UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dept. of Computer Systems and Computation

Integrating Human Behaviour and Cognition into Autonomous Driving Learning Systems

Master's Thesis

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

AUTHOR: García Bohigues, Miguel

Tutor: Onaindia de la Rivaherrera, Eva

Experimental director: Aso Mollar, Ángel

ACADEMIC YEAR: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

MASTER'S THESIS

Integrating Human Behaviour and Cognition into Autonomous Driving Learning Systems

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Academic Course 2023/2024

Miguel García Bohigues

Advisers:
Dr. Eva Onaindia de la Rivaherrera
Ángel Aso Mollar

Abstract / Resum / Resumen

Abstract

Over the last decade, there has been a great bulk of research on the use of Reinforcement Learning (RL) techniques for intelligent decision-making in autonomous driving. The trial-and-error exploratory nature of RL has proven to be a promising solution for learning driving policies. In this work, we propose a human-RL collaborative approach for learning on-road driving by integrating two sources of knowledge, (i) samples of human behaviour representing the expert knowledge of driving on challenging roads and (ii) a representation of the human field of view that models the agent perception to support upcoming decisions on the scene content. Both knowledge sources are integrated into an RL scheme where the human vision imitation helps the agent interact with the environment by removing irrelevant information and the human driving experience is used to reinforce negative actions. The results will show that incorporating both inputs in an RL algorithm helps anticipate cornering scenarios and avoid going off the lane compared to a baseline without such inputs.

Resum

En l'última dècada, s'ha investigat molt de sobre l'ús de tècniques d'aprenentatge per reforç (RL) aplicades a la conducció autònoma. La naturalesa exploratòria del RL, basada en prova i error, ha demostrat ser una solució prometedora per a l'aprenentatge de polítiques de conducció. En aquest treball, proposem un enfocament col·laboratiu Humà-Màquina aplicat a l'aprenentatge de tècniques de conducció en carretera mitjançant la integració de dues fonts de coneixement, (i) mostres de comportament humà que representen el coneixement expert de la conducció en carreteres complexes i (ii) una representació del camp de visió humà que modela la percepció de l'agent, donant-li suport en futures decisions sobre la rellevància del contingut en l'escena. Totes dues fonts de coneixement s'integren en un esquema de RL en el qual la imitació de la visió humana ajuda a l'agent a interactuar amb l'entorn eliminant la informació irrellevant i l'experiència de condició humana s'utilitza per a reforçar les accions negatives. Els resultats reflecteixen que la incorporació de totes dues tècniques ajuden a anticipar escenaris de corbes i a evitar sortir-se del traçat, en comparació amb un model basi mancat d'aquest coneixement.

Resumen

En la última década, se ha investigado mucho sobre el uso de técnicas de aprendizaje por refuerzo (RL) aplicadas a la conducción autónoma. La naturaleza exploratoria del RL, basada en prueba y error, ha demostrado ser una solución prometedora para el aprendizaje de políticas de conducción. En este trabajo, proponemos un enfoque colaborativo Humano-Máquina aplicado al aprendizaje de técnicas de conducción en carretera mediante la integración de dos fuentes de conocimiento, (i) muestras de comportamiento humano que representan el conocimiento experto de la conducción en carreteras complejas y (ii) una representación del campo de visión humano que modela la percepción del agente, apoyándole en futuras decisiones sobre la relevancia del contenido en la escena. Ambas fuentes de conocimiento se integran en un esquema de RL en el que la imitación de la visión humana ayuda al agente a interactuar con el entorno eliminando la información irrelevante y la experiencia de condición humana se utiliza para reforzar las acciones negativas. Los resultados reflejan que la incorporación de ambas técnicas ayudan a anticipar escenarios de curvas y a evitar salirse del trazado, en comparación con un modelo base carente de dicho conocimiento.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

This work lays the foundation of a different approximation to the Autonomous Driving (AD) paradigm. This memory revisits current the state-of-the-art in terms of driving simulators, Reinforcement Learning (RL) algorithms and similar Autonomous Driving Systems (ADS), and proposes a new methodology to enhance the results and convergence of ADS.

