POLITECNICO DI MILANO

Artificial Intelligence in Racing Games

Videogame Design and Programming

Daniele Loiacono

Racing AI in a nutshell

Strategic System

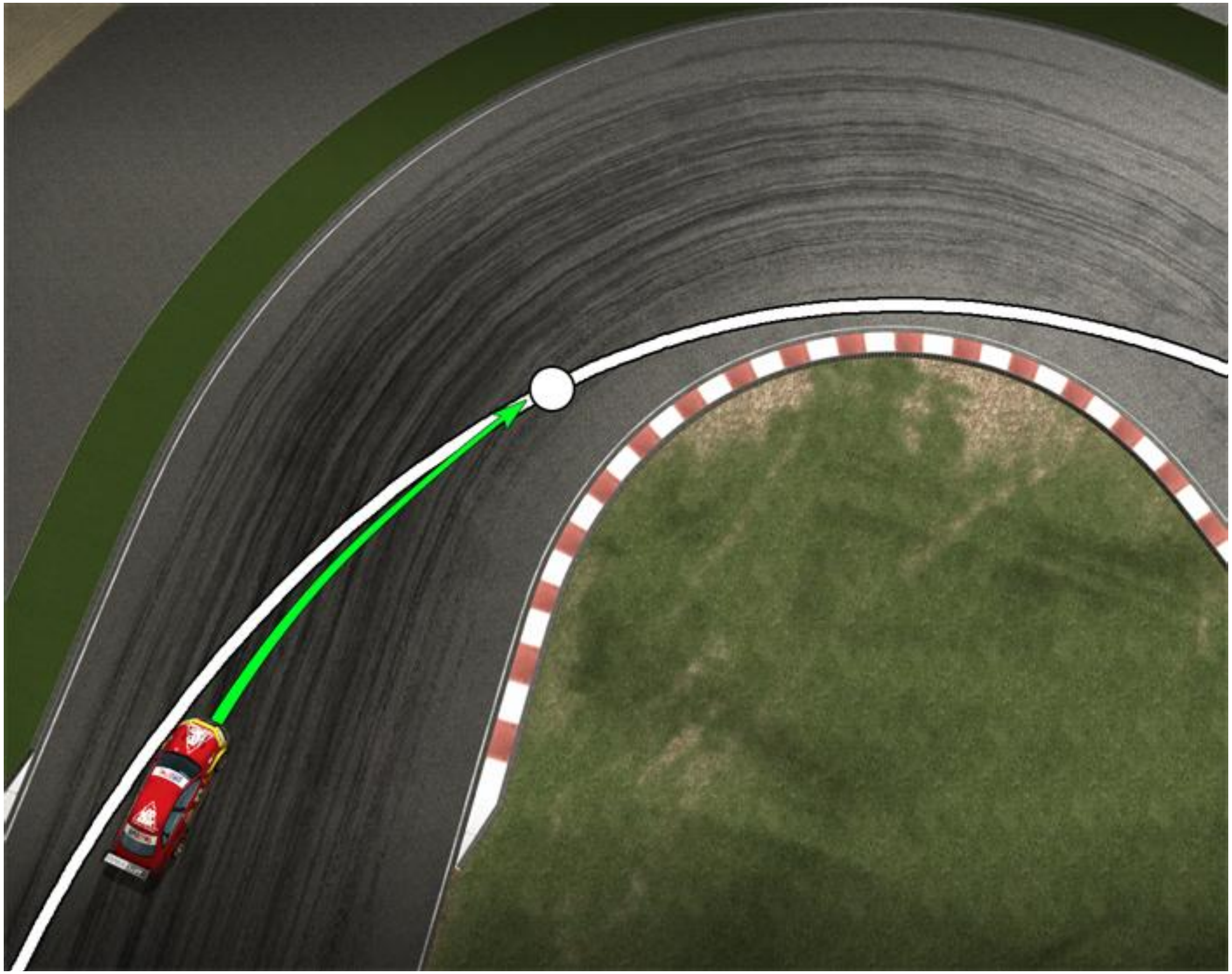
Tactical System

Control System

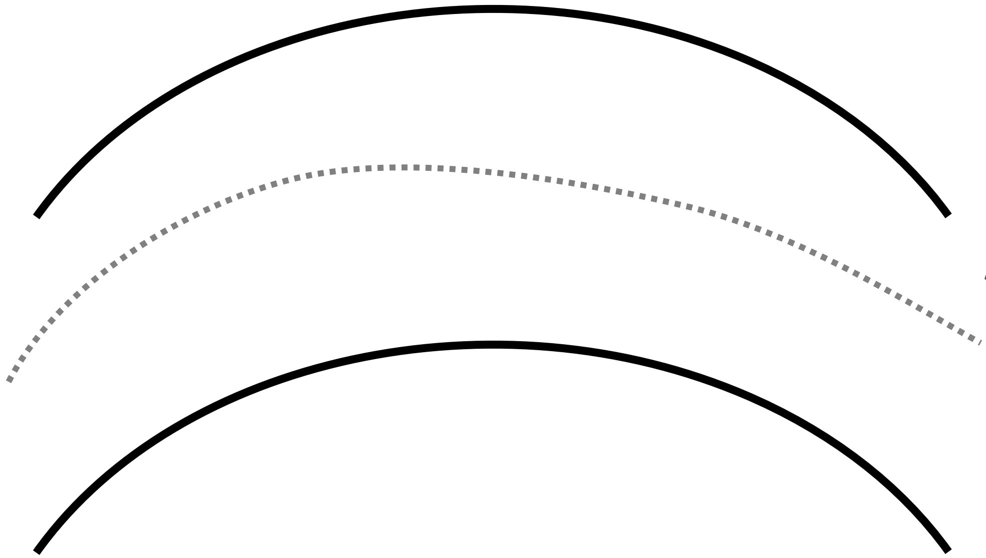
Control System

- Control system takes a control action on the basis of
  - current status of the vehicle
  - target position and speed
- Based on car and environment dynamics
- Might involve heuristics or approximations

What about the target?

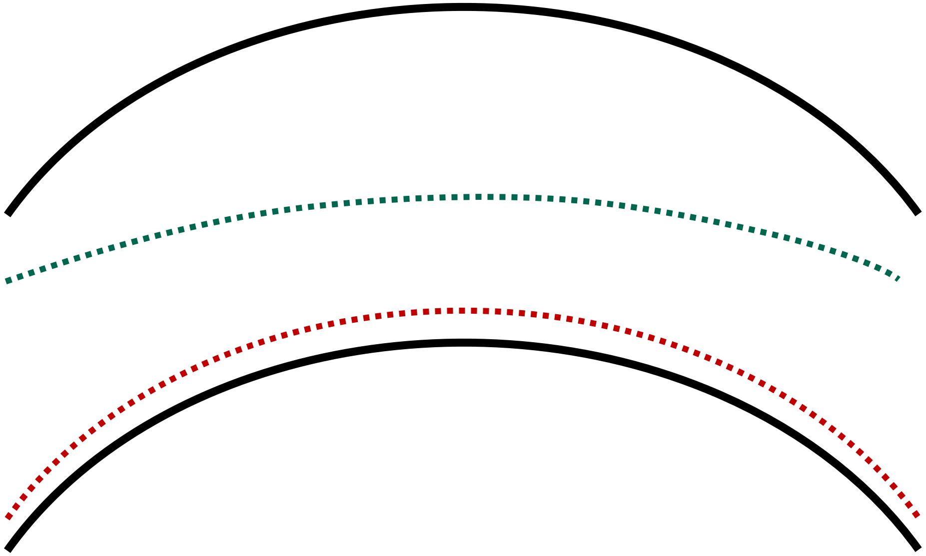


Understanding the problem…



$$v_{max} \approx \sqrt{\mu g r}$$

Understanding the problem…



Shortest path
or
minimum curvature ?

