System for autonomous control of a racing car

(System autonomicznego kierowania samochodem wyścigowym)

Cezary Siwek

Praca inżynierska

Promotor: dr Jakub Kowalski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

31 sierpnia 2017

Abstract

Autonomous car racing is a task that requires solving problems from many fields. In this study, the author presents his system for a car control in the simulated racing environment is presented. The system's performance is compared to human players and other existing controllers. This thesis also describes contributions to various subjects including path planning, control theory, and evolutionary algorithms.

Autonomiczne kierowanie samochodem wyścigowym jest zadaniem, które wymaga rozwiązania problemów z wielu dziedzin. W pracy został przedstawiony autorski system sterujący samochodami w symulowanym środowisku, porównano jego wyniki z ludźmi i innymi algorytmami oraz omówiono wybrane zagadnienia z takich dziedzin jak planowanie ruchu, teoria sterowania i algorytmy ewolucyjne.

Contents

Chapter 1

Introduction

1.1 The real world

As industrial automation progresses, autonomous control methods for wheeled vehicles are being researched in terms of both driving and operations. However, this research usually focuses on the reliability, safety and efficiency[7]. Vehicles operate below their specified performance leaving a significant margin for unexpected situations.

Steering a racing car on the other hand, involves operating at the edge of vehicle's capabilities and opens entirely new challenges i.e. using the slip as a mean of overcoming a specific bend, while industrial vehicles must avoid such hazardous situations at all costs. As the safety requirements are lower, it is possible to use approximative methods which behaviors cannot be predicted or proven safe, but which offer a better driving performance in the mean case. Autonomous drivers are also interesting due to the possibility of objective algorithm evaluation against an other algorithm or a human.

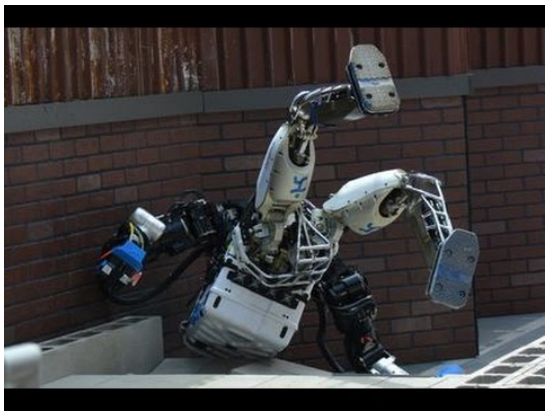


Figure 1.1: Reliability is one of the top concerns in industrial automation.
https://www.youtube.com/watch?v=ww57ZtE53Ic

1.2 The HingyBot

This study presents an autonomously driving program (named HingyBot), which was integrated with two virtual environments attempting to faithfully represent behavior of a physical racing car. The first environment is an open source game TORCS and the second is a modern game engine Unreal Engine 4.

In the past, there were held competitions between autonomous drivers. The most popular competition was the Simulated Car Racing Championship (SCRC) organized at many artificial intelligence-related conferences: *GECCO*, *CIG*, *CEC*. HingyBot is therefore written to follow this competition formula, which gives the performance of other drivers for comparison and makes it easy to compete if such competition will be organized in future.

1.3 TORCS

TORCS (the open source car simulator[4]) features a realistic car simulation model that takes into account such details as individual wheel traction, aerodynamics, and fuel consumption. It also simulates the engine of the car and provides a simple damage model.
While it is still mostly a game, its above-mentioned characteristics and existing infrastructure for bots and tracks construction makes it popular in the scientific research community. It is especially useful studying problems involving evolutionary algorithms, path optimization or autonomous vehicle movement.

Figure 1.2: Example of the TORCS visuals.

1.4 Simulated Car Racing Championship

1.4.1 Rules

The SCRC is a racing competition for autonomous drivers that uses TORCS game as a simulation environment[3]. Its rules are as follows:

- Car type used in the competition is the same for all participating drivers and known beforehand.
- All data the autonomous driver has available is coming from the car's sensors which describe local environment, especially the geometry of the track is not given.
- Before autonomous driver's performance is measured, five warm-up laps with a time limit are given. During this time, the bot can freely drive and store any data for usage in the actual race,
- During the warm-up section and the racing section sensors of the car are artificially noised in order to simulate the real-world application.

1.4.2 Technical details

Organizers have published a modified TORCS version that hosts a game server. They also provide a bot template that connects to the server via UDP socket and, depending on the server configuration races either alone or with other bots and players. For technical reasons HingyBot is written entirely from scratch and does not use the above-mentioned template.

1.4.3 Car input and output

Table 1.1: Sensors of the car (as presented in [3])

Name	**Range (unit)**	**Description**
angle	$[-\pi, +\pi]$ (rad)	Time elapsed during current lap.
curLapTime	$[0, +\infty)$ (m)	Angle between the car direction and the direction of the track axis.
damage	$[0, +\infty)$ (point)	Current damage of the car (the higher is the value the larger is the damage).
forwardpostion	$[0, +\infty)$ (m)	Distance from the start line along the track line.
distraced	$[0, +\infty)$ (m)	Distance covered by the car from the beginning of the race
focus	[0, 200) (m)	Vector of 5 range finder sensors: each sensor returns the distance between the track edge and the car within a range of 200 meters. When noisy option is enabled (see Section 7) sensors are affected by i.i.d. normal noises with a standard deviation equal to the 1% of the sensors range. The sensors sample, with a resolution of one degree, a five degree space along a specific direction provided by the client (the direction is defined with the focus command and must be in the range [-90,+90] degrees w.r.t. the car axis). Focus sensors are not always available: they can be used only once per second of simulated time. When the car is outside of the track (i.e., *crossposition* is less than -1 or greater than 1), the focus direction is outside the allowed range ([-90,+90] degrees) or the sensors has been already used once in the last second, the returned values are not reliable (typically -1 is returned).
fuel	$[0, +\infty)$ (l)	Current fuel level.
gear	{-1, 0, 1, ..., 6}	Current gear: -1 is reverse, 0 is neutral and values from 1 to 6 are the normal gears.
lastLapTime	$[0, +\infty)$ (s)	Time to complete the last lap.
opponents	[0, 200] (m)	Vector of 36 opponent sensors: each sensor covers a span of 10 degrees within a range of 200 meters and returns the distance of the closest opponent in the covered area. When noisy option is enabled (see Section 7), sensors are affected by i.i.d. normal noises with a standard deviation equal to the 2around the car, spanning clockwise from -180 degrees up to +180 degrees with respect to the car axis.
racePos	{1, 2, 3, ..., N}	Position in the race with respect to the other cars.
rpm	$[0, +\infty)$ (Hz)	Number of rotation per minute of the car engine.