This chapter describes the motivation for this work along with the principal objectives that will be achieved during its execution and as a consequence of it. In addition, an outline of the structure is provided at the end of this section.

1.1 Motivation

Since the the invention of the car around 1886 by Carl Benz[1] until today, driving has had a profound impact on the daily lives of individuals worldwide. The automobile has facilitated the closing of geographical distances between distant cities, the creation of new employment opportunities, and the advancement of personal freedom and independence. In the recent years, with the rise and democratisation of Artificial Intelligence (AI), all industries have been shaken by the capability of AI to serve a variety of emerging technologies [67, 46] excelling in driving related tasks involving perception, control and complex strategic execution [75, 51, 68, 87, 12].

The AD paradigm faces a complex field of application, where safety and robustness against failure must be assured. For this reason ADS usually focus on the resolution of one particular driving scenario. More precisely, ADS have been extensively applied to the development of urban self-driving cars [28, 40] or to the modelling of high-speed management and lane-changing policies in highways [60]. However, it can be seen that there is a lack of research focus on conventional and country roads. Although this type of road incorporates the characteristics of urban driving in roundabouts and crossroads, as well as those of highways in terms of high speed, lane changing and overtaking, it is nevertheless a distinct entity for the additional layer of complexity that

[1]`https://en.wikipedia.org/wiki/Car`

the nature of this roads layout represent. The difficulty of these roads, coupled with the potential for transferring the acquired knowledge to other scenarios, constitutes an area of research that is both fascinating and worthy of further investigation.

Conventional roads also are a particular cause for concern in some countries. Spain provides a useful example in this regard, with conventional roads representing 54.03% of the total kilometers included in the national road network[2]. Furthermore, these roads are associated with a significantly higher mortality rate per accident[3]. Indeed, 5.77% of the accidents result in fatalities, in contrast to highways, where this ratio is 3.52%.

Taking into consideration all the aforementioned, the objective of this Master's thesis is to develop a robust ADS capable of adapting to different road layouts, adjusting its speed to drive safely and comfortably around incoming turns. As will be discussed subsequently, the system will be endowed with human behavioural and cognitive capabilities, thereby enhancing the existing state-of-the-art solutions. Furthermore, a benchmark will be established for the development of ADSs in the context of the simulator used.

1.2 Main objectives

With this work we aim to:

- Explore, understand and evaluate current AD platforms selecting the one that best fits our needs.
- Revisit state-of-the-art AD solutions identifying possible improvements.
- Understand and implement cutting edge RL algorithms, being able to upgrade them by endowing human expertise
- Train and evaluate an ADS capable of driving safely around a series of known and unknown roads.
- Establish a general benchmark in ADS development.

1.3 Document structure

This thesis is structured as follows: First, Chapter 2 will provide an introduction to the general concepts and mathematical background behind the state-of-the-art RL algorithms. This will be followed by a more detailed examination of the AD paradigm in Chapter 3, which will include a review of the actual challenges and existing proposals, as well as a discussion of both simulation platforms and related works. After

[2]https://www.transportes.gob.es/carreteras

[3]With respect to the 2020 official annual reports: https://www.transportes.gob.es/recursos_mfom/comodin/recursos/accidentes_con_victimas.pdf, https://www.transportes.gob.es/recursos_mfom/comodin/recursos/victimas_mortales_en_la_rce.pdf

that, Chapter 4 details our methodology for integrating human behavioural and cognitive aspects into a general RL framework. Chapter 5 will transform the preceding theoretical concepts into an in-depth, detailed implementation. Finally, Chapter 6 will present the experimentation procedure and the results obtained. The conclusions of which will be discussed in Chapter 7.