How to find the optimal racing line?

Expert Design

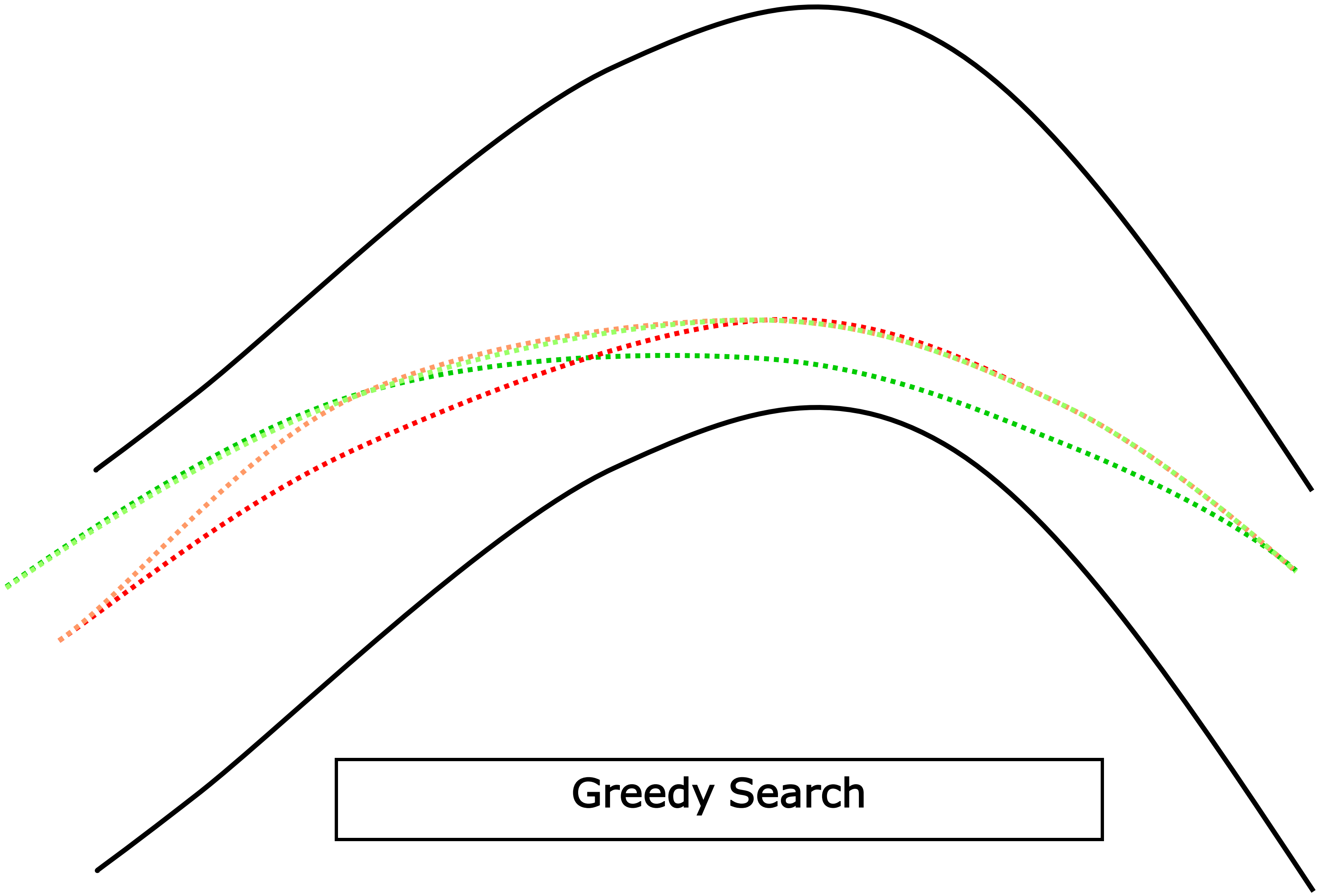
Heuristics

Model-Based Optimization

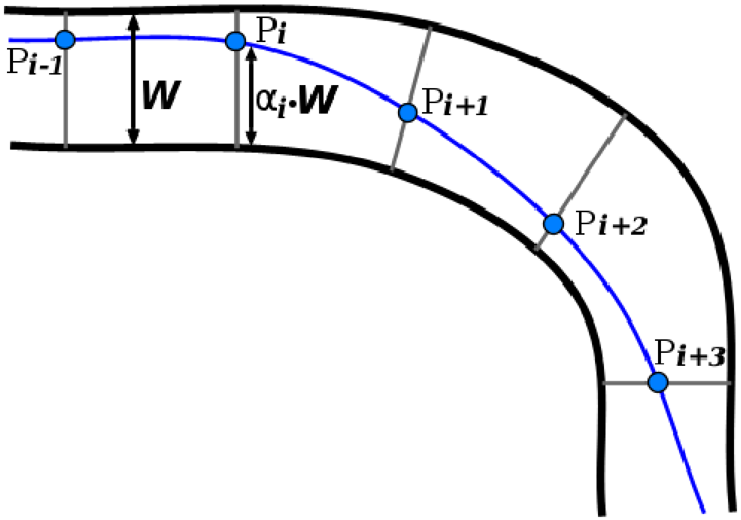
Genetic Algorithms

Expert design +Test

Heuristics



Model Based(1)



SP

$$\min_{\vec{\alpha}} \quad \vec{\alpha}^T H_S \vec{\alpha} + \vec{B_S}\vec{\alpha}$$
$$0 \le \alpha_i \le 1$$

MCP

$$\min_{\vec{\alpha}} \quad \vec{\alpha}^T H_\Gamma \vec{\alpha} + \vec{B_\Gamma}\vec{\alpha}$$
$$0 \le \alpha_i \le 1$$