Name	Range (unit)	Description
speedX	$(-\infty, +\infty)$ (km/h)	Speed of the car along the longitudinal axis of the car.
speedY	$(-\infty, +\infty)$ (km/h)	Speed of the car along the transverse axis of the car.
speedZ	$(-\infty, +\infty)$ (km/h)	Speed of the car along the Z axis of the car.
track	[0,200] (m)	Vector of 19 range finder sensors: each sensors returns the distance between the track edge and the car within a range of 200 meters. When noisy option is enabled (see Section 7), sensors are affected by i.i.d. normal noises with a standard deviation equal to the 10% of sensors range. By default, the sensors sample the space in front of the car every 10 degrees, spanning clockwise from -90 degrees up to +90 degrees with respect to the car axis. However, the configuration of the range finder sensors (i.e., the angle w.r.t. the car axis) can be set by the client once during initialization, i.e., before the beginning of each race.
crossposition	[-1,1]	Distance between the car and the track axis. The value is normalized w.r.t. the track width: it is 0 when car is on the axis, -1 when the car is on the right edge of the track and +1 when it is on the left edge of the car. Values greater than 1 or smaller than -1 mean that the car is outside of the track.
wheelSpinVel	$[0, +\infty)$ (rad/s)	Vector of 4 sensors representing the rotation speed of Wheels.
z	$(-\infty, +\infty)$ (m)	Position along the Z axis.

Table 1.2: Car steering (as presented in [3])

Name	Range (unit)	Description
accel	$(-\infty, +\infty)$ (km/h)	Virtual gas pedal (0 means no gas, 1 full gas).
brake	$(-\infty, +\infty)$ (km/h)	Virtual brake pedal (0 means no brake, 1 full brake).
clutch	$(-\infty, +\infty)$ (km/h)	Virtual clutch pedal (0 means no clutch, 1 full clutch).
gear	{-1, 0, 1, ..., 6}	Current gear.
steering	[-1,1]	Steering value: -1 and +1 means respectively full right and left, that corresponds to the angle of 0.366519 rad.
focus	$[0,+\infty)$ (rad/s)	Focus direction (see the focus sensors in Table 1.1).
meta	$(-\infty, +\infty)$ (m)	This is a meta-control command: 0 do nothing, 1 ask competition server to restart the race.

1.4.4 Existing approaches and studies

- **Racing games** usually employ simplistic, but reliable methods.
  It is common that the optimal racing line is drawn by a human domain expert and later corrected by algorithms specially designed for each particular case. Even when some complex methods are employed, like in the first racing game to feature artificial neural networks: Colin McRae Rally 2.0, the racing line is still hardcoded into a game and the ANN is used only to keep the car on it[5, 9].

- **Genetic algorithm** was used by Cardamone et al. for racing line computation for TORCS[8]. When it was published in 2010 it was considered the best artificial driver. However, this solution requires full and exact knowledge of the track's geometry.

- A bot using the **Monte Carlo Tree Search** was implemented with a good result[6]. The method uses a regression analysis-based forward model, complex evaluation function and records track during the before-mentioned warm-up time. Identically to the HingyBot, it represents geometry of the track with respect to both the car's sensor space and absolute Euclidean coordinate system. It allows to plan the path with absolute coordinates and then do low-level path following with respect to the car's sensor.

- **Gene Regulatory Networks** are used in the winner of SCRC 2015, the *GRNDriver*[1]. This evolutionary algorithm was inspired by processes responsible for the control functions of the biological cells.
  During most of the race GRNs are the only mechanism controlling the GRN-Driver.

- **Various online neuroevolution strategies** were tested by before-mentioned Cardamone et al[10]. In one particular experiment, the authors evolved a separate artificial neural network champion for each track. Then those champions were used to seed the evolution of drivers for other tracks improving both evolution speed and final performance of the best individual.

Chapter 2

Evolutionary approach

2.1 Infrastructure

The autonomous driver presented in this study was developed along with various tools. The first one is an automated tester that concurrently runs the race instances defined in an *.xml* file and outputs performance of the bot. The second one is an evolutionary trainer that uses the tester to train evolutionary algorithms that implement a simple interface.

```
cezar@stacjonarny:/home/cezar/hingy_package/TORCSTester/TORCSTester
cezar@stacjonarny ~/hingy_package/TORCSTester/TORCSTester (devel) λ \
> bin/Debug/TORCSTester.exe cases/view_best.xml 4
Case Street has started.
Case Speedway has started.
Case Alpine has started.
Case Emero has started.
Case Speedway has ended.
Case Mikegrady has started.
Case Emero has ended.
Case Noceda has started.
Case Street has ended.
Case Senhor has started.
Case Mikegrady has ended.
Case Alpine has ended.
Case Noceda has ended.
Case Senhor has ended.
-------------------------
Case Speedway: 422.86 ref 7.859985 (1.893972%)
Case Emero: 647.45 ref -29.54999 (-4.364843%)
Case Street: 850.44 ref -71.56 (-7.761388%)
Case Mikegrady: 750.97 ref -0.0300293 (-0.003998575%)
Case Alpine: 1356.23 ref -222.77 (-14.1083%)
Case Noceda: 596.63 ref 17.63 (3.044906%)
Case Senhor: 800.14 ref 2.140015 (0.2681723%)
Total score: 5424.72
Mean score: 774.96
Mean ref: -3.00449658251767
-------------------------
```

Figure 2.1: Interface with the testing tool.

