

**POLITECNICO DI MILANO**  
**Computer Science and Engineering Master Degree**  
**Dipartimento di Elettronica Informazione e Bioingegneria**



## **TITOLO DELLA TESI**

**Artificial Intelligence and Robotic Laboratory  
of Politecnico di Milano**

**Supervisor: Prof. Marcello Restelli**

**Co-supervisors: Alessandro Lavelli, Dott.**

**Master thesis of:**  
**Nome Cognome Matricola**

**Academic Year 2018-2019**



# Abstract

TESTO SOMMARIO IN INGLESE



# Sommario

TESTO DEL SOMMARIO



# Contents

<b>Abstract</b>	<b>I</b>
<b>Sommario</b>	<b>III</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Context</b>	<b>5</b>
<b>3 Problem Statement</b>	<b>7</b>
<b>4 State of the Art</b>	<b>9</b>
4.1 Theoretical Background . . . . .	9
4.1.1 Reinforcement Learning . . . . .	9
4.1.1.1 Markov Decision Process . . . . .	10
4.1.1.2 Environment . . . . .	11
4.1.1.3 Reward . . . . .	11
4.1.1.4 Policy . . . . .	11
4.1.2 Policy Evaluation . . . . .	12
4.1.3 Temporal Difference Learning . . . . .	12
4.1.3.1 SARSA . . . . .	12
4.1.3.2 Q-Learning . . . . .	12
4.1.3.3 Issues with Temporal Difference Learning . .	12
4.1.4 Deep Reinforcement Learning . . . . .	12
4.1.4.1 Actor-Critic Methods . . . . .	12
4.1.4.2 Fitted Q-Iteration . . . . .	12
4.1.4.3 Deep Deterministic Policy Gradient . . . . .	12
4.2 Related Works . . . . .	15
4.2.1 Reinforcement Learning in Videogames . . . . .	15
4.2.2 Autonomous Driving . . . . .	15
4.2.2.1 Reinforcement Learning in Autonomous Driving . . . . .	17

<b>5</b>	<b>Methods</b>	<b>19</b>
<b>6</b>	<b>Experiments</b>	<b>21</b>
<b>7</b>	<b>Conclusion</b>	<b>23</b>
	<b>Bibliography</b>	<b>25</b>



# List of Figures



# List of Tables



# Chapter 1

## Introduction

In the last decade autonomous driving has been deeply studied in several fields of research: automotive, military, racing cars, and even flight. The goal is to disrupt the need of a human driver or pilot, resulting in a reduction in costs and an increase of safety, for instance by reducing the number of accidents per year due to human error. Moreover, they could learn from their errors, and transmit to each other autonomous vehicle instantaneously, so that each of them is constantly up to date. However, researches in the area of autonomous driving traces back to the birth of the first computers. In the '40, the scientific community were trying and gain insight about the human mind, in order to build artificial agents that could replicate -or even outperform- human in some specific tasks. That's how computer science branched into paths such as artificial intelligence and robotics. One of the main goal of the research of artificial intelligence has always been autonomous driving, that is the capability for an agent of perceiving the world and moving accordingly in order to reach a specific target. Autonomous cars consists in some piece of hardware capable of moving, such as a traditional car, or a drone, equipped with some sensors give insight on the environment, one or more computer units with a decision making algorithm, which compute an action to perform by means of some actuators, such as a throttle, a break, a steering, or a transmission. One of the first experiments in embodying some intelligence in an artificial agent were William Grey Walter's Turtle Robots in the '50s. They were capable of moving in the surroundings by sensing the environment in a simplified manner. They consisted in front wheel drive tricycle-like robots covered by a clear plastic shell, and were provided with a photocell and a bump detector as sensors, which resulted in the action of a motor. Despite their simple behaviors, the technique Walter used are reflected in today's reactive and biologically-inspired robots such as