Chapter 2

Theoretical Framework

In this chapter, we introduce the Reinforcement Learning paradigm, starting from the basics of the field until reaching state-of-the-art algorithms. First, Section 2.1 will introduce the reader to the field of pattern recognition, and establish the main differences between the different kinds of machine learning. Second, Section 2.2 will delve deeper into the key concepts of Reinforcement Learning. It explains some basic definitions and then formally defines each one of the components that compose a Reinforcement Learning problem. Third, in Section 2.3 the reader will find the first approach to solving a problem using this technique. This section will lays the foundation for Section 2.4, which describes further modifications of the mentioned algorithm using Neural Networks as function approximators. Finally, in Section 2.5 the reader will find current state-of-the-art algorithms for control tasks.

2.1 Introduction

The field of Pattern Recognition focuses on the recognition of regularities in data to perform actions such as the classification of data into different categories. [5] The objective of Pattern Recognition is to build a machine that takes a vector x as input and produces a target number representing the class label or the final solution. This process is known as *Machine Learning*, which can be formally expressed as a function $y = f(x)$, where x is the vectorial representation of the input data and $f(x)$ is a function whose shape is determined during the *training phase*, based on the training data.

Machine learning models, as they are called once the training phase has ended, learn certain characteristics inherent to the training data and try to emulate the probability distribution of certain events to happen. The ability to correctly model not only the dynamics of the training set, but also the dynamics of unseen data is known as *generalisation*. Depending on the nature of the training data, machine learning algorithms are categorised into three big families:

- Supervised Learning: works with pairs of samples (x, y) where typically x represents the input feature vector and y represents the target that the algorithm must predict. Depending on the nature of the target, these algorithms can be *Classification algorithms* or *Regression algorithms.* The former aim to associate each input vector with one of a finite number of discrete categories. An example of this kind of algorithm can be the MNIST task [16], where machines must tell which one of the 10 characters (0..9) appear on an image. The later tries to find a continuous variable that corresponds to the given input vector. One example of this kind of learning could be the prediction of the stock value of a certain business based on the previous history.

- Unsupervised Learning: works with data consisting of only a set of input vectors without the corresponding target value. These methods aim to discover groups of similar examples (clusters)[32], determining the distribution of data over the input space (density estimation)[69], or projecting the input data from a high-dimensional source to a target dimension, usually small enough to be printed and used to draw visual conclusions[47].
- Reinforcement Learning: finds suitable actions to perform under a given situation so that the reward obtained by the model is maximised. Instead of pairs of labelled samples, reinforcement learning algorithms learn by trial and error thrrough their interactions with the environment. It has traditionally been applied to games [49], although its use has also resulted in more efficient matrix multiplication algorithms [19].

2.2 Reinforcement Learning: Fundamentals

Reinforcement learning (RL) is a computational approach to understanding and automating goal-directed learning and decision-making [74]. It differs from other approaches in that an agent learns from direct interaction with its environment. A RL system comprises four principal sub-elements: a reward signal, a value function, a policy, and, optionally, a model of the environment.

A *policy* defines how the behavior of the learning agent at any given time, denominated in RL as *state.* The policy can be understood as a mapping function that processes states and outputs actions to be taken at each one of those states. In general, policies are stochastic, although they can be deterministic.

The *reward signal* defines the ultimate objective of an RL problem. At each time, the environment sends a single number, the reward, to the RL agent, which indicates what is beneficial in the short term. Since the rewards received by the agent depend solely on its actions at a given state of the environment, rewards are the defining features of the problem and significantly alter the agent's policy. Should the selected action yield a low reward, the policy may be revised to select alternative actions in the future, to achieve greater rewards. Typically, the reward is a stochastic function defined in terms of the visited state and the action taken.

In contrast, the *value function*, indicates long-term benefits. A value function represents the total amount of reward that an agent expects to accumulate over the future, starting from a given state.

Consequently, value functions are predictions of future rewards that indicate the overall desirability of future states, taking into account the likelihood of future states and the potential rewards available in those future states. In the context of learning a policy, the optimal course of action for an agent is to identify and pursue actions that will result in states of the highest value. These actions represent the overall reward, rather than close reward signals. Unfortunately, the values of states are not known and must be estimated. In order to achieve this, an agent must update its beliefs about state values based on a sequence of observations made over its lifetime.