As configuration of the bot, tester, and the trainer is entirely done through the *.xml* files and runtime arguments, another tool is provided for configuration validation and automation.

All the tools are designed so they can be used with other bots, other evolutionary algorithms or for different goals, like procedural content generation.

2.2 Parameters searching

HingyBot uses many constant parameters for almost any function (>30 in total). Some of those parameters are different for each car type, bot configuration etc.
To find the optimal values for those parameters and test the evolutionary system a simple genetic algorithm was implemented. It stores all parameters in an array which is randomly mutated and uniformly crossed over with other individuals.
This method found much better parameters than two humans could and was even able to find the optimal evolutionary parameters for itself.

2.3 Gene Regulatory Networks

2.3.1 In biology

GRNs are biological systems that regulate metabolism of the living cells[11]. Based on concentrations of certain regulatory proteins they change transcription rate of genes, which further effects concentrations of those proteins.
Cellular organelles are sensitive to change of proteins concentrations, which affects their behavior and guides them through the life cycle of the cell.

2.3.2 In evolutionary computation

HingyBot uses a biology-inspired model of GRNs for regulatory purposes. During the simulation, while controlling an agent, the model can be imagined as a weighted and directed graph consisting:

- n abstract proteins as vertices of the graph.
- n numbers from 0 to 1, each one assigned to other protein. Their values describe state of the network in certain point of simulation. They are analog to the concentrations of biological proteins.
- regulatory edges, which describe how concentration of each protein effects concentration of other proteins through time.

Proteins are divided into three types:

- Input proteins, which are used for feeding the GRN data from the external world. For example, when using the GRN for speed control we provide speed information by artificially changing concentration of the certain protein to a value proportional to the current speed.
- Output proteins, which provide output of the network. They come in pairs (c_1 and c_2) for each output value. In our example, we want the GRN to output desired gas pedal pressure, so we calculate it with measuring concentrations of two output proteins assigned to the gas pedal:

$$\text{gas} = \frac{c_1 - c_2}{c_1 + c_2}.$$

- Regulatory proteins, which regulate and are regulated by other proteins.

Numbers or input and output proteins are determined by the system that the GRN is supposed to steer. Regulatory edges and the number of regulatory proteins describe an individual GRN and are subjected to evolution. There are two kinds of regulatory edges:

- enhancing edges, which describe how protein enhances concentration of other proteins,
- inhibiting edges, which describe how protein decreases concentration of other proteins.

At beginning of the simulation all concentrations are set to a specific value. Then, in each simulation step each protein's p concentration is changed according to the following formula:

$$c'_p = c_p + \frac{d}{n} \sum_{i}^{n} c_i [E(i,p) - I(i,p)]$$

where:

c'_p is new p concentration,

c_p is old p concentration,

d is parameter describing dynamics of the regulation,

n is number of proteins,

$E(i,p)$ is weight of the enhancing edge from i to p,

$I(i,p)$ is weight of the inhibiting edge from i to p.

Because concentrations are not reseted between simulation steps, the output of the GRN is determined not only by the current input state but also by all previous states filtered by the regulatory behavior. This makes GRNs suitable for a wide range of applications ranging from the signal processing to the black box optimization.

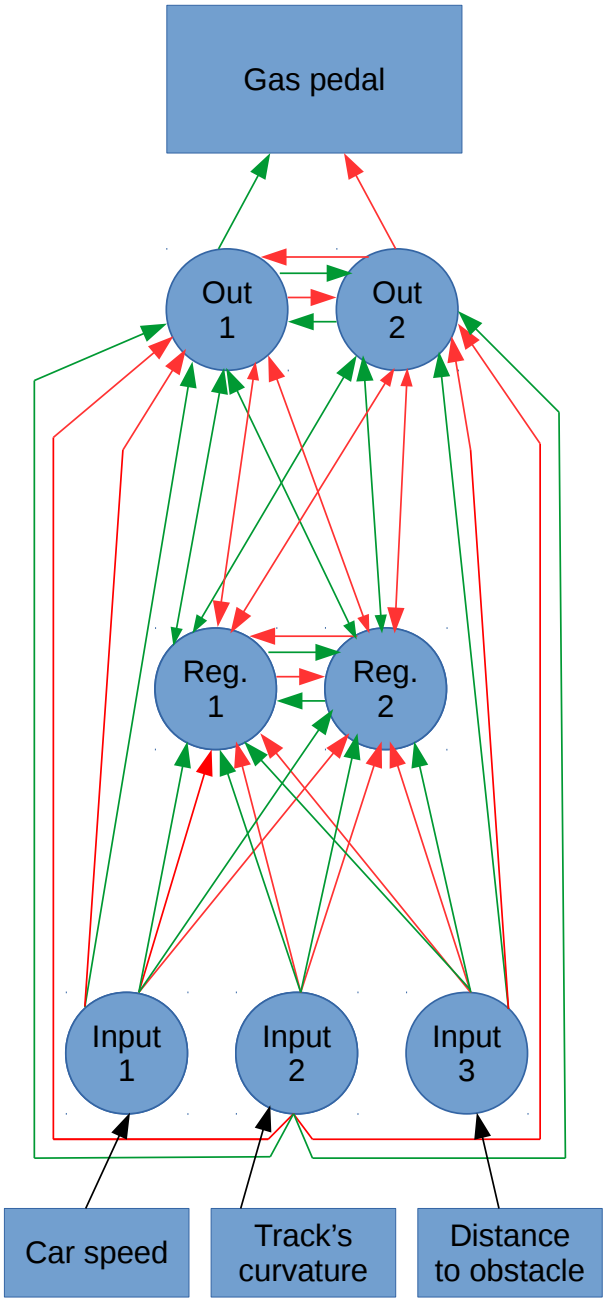


Figure 2.2: Example GRN model (not the one HingyBot uses).