those based on the B.E.A.M philosophy. Later on, in 1989, a new tentative of building an autonomous car was ALVINN, which stands for Autonomous Land Vehicle In a Neural Network, developed for military research. At that time the technology was not sophisticated enough to provide the computation required to drive in real time, but in a sense the premise of the algorithm used nowadays was already there. In fact, neural networks are today the essential tools for building an autonomous car. Later on with the advent of more and more sophisticated electronic components, such as sensors (Lidars and Radars, which are capable of scanning the environment at 360 degree via electromagnetic waves), and with the continuous improvement of electronic components, such as GPUs, autonomous driving started being a hot topic in industry beside scientific research. In the last decade, some of the traditional automotive companies, such as BMW, Mercedes-Benz, General Motors, Audi, started investing in this reasearch providing their cars with a multitude of sensors and algorithms to make their cars autonomous. Ford is another manufacturer with deployments already in play, with self-driving vehicles being tested in Pittsburgh, Palo Alto, Miami, Washington D.C. and Detroit, with Austin, Texas joining them soon. Together with its partner Argo AI, Ford has plans to trial its fleet of self-driving cars in Austin with a view towards launching a wider-reaching autonomous taxi and delivery service in 2021. Tesla, one of the companies founded by Elon Musk, is also making big steps forward in taking autonomy into mainstream use, both in terms of real world use cases and potential monetization of self-driving technologies. Tesla has supplied customers with more than 780,000 vehicles since launching, the majority of which arrive with pre-installed, self-driving capabilities available to users who purchase the requisite software. Tesla autonomous vehicles have logged huge levels of miles driven since their introduction, growing from 0.1 billion miles in May 2016 to an estimated 1.88 billion miles as of October 2019. Waymo, the newborn firm from Google's Alfabet, has been carrying out successful trials of autonomous taxis in California, transporting over 6,200 people in the first month and many thousands since. They're proving a practical business case for autonomous vehicles. Also in the U.S., Walmart is using autonomous cargo vans to deliver groceries in Arizona, while Pizza Hut is working with Toyota on a driverless electric delivery vehicle that even has a mobile kitchen in it to cook pizzas en route to your house. Parallely, in the military field, DARPA (Defense Advanced Research Projects Agency) has been proposing every year since 2007 a challenge called DARPA Grand Challenge, in which scientist teams dare each other to reach some targets as fast as possible. The reasons for this race to the autonomous driving are millions of possible accidents avoided

per year, and a reduction of pollution and costs by sharing cars which are capable of transporting people without human intervention needed. This growth in the research on autonomous cars led to a formalization of the levels of automation: Society of Automotive Engineers (SAE) introduced six levels of automation: level 0 is no automation, that is traditional cars we are accustomed to. With Level 1, driver assistance, relating to computer assistance of simple driving functions like the cruise control or automated braking systems. Cruise control consists in the capability of maintaining a certain target speed, whatever the slope of the road, weather condition, asphalt roughness. It can be accomplished via some speed sensors and a PID controller, there is no need of complex artificial intelligence. Automated braking systems involves stopping the car or reducing the speed whenever an object come across, be it a vehicle or a pedestrian. It requires proximity and distance sensors (ultrasonic or laser sensors) and may follow some manual rules based on thresholds. Lane Crossing Alert makes the car capable of notify whenever it crosses another lane, and can be achieved with camera sensors. Level 2 refers to partial automation, where the vehicle assists drivers with steering or acceleration, allowing drivers to disengage from some tasks. For example instance , Adaptive Cruise Control, Lane Keeping, or self-parking. Level 3 concerns conditional automation, where the vehicle takes over some of the monitoring of the environment from the driver, using sensor technology like LiDAR. That's what the company Tesla is currently developing: their cars are able of moving in the surroundings but the human intervention is still required in dangerous situations. Level 4 is high automation, where much greater control has been handed to the vehicle, which is in charge of steering, braking, accelerating, monitoring the vehicle and roads, and also responding to events like deciding when to change lanes, turn or use signals. Level 5 is full automation. No company is currently able to reach this level of automation. Currently, most of the cars are embodied with level 1 or 2. However, level 3 and 4 are still object of research, especially by tech companies such as Tesla and Waymo, whose promise is to reach these levels of automation in ten years, and level 5 in twenty, enabling their cars with the power of fully replacing human drivers. The reasons why today full autonomous driving has not being implemented yet are the extremely huge amount of data required for perceiving the complex world of the urban scenario and for computing the consequent actions. In fact, in order to be able to perform this vast computations, several computers and GPUs are needed aboard on the car, resulting in cost, weight, and power consumption. This is the strategy adopted by Tesla so far, whose cars to day span from a price of 85000 dollars to 120000 dollars, and weigh about