Finally, a *model of the environment* is a mechanism that emulates the behaviour of the environment (i.e. it predicts transitions between states based on the selected actions). Models are employed to determine the most appropriate course of action in anticipation of future scenarios. RL algorithms that make use of environment models are denominated "model-based" algorithms. On the other hand, systems that explicitly estimate the policy through trial and error without the use of a model are denominated "model-free".

In a formal sense, any RL problem can be defined in terms of a *Markov Decision Process* (MDP). This is a model comprising a set of states (S), a set of actions (A), a reward function (R), a transition function (T) and a distribution over the initial states (ρ). The mathematical representation of this model is given by Equation 2.1. If both S and A are finite sets, we are dealing with a Finite Markov Decision Process.

$$MDP = \langle S, A, R, T, \rho \rangle \tag{2.1}$$

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\} \tag{2.2}$$

Equation 2.2 comprehends the basis of all the dynamics within a MDP. For a given state and action, the equation models the probability of each possible pair of next state and reward. Based on this, we can define all the necessary information about the environment, including the expected rewards (Equation 2.3) and state-transition probabilities(Equation 2.4)

$$r(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r|s, a) \tag{2.3}$$

$$p(s'|s, a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\} = \sum_{r \in R} p(s', r|s, a) \tag{2.4}$$

At time t, an agent selects an action a_t according to a stochastic policy π. π is a mapping function from state $s \in S$ and action $a \in A$, to the probability of taking action a when the agent is in s:

$$\pi : S \to p(A = a_t|S = s_t) \equiv \pi(a_t|s_t) \tag{2.5}$$

The value of a given state s under policy π is defined as the expected reward to be earned as a consequence of following policy π from that state on. This value is known as the *state-value function* and can be computed as follows:

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = a\right] \tag{2.6}$$

Similarly, the *action-value function*, which denotes the value associated with taking action a in state s under policy π, is defined as $q_\pi(a|s)$. This represents the expected return starting from s, taking action a and following policy π thereafter.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{2.7}$$

In both state-value and action-value functions, the discount factor, denoted by the symbol γ, is a real number within the range $\gamma \in [0,1]$, This factor is used to weight rewards in accordance with their proximity to the current state.

Solving an RL task implies finding a policy that achieves the maximum reward over the long run. Formally, a policy π is said to be better than another policy π' if and only if $V_\pi(s) \geq V_{\pi'}(s)\ \forall s \in S$. It can be demonstrated that there exists at least one policy that is at least as good as, or equal to, all other policies. This policy is known as the *optimal policy* and it is denoted by π^*. All policies that are optimal share the same optimal state-value function $v_*(s) = \max_\pi v_\pi(s)$ and action value function $q_*(s,a) = \max_\pi q_\pi(s,a)$.

Unrolling equations 2.6 and 2.7, we can compute the *Bellman Optimality Equation* for both the state-value (2.8) and action-value(2.9) function as follows:

$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \tag{2.8}$$

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a] \tag{2.9}$$

2.3 Q-Learning and Deep Q-Learning

The optimal policy π^* can be obtained through different algorithms; the most basic of all is Q-learning [80]. The Q-learning algorithm defines an iterative process for learning the optimal action-value (Q-value) function, independent of the policy being followed. This algorithm starts with the creation of a table, the *Q-table*, which must comprise as many rows as there are states in the environment and as many columns as there are actions that the agent is capable of performing. The objective is thus to approximate the real $q(s,a)\ \forall s \in S, a \in A$ by interacting with the environment in accordance with an arbitrary policy π. The approximation is calculated by following Equation 2.10. In this equation, the current value is increased if the current estimation of the action-value function for the given action and state is lower than the experienced value, or decreased if the experienced value is otherwise. The parameter α corresponds to the step-size, which is commonly referred to as the *learning rate*.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right] \tag{2.10}$$

It should be noted that in Equation 2.10, Q is no longer the action-value function, but rather the tabular estimator of the optimal action-value function q^*. In [80], it is demonstrated that the estimator Q converges with probability 1 to the optimal action-value function q^*.