Chapter 3

Track recording

It is highly beneficial to have an approximative geometry of the entire track, as it can be used to employ advanced path planning techniques. The HingyBot performs a fundamental recording during the first warm-up lap. As speed of the wheels is given precisely, we can use it for absolute direction estimation, provided we will have excellent traction during the entire recording process. We drive the whole first lap with very low speed and keep our car in the middle using the *crossposition* sensor. Using speed difference of the wheels we are able to calculate approximate absolute heading for each $forwardposition\ t$:

$$h_t = f \int_0^t l_x - r_x dx$$

where:

l_x is rotation speed of the left wheel at $forwardposition\ x$,

r_x is rotation speed of the right wheel at $forwardposition\ x$.

The f is a parameter describing proportion between wheels speed difference over constant period of time and change of heading during the same period.
Because the track makes known k full turns we find this parameter once for each car type by solving for f:

$$f \int_0^1 l_x - r_x dx = k2\pi.$$

Using absolute heading information we can calculate geometry of the whole middle lane:

$$x_t = \int_0^t \frac{l_x + r_x}{2} cos(h_x) dx,$$

$$y_t = \int_0^t \frac{l_x + r_x}{2} sin(h_x) dx.$$

To find geometry of the track's bounds we simply use our left and right distance sensors.

Although rotational speed of the wheels is given almost exact (that is realistic due to the precision of modern rotation sensors) it still produces an error which will accumulate to considerable values, especially in further parts of the track. It is therefore very beneficial to transform track's geometry into car's sensor space using $forwardposition$ and $angle$ sensors. In such case the autonomous driver will only suffer misrepresentation of the local bend, but not all the bends behind him.

Figure 3.1: Example TORCS track to be recorded.

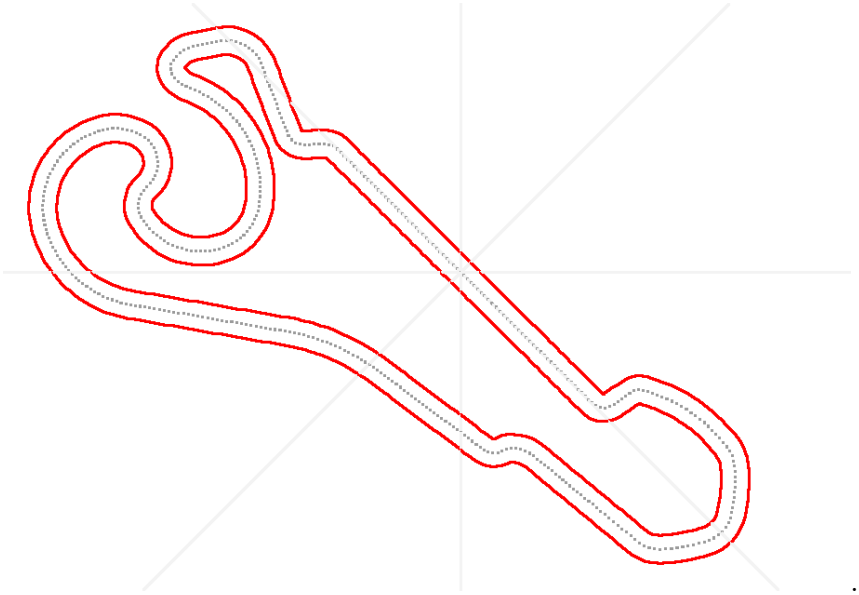


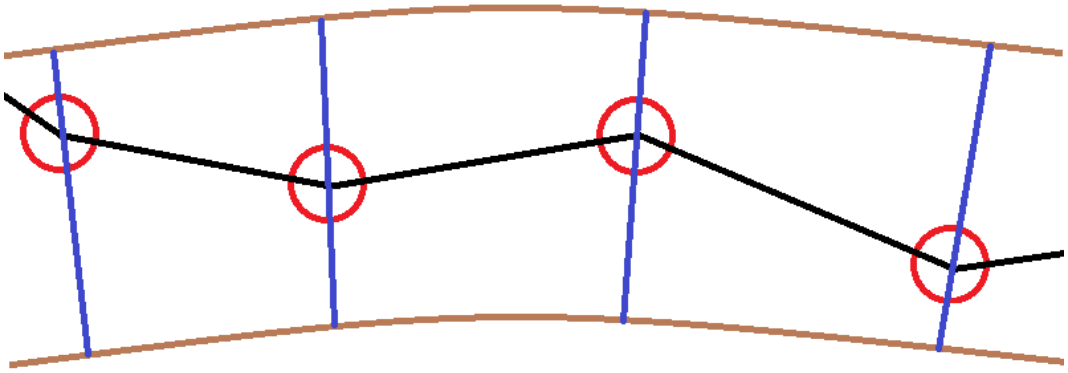
Figure 3.2: Track geometry created by the recording process.

Chapter 4

Path planning

Before the beginning of the actual race, an optimal driving line is computed. For the line to be optimal, we prefer it to be as short as possible and its bends to be as gentle as possible. Those requirements are sometimes exclusive.

In order to compute line that fulfills those requirements, we will use a simulated physical system, which will transform the racing line to more efficient one at every iteration of the simulation. We start with racing line produced by traveling the track and calculating mean position between the bounds. We define such curve as the centerline. Now, we approximate the centerline as a sequence of straight segments. We define points joining the segments as hinges (red) and lines connecting the joints as tubes (black).

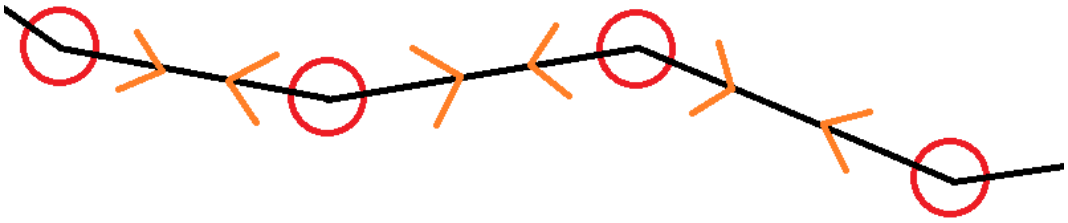


.