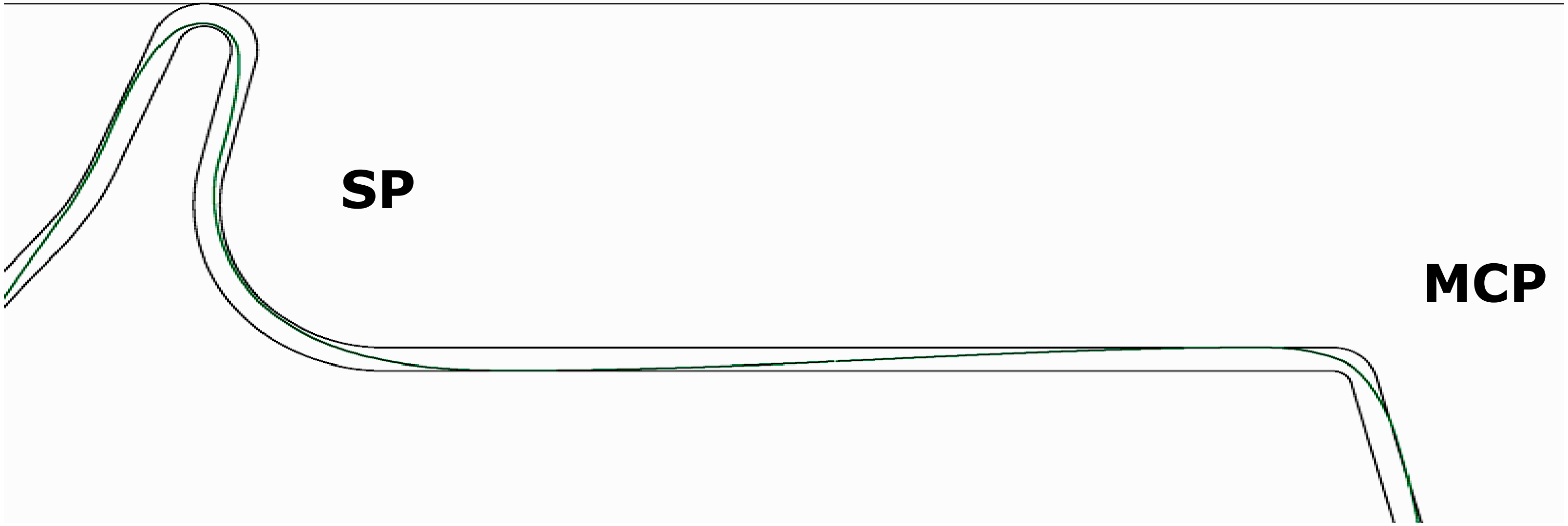
$$\frac{dS_0}{dt} = -S_0\left(f + S_u + S_{V*} + S_{uV} + S_{uu*} + S_{uVV} + S_{uVu*}\right)$$
$$\frac{dS_{V*}}{dt} = S_0\left(\frac{2f}{5} + S_{V*} + S_{uVV}\right)$$
$$\frac{dS_u}{dt} = S_0\left(\frac{3f}{5} + S_u + S_{uu*} + S_{uVu*} + S_{uV}\right) - S_u\left(\frac{4f}{5} + S_{V*} + S_{uVV} + \frac{2}{3}\left(S_u + S_{uV} + S_{uVu*} + S_{uu*}\right)\right)$$
$$\frac{dS_{uV}}{dt} = S_u\left(\frac{2f}{5} + S_{V*} + S_{uVV}\right) - S_{uV}\left(\frac{3f}{5} + \frac{1}{2}\left(S_{V*} + S_{uVV}\right) + \frac{2}{3}\left(S_u + S_{uVu*} + S_{uu*} + S_{uV}\right)\right)$$
$$\frac{dS_{uu*}}{dt} = S_u\left(\frac{2f}{5} + \frac{2}{3}\left(S_u + S_{uVu*} + S_{uu*} + S_{uV}\right)\right)$$
$$\frac{dS_{uVV}}{dt} = S_{uV}\left(\frac{f}{5} + \frac{1}{2}\left(S_{V*} + S_{uVV}\right)\right)$$
$$\frac{dS_{uVu*}}{dt} = S_{uV}\left(\frac{2f}{5} + \frac{2}{3}\left(S_u + S_{uV} + S_{uu*} + S_{uVu*}\right)\right)$$

Driver Model & Car Dynamics

Grid search of the best convex combination of SP and MCP

Model Based (2)

- Controllers in racing games are not ideal: models can lead to suboptimal performance
- It might be difficult to deal with any detail of the tracks
  - different type of borders (curbs, barriers, sand, grass)
  - bumps and banking
  - different friction
- One optimal trade-off between MCP and SP on the whole track?



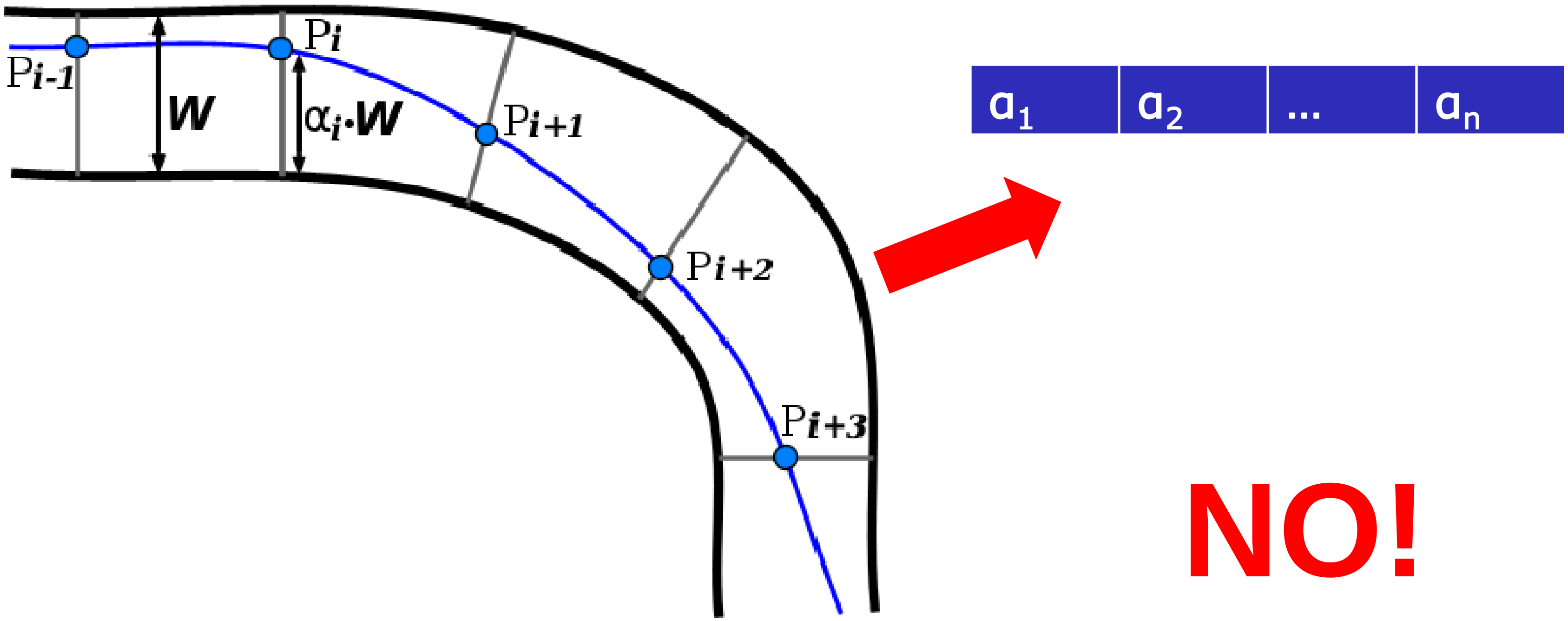
How to extend it?