For the algorithm to converge, this mechanism must ensure that all states are visited and that all actions are tried at least once, thus it must handle what is commonly known as the *exploration-exploitation dilemma*. One may wish for the agent to select only those actions that result in the greatest possible reward. However, the agent is unable to ascertain which action is optimal unless it has a reliable approximation of q^* (Equation 2.9), which comes by trying several different trajectories. One common approach to address this issue is the *ϵ-greedy* exploration mechanism [14]. In it, the agent select the action that maximises the Q value for the current state ($a \leftarrow \text{argmax}_{a'} Q(s, a')$) with probability $p = 1 - \epsilon$, and a random action with probability $p = \epsilon$. Higher values of ϵ result in more exploration and longer convergence, whilst lower values result in faster convergence in exchange for the risk of ending in a local minimum.

Despite the optimal policy for an RL problem being obtained, Q-learning is severely limited in terms of computing resources. The size of the problem is proportional to the number of states and actions in the environment, which is given by the expression $\mathcal{O}(|S| \times |A|)$. In large problems, this implies that the state and action spaces become increasingly challenging to be fully explored. This is commonly understood as the *curse of dimensionality*.

To overcome this issue, authors in [50] propose a novel agent architecture that combines RL with the function approximation power of Neural Networks(NN). The primary objective of incorporating NN is to construct an internal representation of states that accurately captures their key features. In this manner, states that are similar will exhibit similar characteristics, and as a result, an agent should behave similarly in any of them.

Consequently, an action-value function is defined as $Q(s, a; \theta_i)$, where θ_i represents the parameters of the *Q-Network* at iteration i. To avoid the divergence in the RL algorithm produced by the non-linear function approximator[77], authors propose the inclusion of a *Replay Buffer*. This can be understood as a dataset composed of the agent's experience $e_t = (s_t, a_t, r_t, s_{t+1})$ retrieved along several time steps $D_t = \{e_1, ..., e_t\}$. During the learning phase, samples of experiences will be drawn uniformly at random from the dataset D in order to minimise the loss function represented by Equation 2.11. This is the mean squared error (MSE) between the Bellman Optimality Equation (Eq. 2.9) and the actual reward obtained for performing action a at state s.

$$L_i(\theta_i) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \hat{\theta}_i) - Q(s, a; \theta_i) \right)^2 \right] \tag{2.11}$$

It is important to highlight the distinction between $\hat{\theta}$ and θ. The former corresponds to the parameters of the target Q-network, which are only updated every C steps and are kept fixed between individual updates. The latter corresponds to the parameters of the Q-network at iteration i[1]. This network is the one used to select actions in accordance with the aforementioned ϵ-greedy methodology.

Despite proving to be successful in several applications [27, 49], Deep Q-Learning is only applicable in problems with discrete action spaces. This limitation is evident in equation 2.11, where the computation of the approximate Q value for the next state s' requires the evaluation of the Q values for all the applicable actions. In the case of continuous action spaces, the number of actions is infinite. One potential solution can be to discretise the action space. However, this approach has significant limitations, most notably the aforementioned curse of dimensionality. As the number of actions exponentially grows with the number of degrees of freedom available at the problem, this approach becomes increasingly unfeasible.

2.4 Deep Deterministic Policy Gradient

To address the limitations of DQN with continuous action spaces, in [41] the authors propose the Deep Deterministic Policy Gradient (DDPG) method, an off-policy, model-free *Actor-critic*[35] algorithm that is capable of learning policies directly in high-dimensional and continuous action spaces. The term Actor-Critic refers to the two different functions utilised in this algorithm. The actor function, denoted by $\mu(s;\theta^{\mu})$, specifies the current policy by deterministically mapping the states to a specific action. The critic function, $Q(s, a)$, is learned using the Bellman optimality equation as in the two previous sections.