Hinges and tubes are the elements of our physical model. Hinges are free to move along the lines perpendicular to the centerline, but they cannot pass the track's bound. Tubes are always straight but able to stretch or shorten. Position, rotation and length of the tubes are determined by position of the hinges they are connecting. The system is two-dimensional, tubes have no mass, there are no friction forces and no inertia.

There are two forces working in our system:

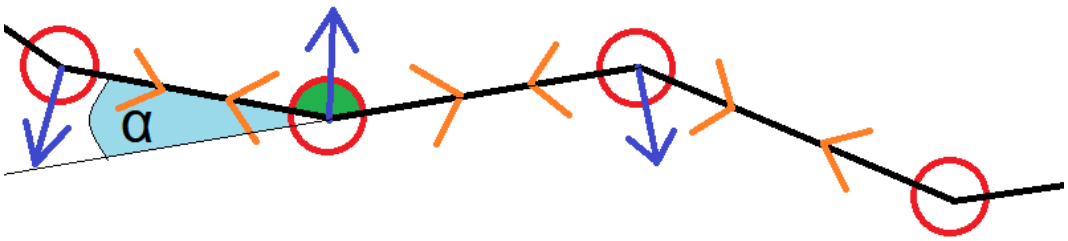
- **The pulling force** (orange), which represents our desire for the line to be as short as possible. This force is applied for each tube to its adjacent hinges. They are pulled towards each other with the force proportional to the length of a tube. This force can be intuitively imagined as if there were springs inside our telescopic tubes. Using this force alone, the resulting racing line will convergence to the convex shell of the inner bound of the track.



.

- **The straightening force** (blue), which represents our desire for the racing line's curvature to be lowest. This force is applied for each hinge for it and to its neighbors. It can be intuitively imagined as if there were springs inside the hinges which are trying to straighten the hinge. Direction of the force applied to neighboring hinges is directed perpendicular to the tube and turned to straighten the hinge. The force is proportional to the square of the angle α (the angle between the current tube and the extension of the adjacent tube).

  An opposite force is exerted on the central hinge (the one containing a spring) from the action-reaction principle.



.

At each iteration, a new hinge position computed by the following formula is snapped to the closest possible point.

$$P' = P + \frac{f_s(\vec{F_{rl}} + \vec{F_{rr}} - \vec{F_{al}} - \vec{F_{ar}}) + f_p(\vec{F_{pl}} - \vec{F_{pr}})}{m}$$

where:

P' is new hinge position,

P is old hinge position,

$\vec{F_{rl}}$ is straightening force applied by the left neighbour,

$\vec{F_{rr}}$ is straightening force applied by the right neighbour,

$\vec{F_{al}}$ is straightening force applied to the left neighbour,

$\vec{F_{ar}}$ is straightening force applied to the right neighbour,

$\vec{F_{pl}}$ is pulling force applied by the left tube,

$\vec{F_{pr}}$ is pulling force applied by the right tube,

f_s is straightening force factor parameter,

f_p is pulling force factor parameter,

m is hinge mass factor parameter.

Optimal proportion $\frac{f_s}{f_p}$ depends on the car's turning capabilities and is chosen experimentally for each car. The lower the m parameter, the fewer system iterations are required for a desired convergence. It is beneficial to find as low m as possible, but high enough for the system to remain stable. Then, as the system develops and straightening forces decrease, m can be further lowered to accelerate the convergence.

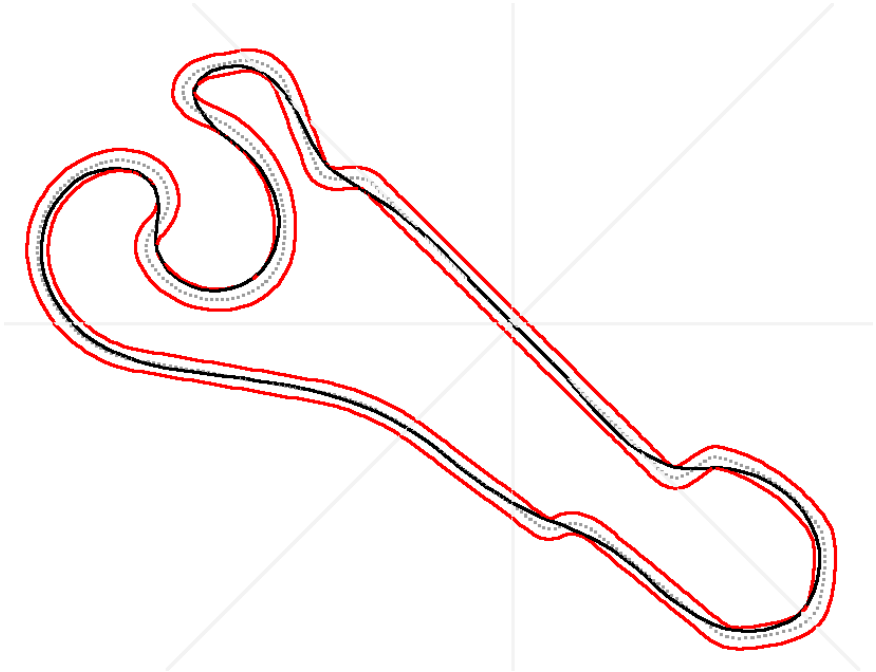
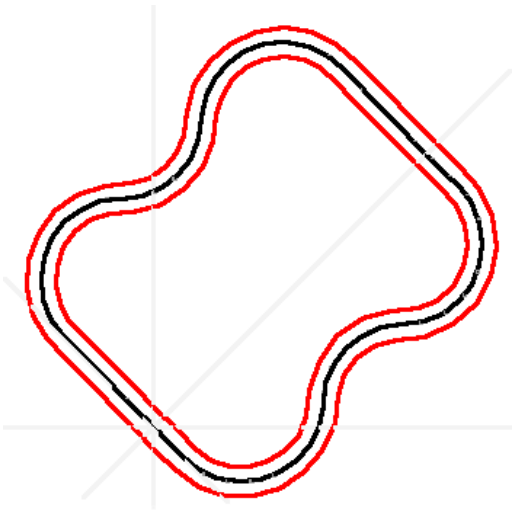
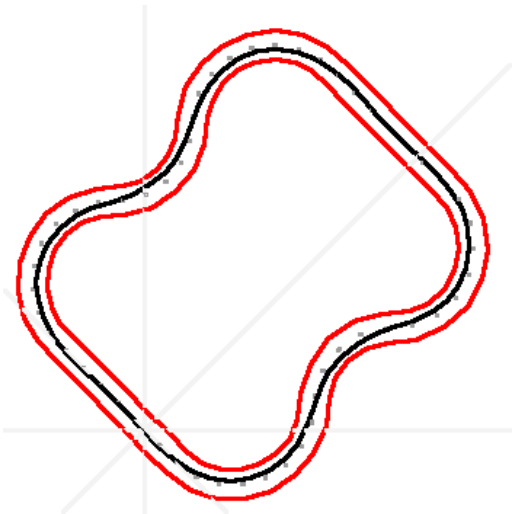