Search for the best trade-off in *each* segment of the track

Replace models with the actual racing simulator

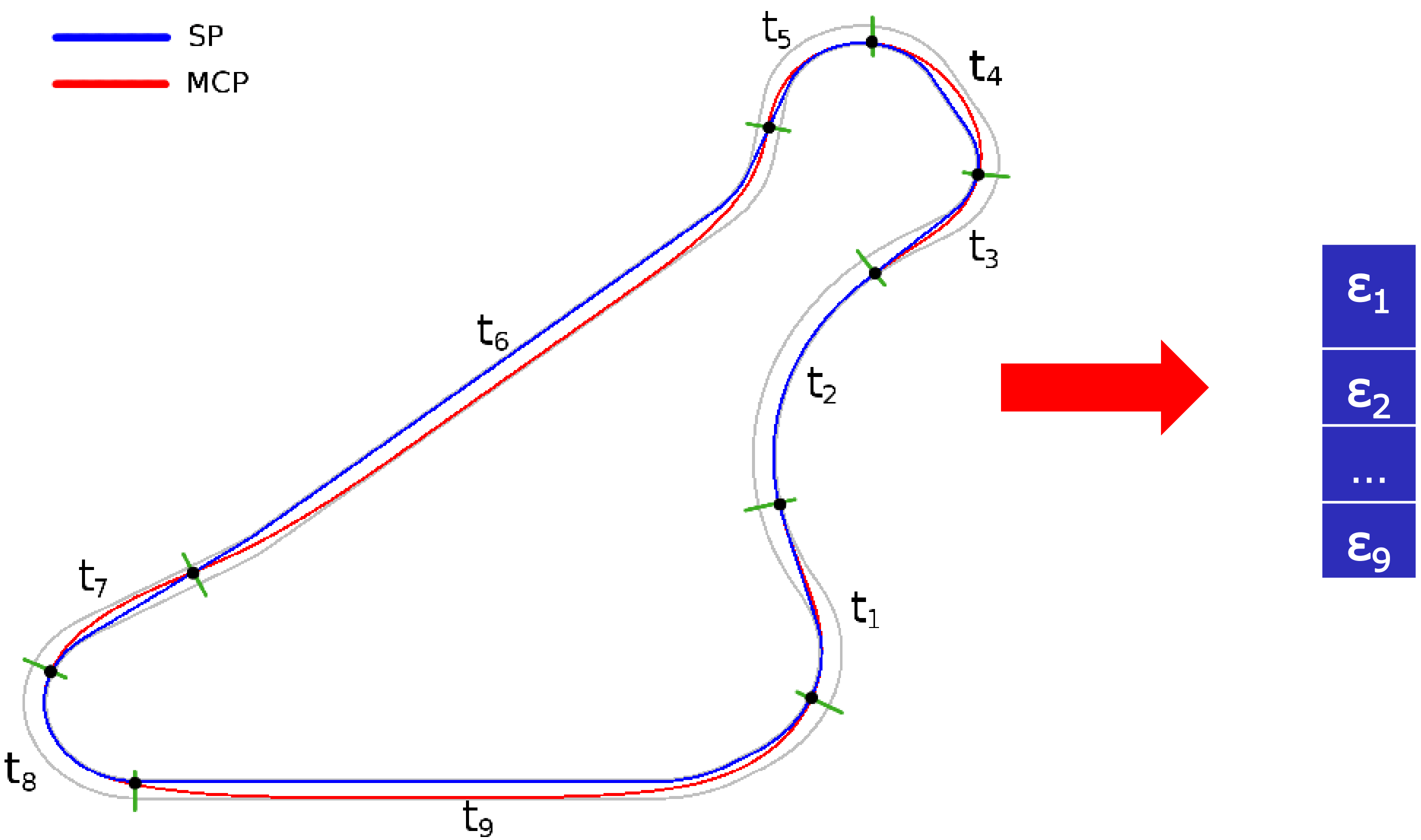
Replace the grid search with a GA

Genetic Algorithms (1)



- Too many variables!
- Does not exploit any domain information (i.e., SP and MCP)

Genetic Algorithms (2)



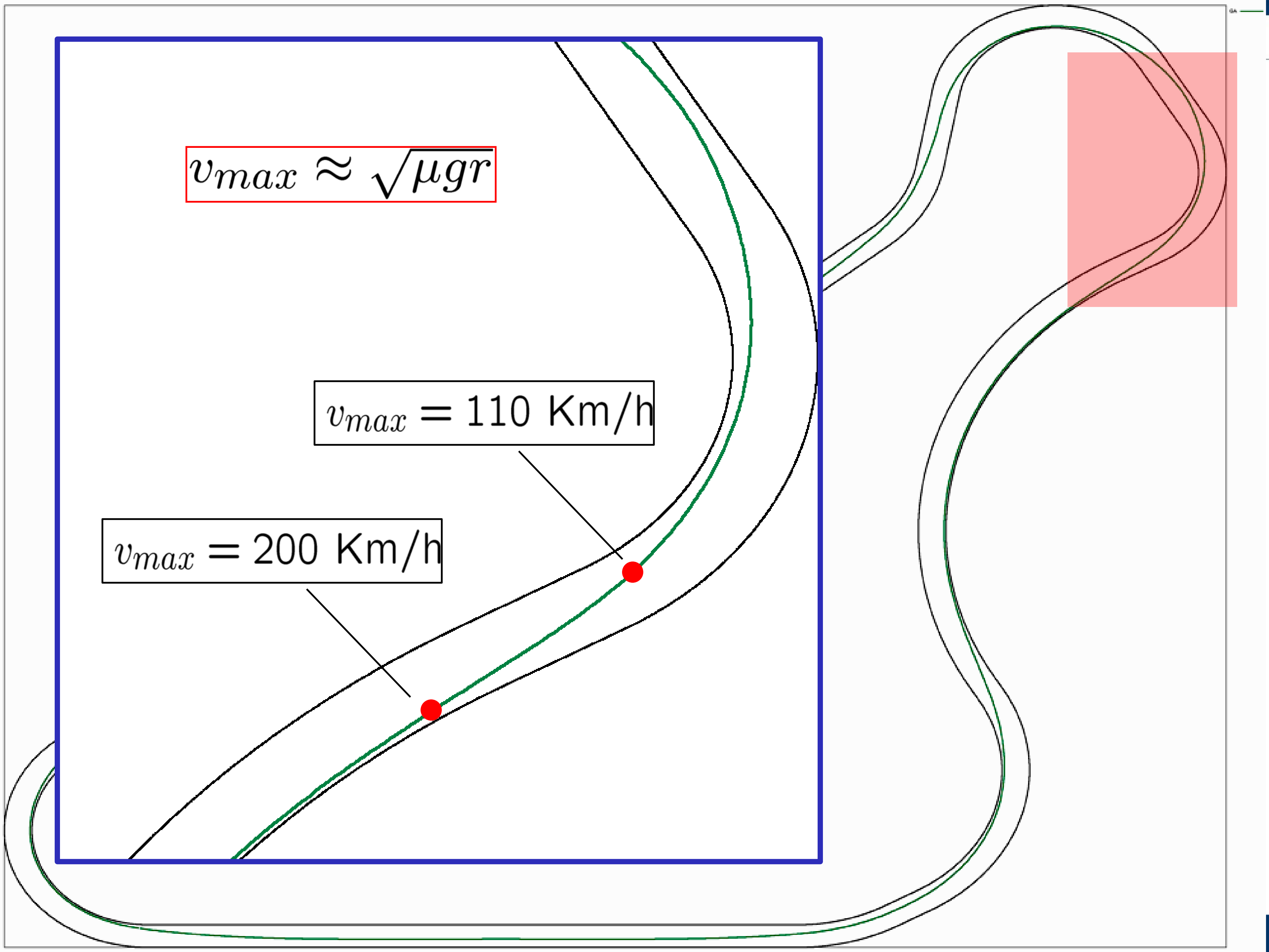
- Few variables (up to 30-40 in the most complex tracks)
- Exploits the knowledge of SP and MCP
- Continuous by design

