtained with the trained parameters at the end of the second experiment:

Track	**Tot. Time**	**Best Time**	**Top Speed**	**Damage**
Forza	213.51 s	103.80 s	265 km/h	0
Wheel-1	217.64 s	106.78 s	218 km/h	389
E-Track-3	208.65 s	100.44 s	240 km/h	108
CG-Track-2	139.33 s	67.32 s	230 km/h	0
E-Track-4	253.68 s	123.98 s	267 km/h	0
Wheel-2	272.23 s	132.81 s	260 km/h	8
E-Road	166.00 s	80.31 s	234 km/h	0

Table 21: Results with the parameters of second experiment.

In general on the selected circuits, the evolved controller has almost everywhere higher performance: the total time, used by the car to complete the two laps, is significantly reduced, especially on the four circuits of the training. We want to highlight that the reduction of the top speed is a consequence of the reduction of out of tracks, typical of the default controller; we have tried to balance these two factors through the scaling, as explained in Section 4.1 and we find out that this is the best compromise. Only on the track `CG-Track-2` we didn't have any improvements, probably because the default controller was already on a good level on this track that is the simpler one and, during the evolution, the controller has learned a generalized behavior that was good on all circuits.
A characteristic to evaluate is the road holding, measured through the *damage* sensor. Remember that the design specifications required to consider the damage suffered by the car, in addition to final position and lap time; the results show us the good improvements in this field on all the tracks the controller raced, and in particular on the tracks that it has never seen.

An important characteristic about which it is interesting to compare the two controllers is the *speed profile*, illustrated in Table 22, that represents the value of the speed along the X-axis (the one of the movement) tick by tick, measure provided by the *speedX* sensor. In this plot, the orange line describes the speed profile of Snakeoil Default Controller, instead the blue line describes the speed profile of car controlled with the parameters of the second experiment; the vertical lines delimit the two laps.

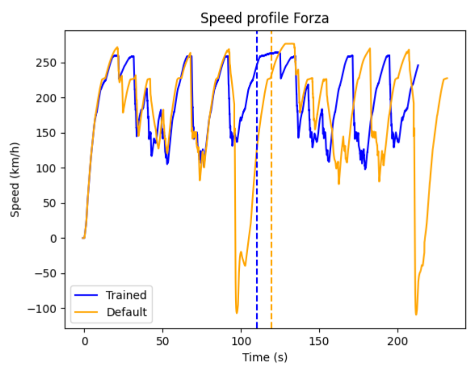
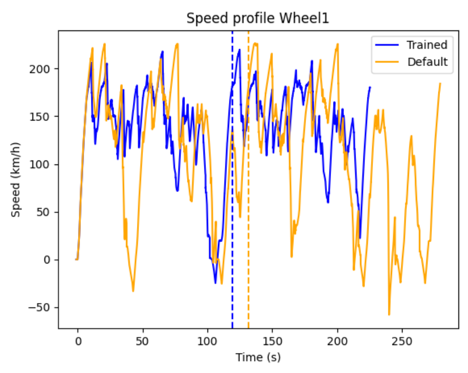
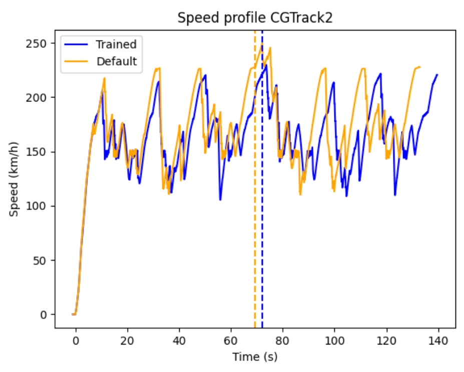
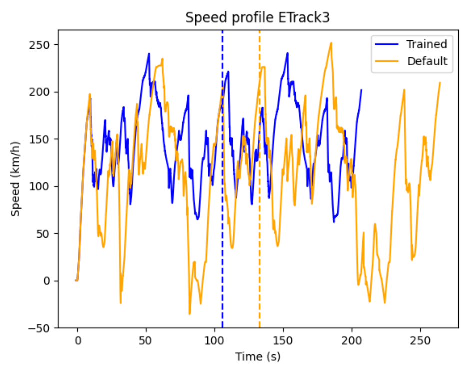
	
	

Table 22: Speed Profiles obtained after second experiment

In general, the results show us a clear deviation of the speed trend: the blue line, especially in *Forza*, has fewer peaks and limited in a smaller range on the ordinates (*speed*). This means that the car has a smooth behaviour when entering a turn which allows it to complete the circuit with an important advantage because it reduces the probability of going outside the tracks, and, also that the average speed is higher than the default controller. The negative values of speed probably means a crash during the race and the controller goes into reverse gear; we can notice that this is reduced by the evolution with respect to the default parameters.