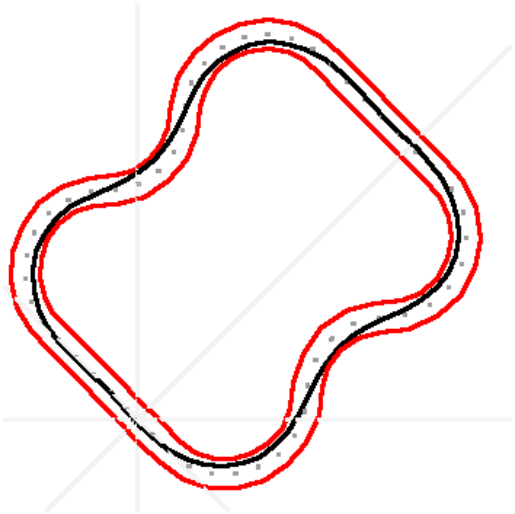
Figure 4.1: Example optimal path calculated by the Path Planning Module.



(a) Initial state



(b) after 50 iterations



(c) after 500 iterations

(d) after 5000 iterations

Figure 4.2: Track development

Chapter 5

Speed planning

5.1 Backwards-traveling agent

After the optimal driving line has been computed, desired speed for each tube is calculated according to the following formula:

$$S_t = s_b + s_f E_t$$

where:

S_t is desired speed at tube t,

s_b is speed base parameter,

s_f is speed factor parameter,

E_t is 'energy' at hinge connected to tube's t more forward end.

Energy is calculated for each hinge traveling backwards, starting from the first behind the start line ($E_n = 1$) according to the following formula:

$$E_{t-1} = E_t + l_t(\Delta - p_1|\alpha_t| - p_2|\alpha_t^3|),$$

where:

E_t	is new energy clamped to $< 0, 1 >$,
E_{t-1}	is previously calculated energy of the next hinge t,
α_t	is heading change at hinge t,
Δ	is parameter loosely related to the car's braking capabilities,
l_t	length of the tube t,
p_1, p_2	are parameters decribing how track's curvature should decrease speed,
n	index of the last hinge.

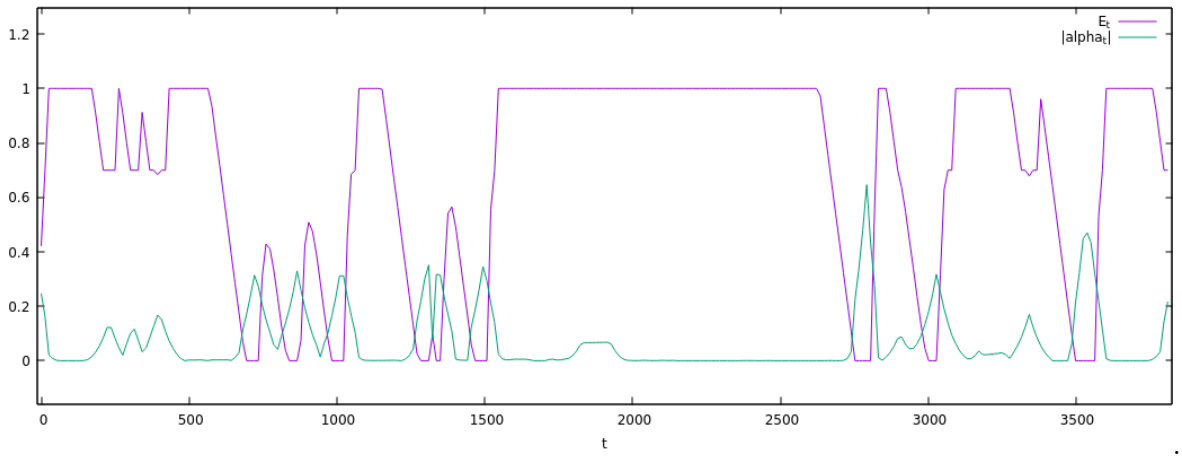


Figure 5.1: Example 'energy' output of an agent.

5.2 Gene Regulatory Network

Before-mentioned Gene Regulatory Networks can be used as another source of desired speed estimation. Each combination of the following implementations were tested:

Online vs offline

- **Online:** as the car drives the track, surrounding bounds geometry is fed to the Gene Regulatory Network in real time. Data comes from both the recorded track and car's sensors.
- **Offline:** Gene Regulatory Network is quickly run through the recorded track and its results are executed later.

Assisted vs unassisted

- **Assisted:** Gene Regulatory Network has the agent's desired speed available as one of the inputs.
- **Unassisted:** Gene Regulatory Network must calculate the speed basing solely on the track's geometry. This is useful for initial GRN learning, because it prevents a population from falling into some local maximum in which GRN repeats speed of the agent.