2000kg. Another approach, adopted by companies such as Google, is to perform computation on remote servers in their datacenters. However, current mobile connections such as 4G, makes impractical the transmission of data provided each second by the vast amount of sensors. In the future, the advent of 5G could be a game changer in this sense, which should provide a larger in bandwidth and more stable internet connection. That's why the scientific community is starting downstepping the complexity of the task, trying to build autonomous agents in a closed and controlled environment. This lead to a simplification of the problem, avoiding the need of a real time mapping of the environment, and excluding unattended events such as pedestrian coming across. Whether it's a matter of hardware or software, today such goal is still out of reach. Rather, some firms are focusing their attention on making a car which is autonomous in a specifing task. For example, good results have been achieved in keeping a lane on a highway, or stopping with a pedestrian coming across. Alternatively, good or complete levels of autonomy could be achieved in , where the perception part could be performed by simple sensors such as gps and odometry sensors. An example could be a race track, such as Formula 1 tracks, where the environment is known apriori, and that's where our thesis move into. In this paper we will tackle the problem of following a trajectory driven beforehand by a human driver on a race car and, if possible, to improve it.



## Chapter 2

# Context

TESTO CAPITOLO 2



La prima sezione io la metterei nell'introduzione

## Chapter 3

Related works

### Problem Statement

In this chapter we briefly discuss related work and techniques used for auto. driving

TESTO CAPITOLO 3

`\section{Nome}`

`\label{L}`

Per riferimenti  
alla sezione  
`\ref{L}`

artificial intelligence

Non chiaro.  
perché si tratta  
di multi agents ma  
non se ne è mai  
parlato

artificial intelligence's research

è più una def.  
di RL

per andare a  
capo all'inizio  
del tex thesis  
`\hyphenation{`  
`tra-di-tio-nal}`

mettere la  
citazione



The experiments

citazione  
citazione

citazione



## Chapter 4

# State of the Art

The objective of this chapter is to go into detail about key aspects of reinforcement learning, the core topic of this thesis. In particular, we will describe the concept of Decision Markov Process, the experiment environment, the concept of state, actions, and reward function. Eventually, we will give insight about the algorithms we used in this project: Fitted Q-Iteration, and Deep Deterministic Policy Gradient. Then, in the following section, we will show some successful application on reinforcement learning to autonomous driving, videogames, and in particular racing games.

### 4.1 Theoretical Background

#### 4.1.1 Reinforcement Learning

Together with Supervised Learning and Unsupervised Learning, Reinforcement Learning is a branch of Machine Learning, the discipline which studies the way a computer program can learn from experience and improve its performance at a specified task. Whereas Supervised Learning algorithms are based on inductive inference where the model is typically trained using labelled data to perform classification or regression, and Unsupervised Learning exploits techniques of density estimation and clustering applied to unlabelled data, in the Reinforcement Learning paradigm an autonomous agent learns to improve its performance at an assigned task by interacting with its environment. Typically, Reinforcement Learning agents are not explicitly taught how to act by an expert. Instead, the agent is free to explore the environment in which it lives, and its performance is evaluated by a reward function. Its goal is to maximize the reward by acting accordingly. The resulting reward results from the sum of the reward gained at every single step in its exploration process: for each state, the agent chooses an

*citazione*



toda  
 action to take and receives a reward based on the usefulness of its decision. Eventually, the agent learns the way to obtain the highest reward by exploiting knowledge learned about the expected utility of different state-action pairs. The challenge of Reinforcement Learning is to design the best tradeoff between exploration and exploitation, that is the capability of an agent of use its knowledge to obtain high rewards and, by contrast, being capable of exploring new possibilities in such a way not to remain stuck in a local optimum. Intuitively, the exploration process is high at the first iterations of the learning process, while it should decrease with accumulating knowledge.