5.2 With Opponents

To better understand the starting point of the problem with the opponents, firstly, we measured the performances of Snakeoil Controller with the default parameters in the same circuits presented in Section 5.1, but with the presence of eight opponents. In each run, the car started in the last position:

Track	**Position**	**Tot. Time**	**Best Time**	**Top Speed**	**Damage**
Forza	9_{th}	263.25 s	124.90 s	267 km/h	1668
Wheel-1	9_{th}	276.80 s	117.86 s	226 km/h	2638
E-Track-3	9_{th}	293.09 s	129.93 s	227 km/h	1123
CG-Track-2	6_{th}	148.57 s	68.79 s	220 km/h	0
E-Track-4	7_{th}	298.43 s	148.37 s	274 km/h	2700
Wheel-2	7_{th}	305.16 s	142.57 s	231 km/h	2268
E-Road	6_{th}	183.31 s	86.30 s	227 km/h	344

Table 23: Results of Snakeoil Default Controller with opponents.

We can notice that the final position, especially in the unknown tracks, is not so bad, considering that these parameters has been obtained without taking into account the opponents; however this controller causes a lot of damage, except for the run on `CG-Track-2` that anyway is a simple circuit.

The next step was to measure the performances of the controller evolved in the second experiment in presence of opponents:

Track	**Position**	**Tot. Time**	**Best Time**	**Top Speed**	**Damage**
Forza	7_{th}	240.25 s	108.69 s	265 km/h	9
Wheel-1	9_{th}	250.18 s	124.68 s	218 km/h	2908
E-Track-3	6_{th}	235.17 s	111.96 s	221 km/h	7
CG-Track-2	6_{th}	148.60 s	68.49 s	220 km/h	16
E-Track-4	4_{th}	280.81 s	132.64 s	259 km/h	2371
Wheel-2	8_{th}	315.20 s	151.44 s	257 km/h	1934
E-Road	6_{th}	183.51 s	85.95 s	230 km/h	718

Table 24: Results of the second experiment parameters with opponents.

Despite the controller was not trained with opponents, these results show us its good starting level, collecting some important position, especially in circuits that it has never seen. Instead, there is a worrying aspect regarding the *damage*, in fact in some circuits the value is very high, even if it's lower with respect to the one of default controller. With some graphic simulations, we saw that this depends also by the presence of the other cars that can hurt our one even if it is proceeding for its trajectory because it is "obstacle-free". For these reasons, this aspect was the first one that we considered for making the fitness function with the opponents, as explained in 4.2, even if with the awareness that we can't limit it at all because it also depends by other cars' behaviour.

Finally, the total time and the best lap time are affected by the presence of opponents, having a few seconds of delay with respect to the ones obtained without the opponents and presented in 5.1; however, they are almost everywhere better the ones obtained by the default controller, even if it is evolved without opponents, as the default.

In the next table we want to show you the results of the fifth experiment. This experiment is a sort of continuation of the fourth: at first, the controller training was characterized by starting of the car in the 4_{th} position, and after we decided to start the car in last position. We expected improvements especially for the *damage* because in the fitness function we considered the minimization of the value of the *damage* sensor, and from a certain point of view we did it. But we sacrificed some position, especially in the last three tracks of the table, and several seconds for the lap time. Here the results:

Track	**Position**	**Tot. Time**	**Best Time**	**Top Speed**	**Damage**
Forza	8_{th}	249.44 s	117.22 s	272 km/h	0
Wheel-1	6_{th}	234.43 s	108.62 s	236 km/h	236
E-Track-3	7_{th}	239.49 s	102.10 s	245 km/h	436
CG-Track-2	6_{th}	148.65 s	68.50 s	227 km/h	0
E-Track-4	7_{th}	297.10 s	145.78 s	280 km/h	3456
Wheel-2	9_{th}	322.63 s	158.31 s	258 km/h	1379
E-Road	7_{th}	185.28 s	87.54 s	237 km/h	224

Table 25: Results of the fifth experiment parameters with opponents.