5.3 Dangerousness metric

Each track has usually one or two bends that are significantly more difficult than others. When we try to find an optimal speed factor, we often can either choose it

for the hardest bend or a mean one. In the first case, we will never crash, but we will lose a lot of time on almost every bend. In the second case, we will have an overall good performance, but if we crash on the hardest bend it will make matters worse than in the first case. This failure of the previously described speed planning algorithm can be neutralized by using four remaining warm-up laps. We choose speed factor optimal for a typical bend, and try to drive around the track. If we crash, we artificially lower desired speed at segments preceding the unfortunate bend to a value that will guarantee safe pass at the trial. During next laps we increase the desired speed at those segments a little bit until we crash again. Our final speed at those segments is the highest speed at which we passed the bend without crashing.

5.4 Using the speed plan to optimize the trajectory

After calculating desired speed on each segment, we can use this information to improve geometry of the track. To do so, we increase pulling to straightening force ratio in regions with low desired speed and vice versa. After that, we can use modified racing line's geometry to recalculate corrected desired speeds. We use the spare computational time for repeating this process during the whole race.

Chapter 6

Drive plan execution

6.1 Transition into sensor space

Geometry of the track is stored in absolute, Euclidean coordinate format, whereas sensor data comes in respect to the car. One way is to estimate absolute car position and perform computation in absolute coordinates. The HingyBot uses another method. It assumes that car's absolute heading corrected by the angle sensor equals to the absolute racing line's heading at current $forwardposition$ of the car. Using this information, track geometry is translated into sensors space rooted in the car, computations are performed and results are translated back into the absolute coordinate system.

6.2 Transmission

A simple automatic transmission was implemented so the regulatory part only needs to provide desired speed change from -1 to 1. It increases the gear when RPM of the engine surpasses a certain threshold, and decreases it when RPM falls below another threshold.

There is an unresolved inefficiency of the transmission, because HingyBot accelerates significantly slower than bots bundled with TORCS at very low and very high speeds.

6.3 Cascade PID controller

In order to keep the car on the optimal racing line two Proportional-Integral-Derivative (PID) controllers are used. The first one is responsible for matching car's heading with the heading of the racing line. The second one keeps car's $crossposition$ on the racing line by modifying first regulator's desired heading.

PID regulators controlling the steering wheel are dynamically tuned based on the speed information.

A separate PID controller is used for controlling the speed of the car.

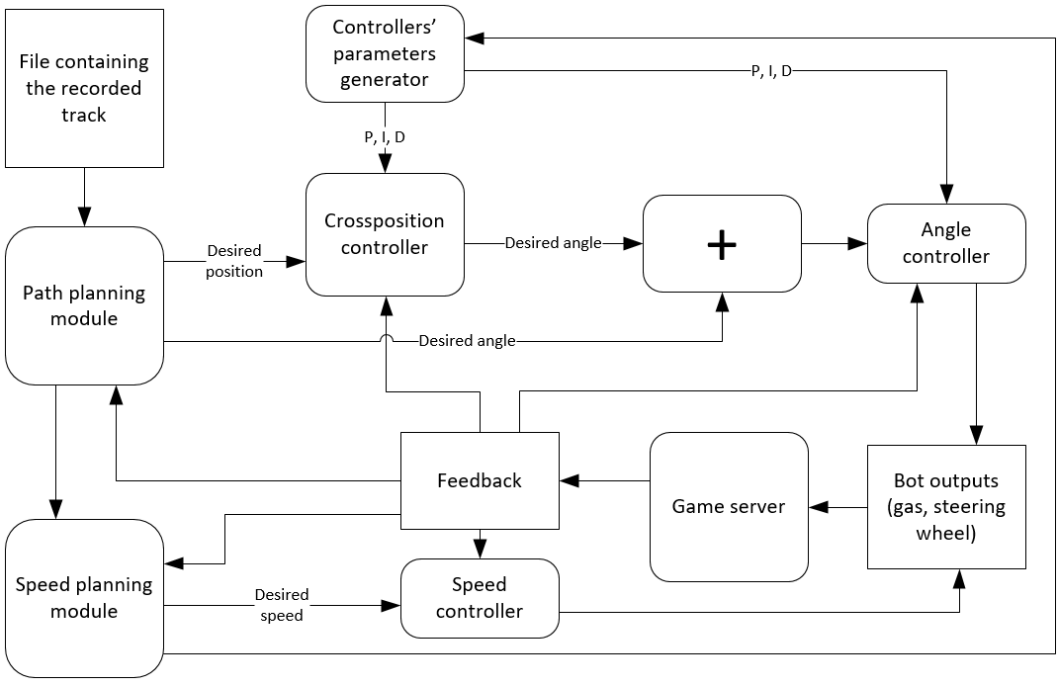


Figure 6.1: A simplified control diagram of the HingyBot.

6.4 Collision avoidance

No collision avoidance against other cars was implemented. Due to the good PID tunning, collisions in 1v1 race are even preferable.

6.5 Returning to track

In case of collision, the PID controllers are expected to handle the situation without any hard-coded behaviors. The HingyBot only executes such behaviors if the situation is unrecoverable i.e. in case of getting stuck or reversed. It can do a multiple point-turn if rotated backward or reverse out if stuck beyond the racing road.

Chapter 7

Results

Figure 7.1: HingyBot about to complete the final lap on the Alpine1 track. Note significant lead distance in comparison to other bots on the minimap.

7.1 Racing against humans

HingyBot does well when racing against human players. It was tested against two humans with experience in racing and arcade games. It wins most of the time. The best strategy against the bot is to force a collision that will lead it unrecoverable situation. Even though it is more effective than the usual way, it is dangerous and

because of finely-tuned PID regulatory the HingyBot is usually expected to handle the collision better than the human player. Not counting such collisions, it took the human players respectively twenty-one and thirty trials to beat the HingyBot on an unfamiliar track.
This is more than enough performance to use the HingyBot in a racing game, in which bots sometimes struggle to defeat a human player despite having absolute knowledge or even cheating.

7.2 Racing against the TORCS bots

HingyBot was put against all bots which come bundled with TORCS and have the SCRC car available. It was tested on three tracks and the races took three laps. HingyBot is named as the *scr_server*1 or *scr_server*3.