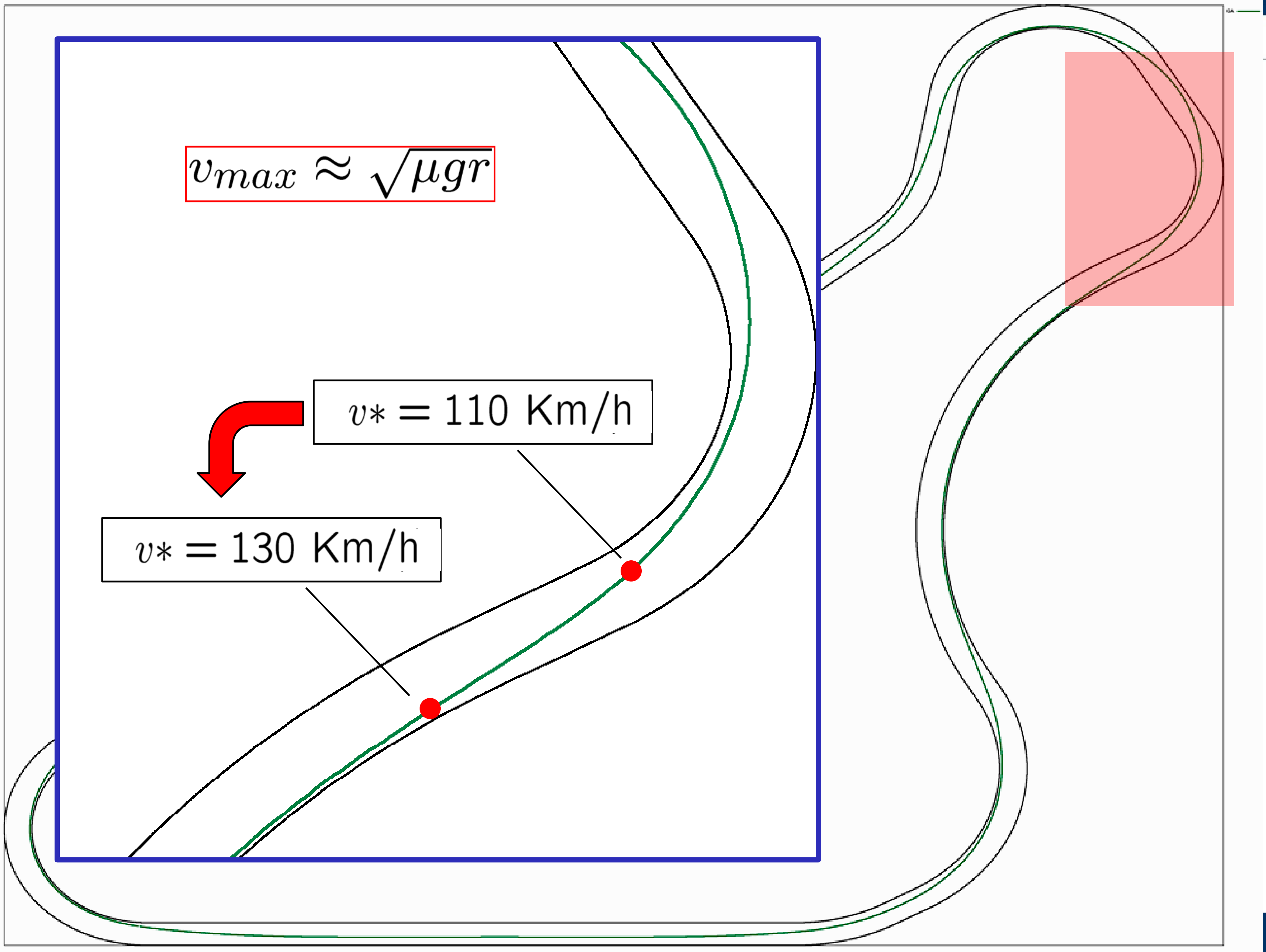
Genetic Algorithms (3)

- Results achieved in a case study:

Track	GA	Model-Based	Heuristics
Aalborg	69.928	+0.766	+0.834
Alpine 1	121.481	+1.063	+1.395
Alpine 2	92.527	+0.549	+1.807
A-Speedway	24.701	+0.437	+2.857
Forza	85.210	+0.476	+1.398
CG-Speedway	39.372	+0.422	+0.748
Michigan-Speedway	33.866	+0.024	-0.124
Olethros Road	111.656	+1.270	+2.974
Ruudskogen	62.732	+0.476	+0.474
Street 1	75.613	+0.511	-0.933
Wheel 1	74.887	+0.519	-0.963

How control system uses the racing line?