#### 4.1.1.1 Markov Decision Process

The Reinforcement Learning problem can be formalized as a Markov Decision Process. A MDP is a tuple composed by:

- A finite set of states  $D$ : it encompasses every possible state of the process
- A finite set of actions  $A$ : it represents the possible action the agent can take at any moment
- A reward function  $r = \psi(s_t, a_t, s_{t+1})$ , which represents the reward given to the agent at state  $s_t$  taking the action  $a_t$  landing in the state  $s_{t+1}$
- A transition probability model  $T(s_t, a_t, s_{t+1}) = p(s_{t+1}|s_t, a_t)$ , which indicates the probability of landing in a state  $s_{t+1}$  being in a state  $s_t$  taking an action  $a_t$

\section{}  
 This provides the framework in which Reinforcement Learning operates: its goal is to find a policy  $\pi$  which specifies the actions to take in each state in order to maximize the reward over time i.e. fulfill the task. Reinforcement learning can solve Markov decision processes without explicit specification of the transition probabilities; the values of the transition probabilities are needed in value and policy iteration. In reinforcement learning, instead of explicit specification of the transition probabilities, the transition probabilities are accessed through a simulator that is typically restarted many times from a uniformly random initial state. Reinforcement learning can also be combined with function approximation to address problems with a very large number of states. Reinforcement Learning operates under the Markov Assumption, under which the current state represents all the information

In generale ogni metodo che viene citato deve avere il suo riferimento nella bibliografia con bibtex.

needed to take an action, regardless from the past states and actions. In other words: "The future is independent of the past given the present"

#### 4.1.1.2 Environment

The environment is the mathematical representation of the world in which the agent operates. It reflects the set of features of a machine learning problem, which could be numerical (such as e.g. coordinates, velocities, acceleration) or raw (such as images or signals). On the representation of the environment depends the choice of the algorithm suitable to perform a learning process.

#### 4.1.1.3 Reward

At each timestep, the agent receives a reward by the reinforcement learning algorithm, accordingly to the usefulness of action taken. At the end of an episode (a finite sequence of states) the return is typically computed as

$$R_t = \sum_{k=0}^T \gamma^k r_{t+k} + k + 1$$

where  $\gamma$  represents a discount factor comprised between 0 and 1, which stands for a learning parameter that influences how the agent considers future or present reward. A small  $\gamma$  results in a higher consideration of present rewards than the future (often called "myopic" evaluation), while with a  $\gamma$  near to 1, the agent will consider equally the reward coming from the present either from the future (also called "far-sighted" evaluation). The role of the discount factor resides in the fact that, besides being mathematically convenient, it avoids infinite returns in cyclic Markov Process, and it's capable of addressing a non-fully represented uncertainty about the future.

#### 4.1.1.4 Policy

The goal of reinforcement learning is finding a policy  $\pi$ , which maps states to a probability distribution over the actions, in order to maximize the return. This policy is called optimal policy  $\pi^*$ . If from a state  $s_t$  the agent takes always the same action  $a_t$ , the policy is deterministic:  $\pi(s_t) = a_t$ . Otherwise, if the taking of an action rather than another is due to a probability distribution, the policy is called stochastic:  $\pi(a_t|s_t) = p_i, 0 \leq p_i \leq 1$

### 4.1.2 Policy Evaluation

In order to evaluate the usefulness of a policy, we now introduce the concept of Value Function, which is computed as the discounted sum of rewards as discussed before in the return.

$$V^\pi(s) = \mathbb{E}_\pi[R_t | s_t = s] = \mathbb{E}_\pi \left[ \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} | s_t = s \right] = \mathbb{E}_\pi \left[ r_{t+1} + \sum_{i=0}^{\infty} \gamma^i r_{t+i+2} | s_t = s \right]$$

. . . finire

Parallel to the value functions is the concept of action-value function, or Q-value, which is the expected return after taking an action  $a_t$  in state  $s_t$  and thereafter following policy  $\pi$ :

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i > t} \sim, a_{i > t} \sim \pi} [R_t | s_t, a_t]$$

from which can be derived the recursive form called Bellman Equation:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E} [r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi} [Q^\pi(s_{t+1}, a_{t+1})]]$$

### 4.1.3 Temporal Difference Learning

#### 4.1.3.1 SARSA

#### 4.1.3.2 Q-Learning

#### 4.1.3.3 Issues with Temporal Difference Learning

### 4.1.4 Deep Reinforcement Learning

#### 4.1.4.1 Actor-Critic Methods

The "Critic" estimates the value function. This could be the action-value (the Q value) or state-value (the V value). The "Actor" updates the policy distribution in the direction suggested by the Critic (such as with policy gradients). and both the Critic and Actor functions are parameterized with neural networks. In the derivation above, the Critic neural network parameterizes the Q value - so, it is called Q Actor Critic.