Race Results
Alpine 1

Rank	Driver	Total	Best	Laps	Top Spd	Damages	Pit Stops
1	scr_server 3	07:04:40	02:19:38	3	220	9636	0
2	bt 3	+38:58	02:31:23	3	237	4477	0
3	lliaw 3	+38:89	02:29:68	3	255	4994	0
4	berniw 3	+40:35	02:31:95	3	250	3592	0
5	Olethros 3	+3 Laps	00:00	0	213	10155	0
6	inferno 3	+3 Laps	00:00	0	205	10002	0

Race Results
Senhor-hill

Rank	Driver	Total	Best	Laps	Top Spd	Damages	Pit Stops
1	bt 3	04:12:97	01:21:95	3	252	1	0
2	scr_server 1	+03:07	01:22:77	3	228	8055	0
3	lliaw 3	+04:40	01:22:33	3	251	3700	0
4	berniw 3	+05:93	01:20:43	3	251	3049	0
5	Olethros 3	+22:16	01:29:19	3	226	3998	0
6	inferno 3	+24:37	01:28:79	3	242	6451	0

Race Results
Emero-city

Rank	Driver	Total	Best	Laps	Top Spd	Damages	Pit Stops
1	scr_server 3	03:20:79	01:04:95	3	252	2174	0
2	bt 3	+15:34	01:09:66	3	259	1495	0
3	inferno 3	+16:24	01:08:47	3	260	1494	0
4	berniw 3	+16:47	01:10:52	3	259	4177	0
5	lliaw 3	+16:61	01:09:73	3	255	1595	0
6	Olethros 3	+37:16	01:18:31	3	244	105	0

It is important to note that bots embedded in TORCS have absolute knowledge about the track geometry, opponents position, and they don't have any noise on this data.[4]

The HingyBot was also always placed at the last starting position, meaning it had to take over all its opponents and do not suffer a fatal collision in this process.

7.3 Racing against the SCRC2015 champion

The last SCR competition was organized in 2015. Its winner was the beforementioned GRNDriver, which relies on the Gene Regulatory Networks, evolution strategies for them, and the dangerousness metric[12].

The GRNDriver used for comparison below has all features and speed optimizations enabled.[2] Eight tracks with ten laps were used. One of the tracks had a dirt-type surface (Zlovenovice).

Track	**HingyBot time [s]**	**GRNDriver**[2] **time [s]**	**Percentage difference**
CG Speedway 1	422	**415**	1.91%
Alpine 1	**1356.3**	1579	-14.1%
Street 1	**846.4**	922	-8.2%
Noceda-city	596.8	**579**	3.06%
Emero City	**647.5**	677	-4.35%
Mikegardy-hill	750.98	751	0%
Senhor-hill	800.5	**798**	0.3%
Zlovenovice	**733.2**	772	-5.02%

Table 7.1: Peformance comparsion between the HingyBot and the GRNDriver.

HingyBot in its current state cannot handle jump ramps or certain tracks with dirt surface, so they are not included in the performance comparison.
Despite this, selected tracks are very diverse in their geometries. They all differ and represent many combinations of track width, turns geometry, change in elevation and turns sequences. HingyBot seems to be working best on narrow, dangerous, and complex tracks like Alpine1 or Street1 due to very efficient path planning and regulation.
However, it suffers from poor optimal speed estimation on broader tracks, where turns happen slower.
It is interesting to note that HingyBot's relative performance is best on the Alpine1 track, which belongs to the GRNDriver's primary learning set.

7.4 Further improvements

Many experimental features of the HingyBot were disabled. They all have expected or shown improvement in certain situations, but are still unreliable. Using them is likely to result in a fatal crash, which will outweigh any of their advances.

Those include:

- Gene Regulatory Network for the speedplan correction,
- Dangerousness metric (note it was enabled for the GRNDriver),
- advanced, dynamic PID tunning for a better control in unusual cases,
- online racing line optimization for further improvements of the racing line shape,
- online track's geometry correction for better path representation,
- overtaking strategies which will reduce risk of the car destruction during overtaking with large speed difference,
- advanced return to optimal racing line after unusual situations, like a crash.

With these features implemented reliably, performance of the HingyBot is expected to be even better.

Bibliography

[1] SCRC 2015 ladderboard. `http://cs.adelaide.edu.au/~optlog/SCR2015/awards.html`.

[2] Stéphane Sanchez; Sylvain Cussat-Blanc. *Gene regulated car driving: Using a gene regulatory network to drive a virtual car.* Springer, 2014. p. 29.

[3] Luigi Cardamone Daniele Loiacono and Pier Luca Lanzi. *Simulated Car Racing Championship Competition Software Manual.* April 2013.

[4] Erica Espié and Christophea Guionneau. *TORCS game engine.* 2001.

[5] Ralf Herbrich and Applied Games Group Thore Graepel; Microsoft Research Cambridge. *Playing Machines: Machine Learning Applications in Computer Games.* ICML, 2008.

[6] Mathias Vielwerth Julian Togelius Sebastian Risi1 Jacob Fischer, Nikolaj Falsted. *Monte Carlo Tree Search for Simulated Car Racing.* 2008.

[7] Dhruv Mittal James Martin, Namhoon Kim and Micaiah Chisholm. *Certification for Autonomous Vehicles.*

[8] P. Lanzi L. Cardamone, D. Loiacono and A. Bardelli. *Searching for the optimal racing line using genetic algorithms.* IEEE Symposium on Computational Intelligence and Games, 2010.

[9] Stefano Lecchi. *Artificial intelligence in racing games.* ACM, 2009.

[10] Daniele Loiacono Luigi Cardamone and Pier Luca Lanzi. *Learning to Drive in the Open Racing Car Simulator Using Online Neuroevolution.* IEEE Symposion on Computional Intelligence and AI in Games, 2014.

[11] Lesley T. MacNeil and Albertha J.M. Walhout. *Gene regulatory networks and the role of robustness and stochasticity in the control of gene expression.* Cold Spring Harbor Laboratory Press, 2011.

[12] Stéphane Sanchez Sylvain Cussat-Blanc, Jean Disset. *Dangerousness Metric for Gene Regulated Car Driving.* Springer, 2016.