$v_{max} \approx \sqrt{\mu g r}$
v_{max} = 110 Km/h
v_{max} = 200 Km/h



$v_{max} \approx \sqrt{\mu g r}$
$v* = 110$ Km/h
$v* = 130$ Km/h

Following the racing line

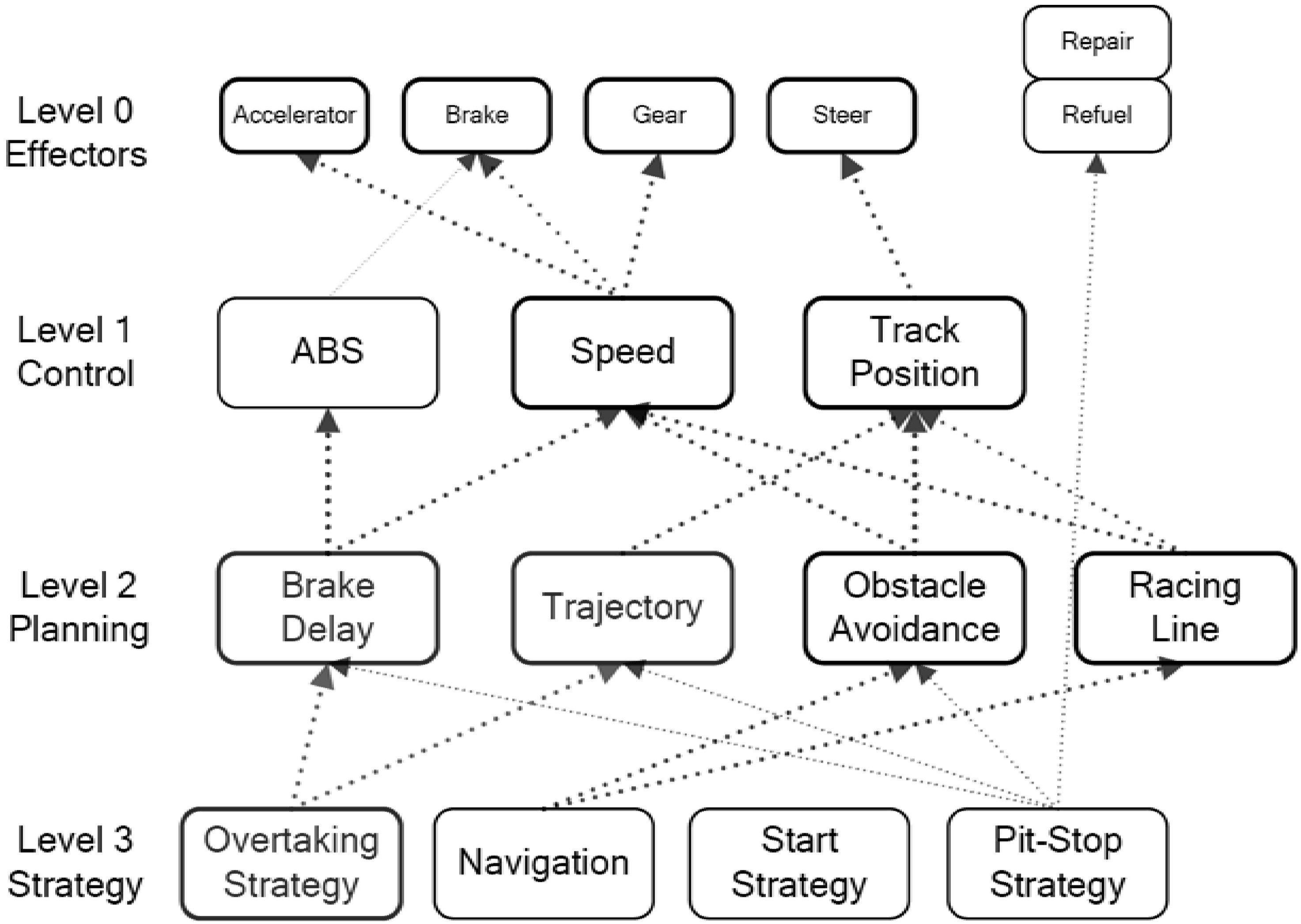
- Control system follows a racing line provided in input
- It is usually programmed based on the following domain knowledge:
  - Car parameters (e.g., engine power, brakes efficiency)
  - Environment parameters (e.g., friction of the asphalt)
  - In-game dynamics (e.g., aerodynamics)
- It is generally fine tuned to guarantee an optimal behaviour

Tactical System

Tactical system

- Performs complex maneuvers
  - Follows a preceding vehicle taking its slipstream
  - Overtakes when appropriate
  - Blocks following vehicles
- Handles specific situations
  - Avoids imminent collisions
  - Recovers the vehicle if it gets stuck against a border

Behavior Decomposition



How to design complex behaviors?

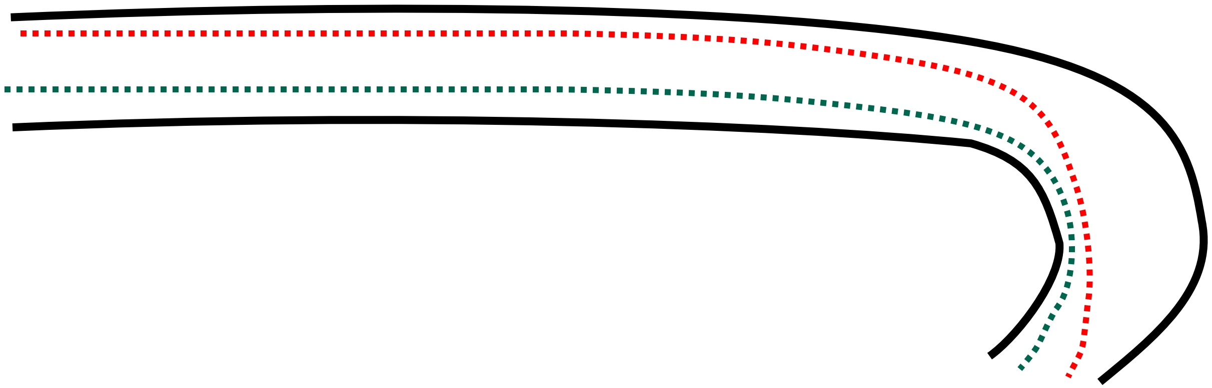
Programmed Heuristics

Domain Expert Rules

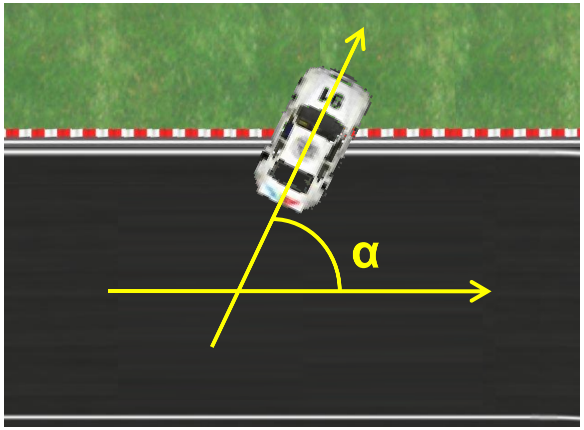
Learning

Examples of Heuristics

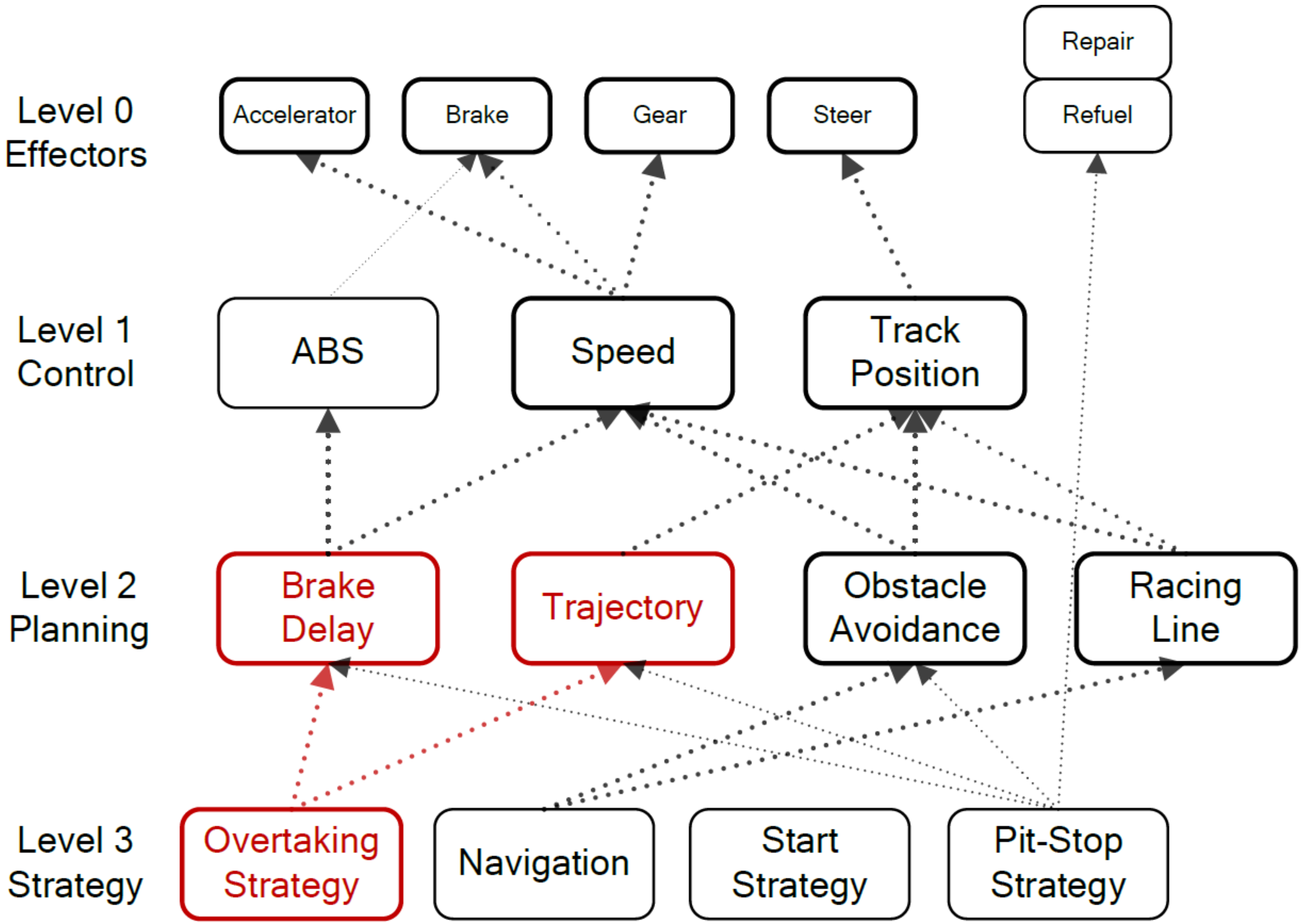
- Alternative racing lines for overtaking