#### 4.1.4.2 Fitted Q-Iteration

#### 4.1.4.3 Deep Deterministic Policy Gradient

It is not possible to straightforwardly apply Q-learning to continuous action spaces, because in continuous spaces finding the greedy policy requires an



optimization of a  $t$  at every timestep; this optimization is too slow to be practical with large, unconstrained function approximators and nontrivial action spaces. Instead, here we used an actor-critic approach based on the DPG algorithm. 23

- FQI


When the state and action spaces are finite and small enough, the  $Q$ -function can be represented in tabular form, and its approximation (in batch and in on-line mode) as well as the control policy derivation are straightforward. However, when dealing with continuous or very large discrete state and/or action spaces, the  $Q$ -function cannot be represented anymore by a table with one entry for each state-action pair. Moreover, in the context of reinforcement learning an approximation of the  $Q$ -function all over the state-action space must be determined from finite and generally very sparse sets of four-tuples. To overcome this generalization problem, a particularly attractive framework is the one used by Ormonite and Sen (2002) which applies the idea of fitted value iteration (Gordon, 1999) to kernel-based reinforcement learning, and reformulates the  $Q$ -function determination problem as a sequence of kernel-based regression problems. Actually, this framework makes it possible to take full advantage in the context of reinforcement learning of the generalization capabilities of any regression algorithm, and this contrary to stochastic approximation algorithms (Sutton, 1988; Tsitsiklis, 1994) which can only use parametric function approximators (for example, linear combinations of feature vectors or neural networks). In the rest of this paper we will call this framework the fitted  $Q$  iteration algorithm so as to stress the fact that it allows to fit (using a set of four-tuples) any (parametric or non-parametric) approximation architecture to the  $Q$ -function. The fitted  $Q$  iteration algorithm is a batch mode reinforcement learning algorithm which yields an approximation of the  $Q$ -function corresponding to an infinite horizon optimal control problem with discounted rewards, by iteratively extending the optimization horizon (Ernst et al., 2003). At each step this algorithm may use the full set of four-tuples gathered from observation of the system together with the function computed at the previous step to determine a new training set which is used by a supervised learning (regression) method to compute the next function of the sequence. It produces a sequence of  $Q$   $N$  -functions, approximations of the  $Q$   $N$  -functions defined by Eqn (5). -i mettere algoritmo a pag 6 di tree based batch reinforcement learning




per gli scenti  
 $\{a\}$

- extra trees Besides Tree Bagging, several other methods to build tree ensembles have been proposed that often improve the accuracy with respect to Tree Bagging (e.g. Random Forests, Breiman, 2001). In this paper, we

evaluate our recently developed algorithm that we call "Extra-Trees", for extremely randomized trees (Geurts et al., 2004). Like Tree Bagging, this algorithm works by building several ( $M$ ) trees. However, contrary to Tree Bagging which uses the standard CART algorithm to derive the trees from a bootstrap sample, in the case of Extra-Trees, each tree is built from the complete original training set. To determine a test at a node, this algorithm selects  $K$  cut-directions at random and for each cut-direction, a cut-point at random. It then computes a score for each of the  $K$  tests and chooses among these  $K$  tests the one that maximizes the score. Again, the algorithm stops splitting a node when the number of elements in this node is less than a parameter  $n_{min}$ . Three parameters are associated to this algorithm: the number  $M$  of trees to build, the number  $K$  of candidate tests at each node and the minimal leaf size  $n_{min}$ . The detailed tree building procedure is given in Appendix A.

- Double learning - Double FQI
- DDPG - DPG
- background: pag 3-4 cinesi

Presented in [16], Deep Q-learning from Demonstrations (DQfD) vastly accelerates DQN by pretraining an initial behavior network, and also introducing a supervised loss and a L2 regularization loss when training the target network. 

A combination of the advantages of both, the speed of the Riccati controller and the generality of MPC, can be achieved by finding a function that maps state values to control variables, e. g., by training a deep neural network. Such a model could, for example, be learned supervised, as done for PILOTNET, or by reinforcement learning. The latter in particular led to excellent results in the training of such agents for controlling real-world systems such as robots or helicopters. Recent work also shows promising applications of reinforcement learning for autonomous driving by making strategic decisions. Autonomous driving tasks where RL could be applied include: controller optimization, path planning and trajectory optimization, motion planning and dynamic path planning, development of high-level driving policies for complex navigation tasks, scenario-based policy learning for highways, intersections, merges and splits, reward learning with inverse reinforcement learning from expert data for intent prediction for traffic actors such as pedestrian, vehicles and finally learning of policies that ensures safety and perform risk estimation. Further, it turns out to be suitable in contexts of autonomous racing: the driverless racer could learn a policy that is able to outperform the performance of a human driver, or a policy taught by experts.   
  