Launching the simulation in graphic mode, we noticed again that most of the damage is due to collisions with other cars, in particular the opponents go on to our car, while following its trajectory, that is a behavior we cannot predict and limit a lot with the evolution, which is probably influenced by it. Moreover, sometimes, when the car returns to the track after an accident, it is hit by the other cars that are approaching. These two factors contribute greatly to the increase in damage during a race.

Then we report the results of the last experiment that was characterized by new fitness function, in which we considered directly the lap time instead the average speed for the reasons explained in Section 4.2.4. We expected an improvement on this aspect, but, as you can see in Table 26, we did it only on the track chosen by us and on the circuit of *Forza.* In general, we can consider the damage as the same of Table 25, due to the same damage factor in fitness function.

Track	**Position**	**Tot. Time**	**Best Time**	**Top Speed**	**Damage**
Forza	7_{th}	242.20 s	109.57 s	268 km/h	6
Wheel-1	9_{th}	248.59 s	113.99 s	227 km/h	435
E-Track-3	9_{th}	254.62 s	117.57 s	227 km/h	1688
CG-Track-2	6_{th}	149.73 s	69.34 s	225 km/h	0
E-Track-4	6_{th}	286.24 s	133.62 s	262 km/h	1294
Wheel-2	8_{th}	315.00 s	144.03 s	237 km/h	3778
E-Road	7_{th}	183.27 s	84.75 s	234 km/h	368

Table 26: Results of the sixth experiment parameters with opponents.

The next two table show the percentage of improvement of the total time, best time and damage of the controller obtained with parameters of the last two experiments. A negative value means an improvement compared to a Snakeoil Default Controller, while a positive value means a deterioration of the performance.

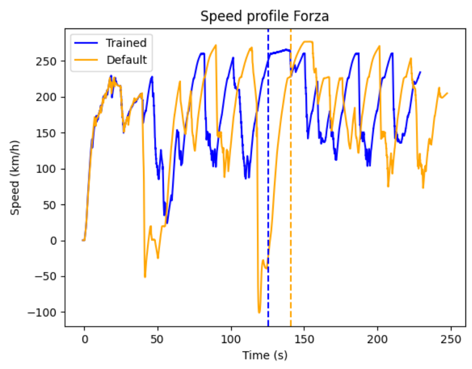
Track	**Tot. Time**	**Best Time**	**Damage**
Forza	-5.24 %	-6.15 %	-100 %
Wheel-1	-15.30 %	-7.84 %	-91.0 %
E-Track-3	-18.28 %	-21.42 %	-62.07 %
CG-Track-2	$+0.05$ %	-0.42 %	0 %
E-Track-4	-0.44 %	-1.75 %	$+28.0$ %
Wheel-2	$+5.72$ %	$+11.04$ %	-39.20 %
E-Road	$+1.07$ %	$+1.44$ %	-34.88%

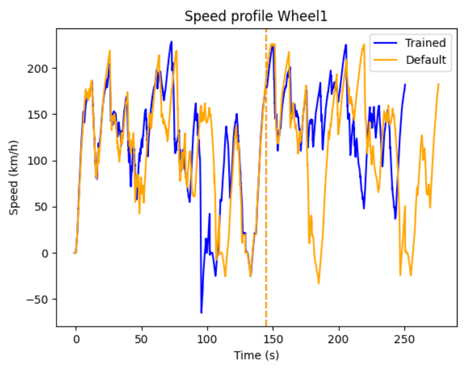
Table 27: Comparison between Default Controller and Controller of fifth experiment.

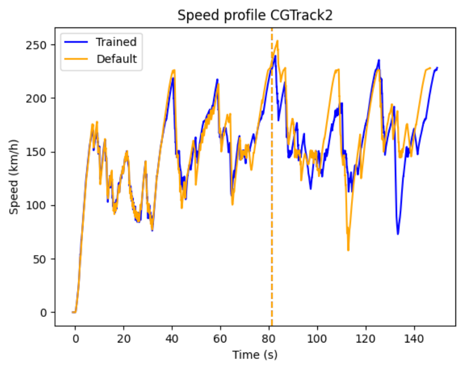
Track	**Tot. Time**	**Best Time**	**Damage**
Forza	-8.0 %	-12.27 %	-99.64 %
Wheel-1	-10.19 %	-3.28 %	-83.51 %
E-Track-3	-13.13 %	-9.51 %	$+50.31$ %
CG-Track-2	$+0.78$ %	$+0.80$ %	0 %
E-Track-4	-4.08 %	-9.94 %	-52.07 %
Wheel-2	$+3.22$ %	$+1.02$ %	$+66.57$ %
E-Road	-0.02 %	-1.80 %	$+6.97$%

Table 28: Comparison between Default Controller and Controller of sixth experiment.