- Programmed recovery policy

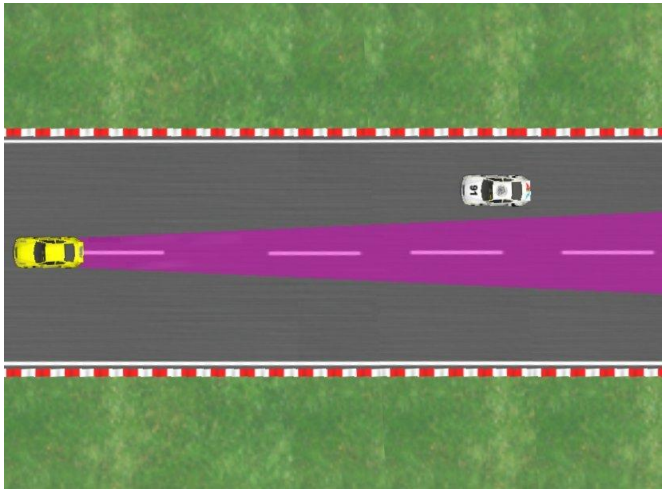


Learning driving behaviors

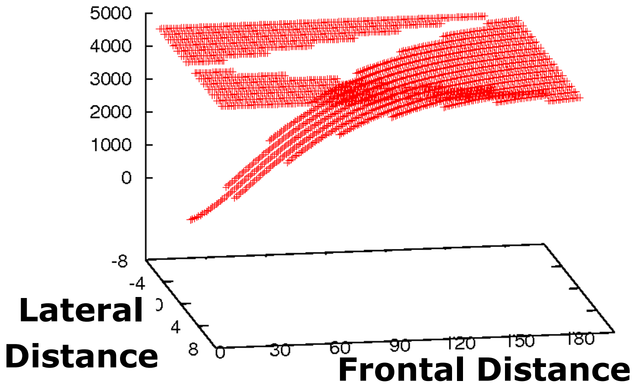


Learnig Overtaking Behavior: Problem Definition

- State Space
  - Frontal distance from the opponent car
  - Lateral distance from the opponent car
  - Distance from the side of the track
  - Speed difference
- Action
  - move 1m on right
  - keep current trajectory
  - move 1m on left
- Reward
  - +1 overtake completed
  - -1 collision or out of track
  - 0 otherwise



Aerodynamic Friction



Learning Brake Delay: Problem Definition

- State Space
  - Frontal distance from the opponent car
  - Distance from the next turn
  - Speed difference
- Action
  - Do **not** brake
- Reward
  - +1 overtake completed
  - -1 collision or out of track
  - 0 otherwise
- **Works on top of the driving policy**

Strategic System

Strategic system

- Balance AI skills
  - Determines how fast an AI should race depending to difficulty level, pilot skills, etc.
  - Forces mistakes at a realistic rate
- Handles resources in high simulative titles
  - Manages fuel consumption, tyre wear and damages
  - Chooses when go to pit

Examples of Strategic System

- Rubber band
  - Skill of vehicles behind the player is increased
  - Skill of vehicles ahead of the player is reduced
  - Too simple: has some drawbacks
- Scripted strategy
  - Much more customizable by designers
  - Allows different and, possibly, more realistic strategies
  - Offers more opportunities for research
  - Ref. Jimenez, E. (2008). The Pure Advantage: Advanced Racing Game AI. (http://www.gamasutra.com/)

How to get started?

Simulated Car Racing

- Simulated Car Racing is a **scientific** competition based on The Open Racing Car Simulator (TORCS)
- Competitors are provided with
  - a simple API (Java and C++) to build their own controller
  - a complete sensors/effectors model
- Goal of the competition is developing the fastest controller
- Competition software is open source and is a good starting point to **learn programming a racing AI**

http://cig.ws.dei.polimi.it/?page_id=134

http://groups.google.com/group/racingcompetition

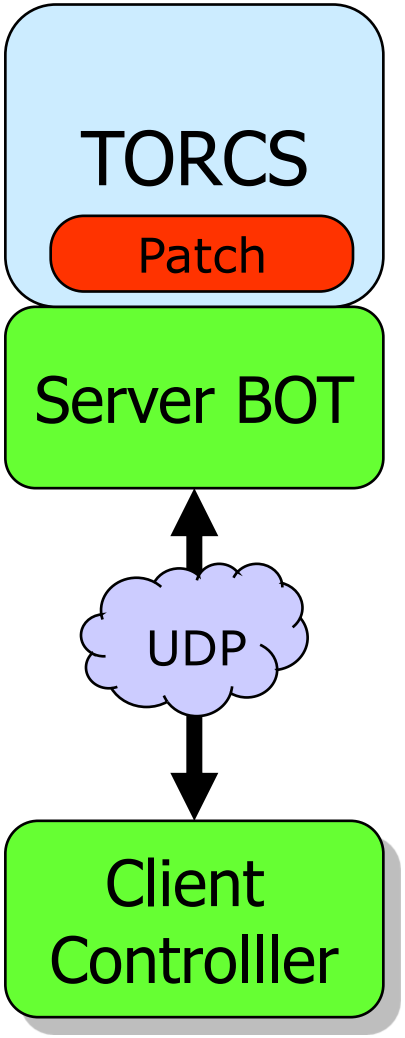
The Open Racing Car Simulator

- TORCS is a state of the art open source simulator written in C++
- Main features
  - Sophisticated dynamics
  - Provided with several cars, tracks, and controllers
  - Active community of users and developers
  - Easy to develop your own controller