Esempio: All the reinforcement learning algorithms require interaction with the environment to learn a policy from the gathered experience. Trajectories can be collected by using real world environment or through a simulator. Both approaches have pros and cons ... (Tari esempi)  
However, the choice between real environment and a simulated one mainly depends on the context and the application of the learning agent. When exploration is not dangerous real world environment are preferred whereas simulators are better when the real agent can hurt itself or damage the environment.

## 4.2 Related Works

In this section we'll describe some of the applications of Reinforcement Learning to videogames, autonomous driving and racing.

in the context of

### 4.2.1 Reinforcement Learning in Videogames

Due to the flexibility of the reinforcement learning algorithms, which enable an agent to learn a behavior from its experience, by giving a representation of the world a which don't require an apriori knowledge of its dynamics, reinforcement learning is particularly suitable to the contexts which permit to perform many tries of a particular task. For this reason, videogames and computer simulator are adapt to run a reinforcement learning algorithm on. A notorius videogame played by a rl autonomous agent is Atari [4] by Google's Deepmind. By learning a deep convolutional neural network to approximate the Q-function, Mnih et al. successfully construct a Deep Reinforcement Learning framework called Deep Q-Network (DQN) which plays Atari games in human level. It takes as representation of the world the raw pixels and learns ~~and learns~~ an end-to-end policy which maximizes the q-values. Another famous example is Alphago also by Deepmind [5], which beated for several years consecutively the human champion at Go game. This is not a videogame but a board game, but the case is noteworthy, because such game is known for being one of the most complex for his huge set of rules, which makes brute-force approaches infeasible. To achieve this result, it has been trained with a supervised learning neural network from past experience, and then, a reinforcement learning algorithms has been applied to try and beat itself's own play. In racing games, which is the scope of our thesis, different solutions has been proposed over the years. In the next section we'll introduce the concept of autonomous driving and in particular in the racing context. We'll expose the problem of finding the optimal trajectory and explore some of the state-of-the-art-techniques which tackle this issue.

È un po' contorto perché contiene diverse cose.

→ RL

### 4.2.2 Autonomous Driving

Finding a racing line that allows to achieve a competitive lap-time is a key problem in real-world car racing as well as in the development of non-player characters for a commercial racing game. The optimal racing line is defined as the line to follow to achieve the best lap-time possible on a given track with a given car. The optimal racing line is the path that a driver should follow to complete a lap on a given track in the smallest amount of time

Per citare i paper dire ad esempio "M. Botts et al. [2] show ..."  
oppure "Authors in [2] show ..."

possible. As the lap-time depends both on the distance raced and on the average racing speed, finding the optimal racing involves two different sub-problems: racing the shortest distance possible and racing as fast as possible along the track.

Over the years, different attempts to achieve time-optimal racing, evolving together with technology. Besides Reinforcement Learning, controlling a self-driving car can be done with a planner, if the optimal trajectory is a priori computed, or by a controller. In this section, we provide an overview of such techniques. *Botts et al. encoded*

*\{a\}* ← [ This paper [2] shows how to encode a racing line by a set of connected Bézier curves, such that each gene defines a small portion of the racing line. Therefore, the evolution is responsible of the entire design of the racing line. In addition, the authors compare two different methods to evaluate the evolved racing line; the first one is based on testing the evolved racing lines in a racing simulator; the second one consists of estimating the performance of a racing line through a computational model. ] *un po' striminzito secondo me puoi aggiungere qualche dettaglio in più*