Finally we report the speed profiles of the controller evolved in the sixth experiment compared to the default one on the 4 circuits of the requirements:







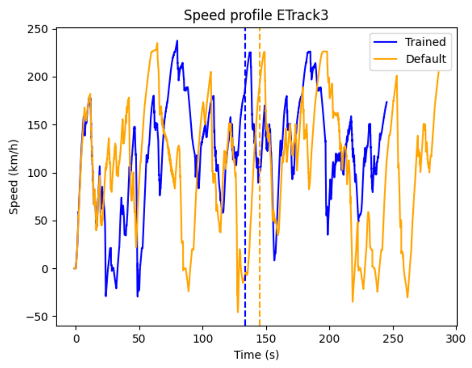


Table 29: Speed Profiles obtained after sixth experiment.

We can notice that the evolved controller has a shorter lap time, except on `CG-Track-2` where lap times are almost the same; this advantage is mostly obtained in the second lap because, as you can see, the vertical dashed lines in `Wheel1` are in the same point, so the first lap is equal, while in `E-Track-3` have a difference less than the final ones. Moreover, on `E-Track-3`, we have reduced a lot the important crashes, as you can see by the less number of peak towards a negative value of the speed.

6 Conclusions

Considering all the work we made in the project, we were able to obtain a good controller, capable of improving the original positions of the Snakeoil Default Controller. We used the Differential Evolution algorithm to change the value of the parameters and to train the controller. Our idea was to divide the goal of the project in two steps:

1. Improve the *driving skill* of the car, racing alone, with a good balance on every circuit, increasing the maximum speed and decreasing the lap time.

2. Learn the *overtaking skill* for the opponents and to limit of the damage of the car in any contacts.

We have carried out several experiments, each guided by the results obtained from the previous ones. We performed (as racing video game mode) some laps on the assigned circuits and this helped us to understand how the Torcs Simulator models physics and was useful to refine the different fitness functions.
The results were also graphically tested in TORCS, where visually we saw the clear improvement of the controller step by step: some of the generated controllers were superior as they were able to complete two laps of the race with extremely high speed without bumping to any wall or obstacle during the races. Another example of improvement was that along the track there is a darker strip, which represents the optimal trajectory, that is the one that guarantees maximum speed and road holding. During evolution, it was observed that the controllers tended increasingly to converge in many areas to the optimal path, and this was interpreted as a symptom of promising results.

Finally, the results suggest that either the *driving skill* or the *overtaking skill* alone does not lead to a very competitive and reliable car controller. Instead, the controllers that exploits both the skills is able to outperform all the other controllers.

6.1 Future Development

In general, we have seen how, sometimes, increasing the number of generations has not led to significant improvements; therefore, we could think of conducting new experiments by varying the hyper-parameters of the algorithm, that we could not do in this work due to time constraints, relying on the fact that DE in average returns quite similar results with different initialization. Also, a more complex fitness function will be considered in the future in order to generate better controllers that can be used in different tracks and that may be more inclined to overtake opponents, eventually modifying also the provided controller not only its parameters. Finally, it would be interesting to try other algorithms to compare the results and, eventually, further improve the ability to drive and overtake.

References

[1] Edwards C., (2013) *SnakeOil*, URL: `http://xed.ch/p/snakeoil/`

[2] Wymann B. et al. TORCS, (1997) *The Open Racing Car Simulator*, URL: `http://torcs.sourceforge.net/index.php`

[3] Loiacono D., Togelius J., Lanzi P. L., Kinnaird-Heether L., Lucas S. M., Simmerson M., Perez D., Reynolds R. G., Saez Y., (2008) *The WCCI 2008 Simulated Car Racing Competition*, URL: `https://www.researchgate.net/publication/224491330_The_WCCI_2008_simulated_car_racing_competition`

[4] Loiacono D., Cardamone L., Lanzi P. L., (2013) *Simulated Car Racing Championship Competition Software Manual*, URL: `https://arxiv.org/abs/1304.1672`