Developing car driving agent in a virtual TORCS environment

(Tworzenie agenta kierującego pojazdem w wirtualnym środowisku TORCS)

Kacper Kulczak

Praca inżynierska

Promotor: dr Paweł Rychlikowski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

27 grudnia 2018

Abstract

...

...

Contents

Chapter 1

Introduction

That's why companies around the globe put a lot of effort into making the car software for autonomous driving. It is very interesting task from artificial intelligence perspective. During my work I wanted approach the problem by building my own agent capable of drive safely on a racing track.

1.1 TORCS Environment

"The online racing car simulator"(TORCS) is highly portable, multi platform car simulator, with various cars and tracks. The simulation features a simple damage model, collisions, fuel consumption, tire and wheel properties (springs, dampers, stiffness), aerodynamics and much more. It is designed to enable programed agents compete against each others. There is very ditalied instruction for developing yourn own bot. However the game and bots as well is written in C++ language.

TO REMOVE.

TO REMOVE. TO REMOVE.

TO REMOVE.

TO REMOVE. TO REMOVE. TO REMOVE. TO REMOVE. TO REMOVE. TO REMOVE. TO REMOVE. TO REMOVE. TO REMOVE. From the beginning I really wanted to use python language, so TORCS environment did not meet all my expectations.

Figure 1.1: Screen shot from TORCS race

1.2 Simulated Car Racing Championship

SCRC competition took place between 2007 - 2015 with some breaks. It was organized by the University of Adelaide and the Politecnico de Milano. THey used TORCS engine for competition, but organizers provided official patch which changed architecture of the programme. After patching, TORCS became client-server application which allows multiple bots communicate with game engine via UDP connections.

Server sends current sensor inputs (track border, speed, lap time, etc...) and waits for 20ms for the client action (gas, break, steer, etc...). API is details are descrived in table from manual. With that change participants cant choose whatever language they want. That's why I decided to use patched version of TORCS game engine in version 1.3.7

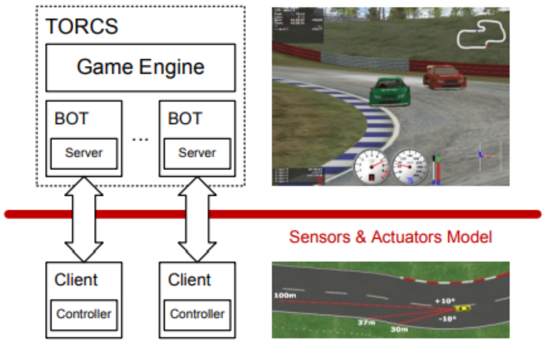

Figure 1.2: Simulated Car Racing Championship - architecture overview

Organizers provided "clients" programmes only in Java and C++. I didn't want to implement communication interface on my own, beacuse it's time consuming and uninteresting task. After some research, I found `http://xed.ch/p/snakeoil` project, which provides Python Class handling communication with TORCS server. It allowed me to add layer of abstraction and focus directly on driving functionality.

1.3 Game sensors and actions

Chapter 2

Experiments

2.1 Line follower

The first approach to the problem was developing simple bot which follows the middle line of the track. It uses following parameters: PosTrack. angle, sppedX

Chapter 3

Adidtonal Problems

- **Transmission** Choosen architecture didn't allowed me to use automatic transmission included with the torcs game. I had to build my own basic automatic transmission system, which is far from perfect. I haven't change it during development of different agents, so all results are comparable.

Bibliography

[1] ...

Apendix

Name	**Range (unit)**	**Description**
angle	$[-\pi, +\pi]$ (rad)	Angle between the car direction and the direction 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 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 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., pos 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	$\{-1, 0, 1, \ldots, 6\}$	Current gear: -1 is reverse, 0 is neutral and the gear from 1 to 6.
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, sensors are affected by i.i.d. normal noises with a standard deviation equal to the 2% of sensors range. The 36 sensors cover all the space around the car, spanning clockwise from -180 degrees up to $+180$ degrees with respect to the car axis.
racePos	$\{1, 2, \ldots, N\}$	Position in the race with respect to other cars.
rpm	$[0, +\infty)$ (rpm)	Number 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
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, 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. to the car axis) can be set by the client once during initialization, i.e., before the beginning of each race. 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 (typically -1 is returned).

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 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	$(-\infty, +\infty)$ (km/h)	Distance of the car mass center from the surface of the track along the Z axis

Table 3.1: : Description of the available sensors.

Name	**Range (unit)**	**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).
cell1	$[0, 1]$	Virtual clutch pedal (0 means no clutch, 1 full clutch).
cell1	$\{-1, 0, 1, \ldots, 6\}$	Gear value.
cell1	$[-1, 1]$	Steering value: -1 and $+1$ means respectively full right and left, that corresponds to an angle of 0.366519 rad.
cell1	$[-90, 90]$	Focus direction (see the focus sensors) in degrees.
cell1	$\{0, 1\}$	This is meta-control command: 0 do nothing, 1 ask competition server to restart the race.

Table 3.2: Description of the available action parameters.