*\begin{itemize}* ← [ This work [1] proposes a controller called Ahura for TORCS. TORCS is the state-of-the-art simulator for racing. (Later on we will describe more in detail the architecture.) The controller uses five modules: steer controller, speed controller, opponent manager (it creates a map of opponents around and finds the vacant slot to overtake), dynamic adjuster (friction, bumps), stuck manager (uses the idea proposed in to control the vehicle when it is out of the track or it has stuck somewhere.). There are 23 parameters in Ahura that need to be determined: 1) eight parameters for the steer controller 2) ten parameters for the speed controller 3) five parameters for the opponent manager. They were determined by using the optimization algorithm CMA-ES: *which* it has a good performance in continuous space, works with nonlinear systems, no constraint handling technique is required, and it is appropriate for nonseparable search spaces. *→ riferimento*

Model Predictive Control (MPC) is a multivariable control algorithm that uses an internal dynamic model of the process, a cost function over the receding horizon and an optimization algorithm minimizing the cost function using the control input. Existing advanced control based attempts to minimum-time driving usually provide only offline open-loop solutions. In ~~this paper~~ [6] the authors directly use a one-level approach, solving the nonlinear MPC problem with an economic cost function in real-time. The tight real-time bounds imposed on computational times make it necessary to reformulate the problem so as to allow for the use of efficient algorithms. It provides real-world experimental results of the proposed nonlinear MPC scheme from a miniature race-car setup that is detailed in the paper. *→ riferimento per CMAES*

#### 4.2.2.1 Reinforcement Learning in Autonomous Driving

With the evolving of technologies and algorithms, the focus has been shifting towards reinforcement learning and in general data-driven algorithms, that has been proven to outperform classical algorithms in a variety of applications, including autonomous driving and racing.

Due to the complexity of the problem, the most successful approaches has been proved to be the ones that incorporate an automatic learning process with human expertise.

spiegare perché  
è bene usare  
dimostrazioni di  
esperti

In this work [3] Cardamone, Loiacono and Lanzi applied supervised learning to develop car controllers for The Open Car Race Simulator from the logs collected from other drivers. They considered two representations of the current state of the car: the set of rangefinder inputs usually employed in simulated car racing competitions, and a high-level, qualitative, representation involving basic lookahead information about the track in front of the car. Instead of predicting the typical low-level actions on the car actuators available in TORCS (namely, the steering wheel, the gas pedal, the brake pedal and the gear change), their approach predicts a target speed and the car position with respect to the track axis. They considered two supervised learning methods, multi-layer neural networks and k-nearest neighbor classifiers, and applied them to compute a model mapping input sensors to actions which could imitate the behavior of an observed driver.

\ In this paper [7], the authors addresses an issue which is similar to ours: they aim at integrate a reinforcement learning algorithm with human expertise. Their claim is that this process could help accelerate the exploration and increase learning agent's stability. Their framework is a continuous reinforcement learning setting, in which they integrate DDPG with human demonstrations. To achieve that, they formulate a loss function to optimize which combines the experience gathered from the environment and by the demonstrations. In such a way, with respect to a state, the agent should pick an action wich is consisent with the human behavior.

phase

Our work proposed different approaches to solve time-optimal racing problem in the framework of reinforcement learning. ampliare, parlare di torcs

Nel complesso aggiungerei dei dettagli in più nella spiegazione dei metodi.



## Chapter 5

# Methods





## Chapter 6

# Experiments

TESTO CAPITOLO 7



## Chapter 7

# Conclusion

TESTO CAPITOLO 8



# Bibliography

- [1] M. R. Bonyadi, Z. Michalewicz, S. Nallaperuma, and F. Neumann. Ahura: A heuristic-based racer for the open racing car simulator. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(3):290–304, 2017.
- [2] M. Botta, V. Gautieri, D. Loiacono, and P. L. Lanzi. Evolving the optimal racing line in a high-end racing game. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 108–115, 2012.
- [3] L. Cardamone, D. Loiacono, and P. L. Lanzi. Learning drivers for torcs through imitation using supervised methods. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 148–155, 2009.
- [4] Silver Mnih, Kavukcuoglu. Playing atari with deep reinforcement learning.
- [5] Maddison Silver, Huang. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.
- [6] Robin Verschueren, Stijn Bruyne, Mario Zanon, Janick Frasch, and Moritz Diehl. Towards time-optimal race car driving using nonlinear mpc in real-time. volume 2015, 12 2014.
- [7] Zhu Zuo, Wang. Continuous reinforcement learning from human demonstrations with integrated experience replay for autonomous driving. *International Conference on Robotics and Biometrics*, 2017.