- OS Support
  - Linux: binaries and building from sources
  - Windows: binaries and "limited" building from sources support
  - OSX: legacy binaries and no building from sources support ☹

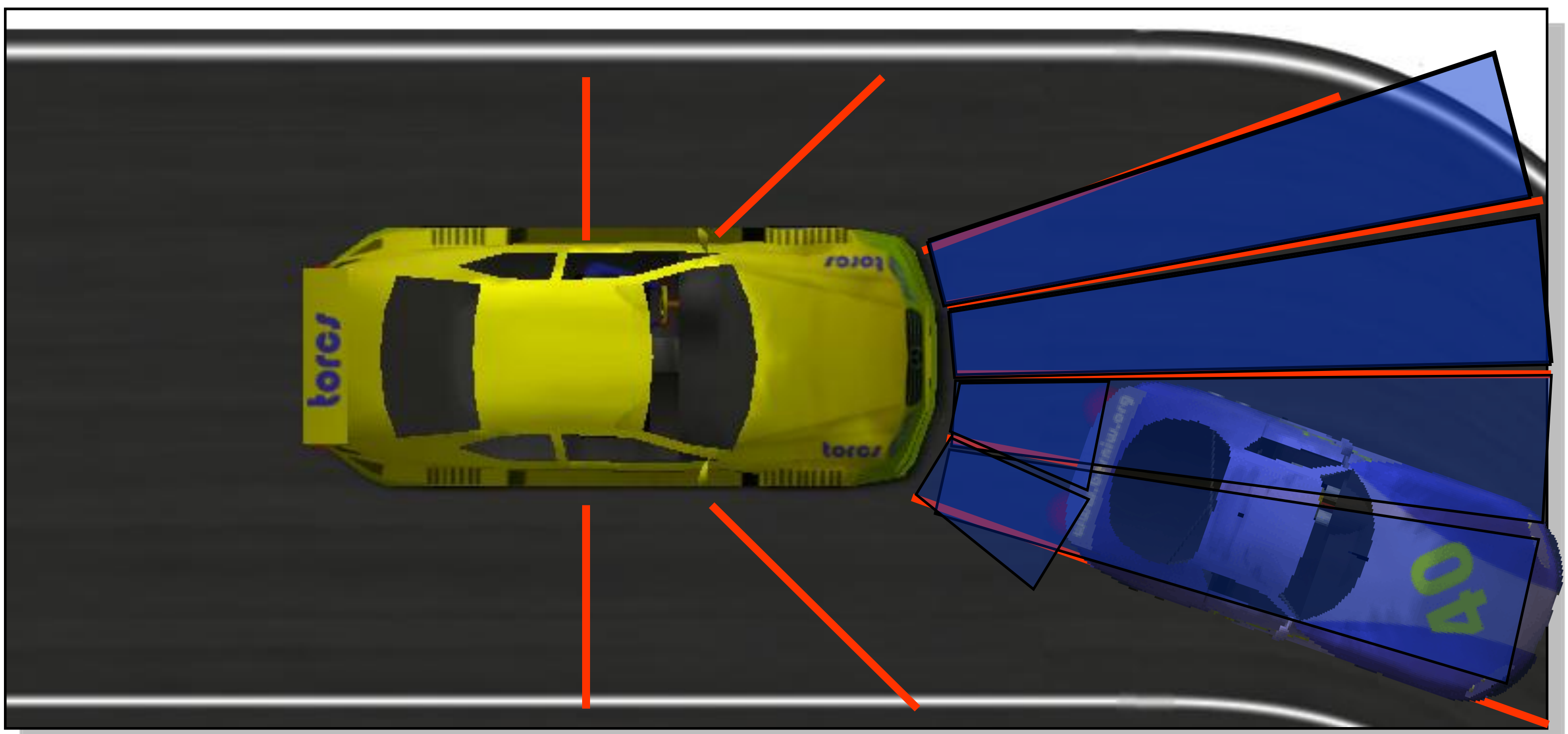
Software Overview

- To make TORCS more easy to use we developed an API based on socket (UDP)
- Values of sensors and effectors are sent through UDP
- 3 components
  - Torcs Patch
  - Server Bot (C++)
  - Client API (C++ and Java)



Main Sensors

- Rangefinders for…
  - …edges of the track
  - …opponents
- Speed, RPM, fuel, damage, angle with track, distance race, position on track, etc.



Sensors (1)

Name	Range (unit)	Description
angle	$[-\pi,+\pi]$ (rad)	Angle between the car direction and the direction of the track axis.
curLapTime	$[0,+\infty)$ (s)	Time elapsed during current lap.
damage	$[0,+\infty)$ (point)	Current damage of the car (the higher is the value the higher is the damage).
distFromStart	$[0,+\infty)$ (m)	Distance of the car 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 sensors returns the distance between the track edge and the car within a range of 200 meters. Sensor are affected by i.i.d. normal noises with a standard deviation equal to the 1% of sensors range. The sensors sample, every 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 $[-\pi/2,+\pi/2]$ 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., pos is less than -1 or greater than 1), the focus direction is outside the allowed range ($[-\pi/2,+\pi/2]$) 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.

Sensors (2)

gear	{-1,0,1,$\cdots$ 7}	Current gear: -1 is reverse, 0 is neutral and the gear from 1 to 7.
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 $\pi/18$ (10 degrees) within a range of 200 meters and returns the distance of the closest opponent in the covered area. Sensor are affected by i.i.d. normal noises with a standard deviation equal to the 5% of sensors range. The 36 sensors covers all the space around the car, spanning clockwise from $+\pi$ up to $-\pi$ with respect to the car axis.
racePos	$\{1,2,\cdots,N\}$	Position in the race with to respect to other cars.
rpm	[2000,7000] (rpm)	Rumber of rotation per minute of the car engine.
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.

Sensors (3)

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. Sensor are affected by i.i.d. normal noises with a standard deviation equal to the 5% of sensors range. The sensors sample the space in front of the car every 10 degrees, spanning clockwise from $+\pi/2$ up to $-\pi/2$ with respect to the car axis. When the car is outside of the track (i.e., pos is less than -1 or greater than 1), the returned values are not reliable.
trackPos	$(-\infty,+\infty)$	Distance between the car and the track axis. The value is normalized w.r.t to 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 means 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)	Distance of the car mass center from the surface of the track along the Z axis.

Main Effectors

- Basically 4 main effectors
  - Steering wheel [-1,+1]
  - Gas pedal [0, +1]
  - Brake pedal [0,+1]
  - Gearbox {-1,0,1,2,3,4,5,6,7}

Effectors

Name	Range	Description
accel	[0,1]	Virtual gas pedal (0 means no gas, 1 full gas).
brake	[0,1]	Virtual brake pedal (0 means no brake, 1 full brake).
clutch	[0,1]	Virtual clutch pedal (0 means no clutch, 1 full clutch).
gear	-1,0,1,···,7	Gear value.
steering	[-1,1]	Steering value: -1 and +1 means respectively full right and left, that corresponds to an angle of 0.785398 rad.
focus	[-90,90]	Focus direction (see the *focus* sensors in Table 1) in degrees.
meta	0,1	This is meta-control command: 0 do nothing, 1 ask competition server to restart the race.