As DQN, DDPG presents a replay buffer D to address the divergence issues produced by the direct correlation between states when utilising non-linear approximations. The buffer behaves in the same manner as previously described. It has a fixed size and, once full, is cleared by randomly sampling transitions of the form $d_t = (s_t, a_t, r_t, s_{t+1}) \sim D$, which will be used in the learning phase.

However, a major challenge arises during the exploration phase. Under continuous action space dynamics, the ϵ-greedy procedure is no longer applicable, as there does not exist a finite number of actions from which to sample. Instead, a continuous action space requires a different exploration mechanism. The authors propose the implementation of a constructed exploration policy μ' which incorporates noise sampled from a normal distribution.

The shape of the DDPG algorithm (Algorithm 1) with replay buffer is analogous to the DQN in terms of making use of different weights for targets (Q', μ') and sample (Q, μ) networks. DDPG works as follows: Given the previously stated networks, for each one of the training episodes, the algorithm samples the first state via the distribution over initial states. Then, until the episode is over, it selects the best action for s_t according to θ^{μ} and adds to it some random noise to boost exploration. After that, the action is committed to the environment. As a result, the next state and the associated reward are obtained. The algorithm stores the transition in the

[1]The use of two different Q-Networks (policies), one for sampling actions and another for computing the expected future reward, makes Deep Q-Learning(DQN) an “Off-policy” algorithm. In contrast, algorithms that share the same policy for sampling and computing the expected reward are described as “On-policy” algorithms

Algorithm 1 Deep Deterministic Policy Gradient (adapted from [41])

```
Require: Critic network with random weights $Q(s,a;\theta^Q)$, Actor network with
    weights $\mu(s;\theta^\mu)$
 1: Initialize target networks $Q'$ and $\mu'$ with $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
 2: $D \leftarrow \emptyset$
 3: for episode=1, M do
 4:     Initialize random process $\mathcal{N}$ for action exploration
 5:     $s_t \leftarrow \rho(S)$
 6:     for t=1,T do
 7:         $a_t \leftarrow \mu(s_t;\theta^\mu) + \mathcal{N}_t$
 8:         $s_{t+1}, r_t \leftarrow$ perform_action$(a_t, s_t)$
 9:         $D \leftarrow D \cup (\phi_t, a_t, r_t, \phi_{t+1})$
10:         $N \leftarrow (\phi_j, a_j, r_j, \phi_{j+1}) \leftarrow$ random_samples$(D)$
11:         $y_j \leftarrow r_j + \gamma Q'(s_{j+1}, \mu'(s_{j+1};\theta^{\mu'});\theta^{Q'})$
12:         Update the critic by minimizing the loss $L = \frac{1}{N}\sum_j \left(y_j - Q(s_j, a_j;\theta^Q)\right)^2$
13:         Update the actor policy by using the sampled policy gradient:
14:
                $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_j \nabla_a Q(s,a;\theta^Q)|_{s=s_j, a=\mu(s_j)} \nabla_{\theta^\mu}\mu(s;\theta^Q)$$
15:         Update target networks:
                $$\theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^{Q'}$$
                $$\theta^{\mu'} = \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
16:     end for
17: end for
```

replay buffer (D), and proceeds to compute the expected action-value for the current state using the target networks (line 11). The critic weights are updated by MSE over the predicted action-value. The actor weights, on the other hand are updated by the sampled policy gradient, which is the gradient of the action-value function times the gradient of the state-action probability distribution. In other words, how good or bad an action has proved to be, times the probability of performing that action. Finally, the weights of the target networks(Q', μ') are updated in a way that they are forced to slowly track the learned weights (θ^Q, θ^μ) by applying a $\tau \ll 1$. This improves the stability of learning.

2.5 Proximal Policy Optimisation

Policy gradient methods represented a significant advance in the development of RL algorithms. Since DDPG, a serious investigation has been conducted, resulting in the emergence of numerous algorithms, including Soft Actor-Critic (SAC)[25], Advantage

Actor-Critic (A3C)[48] and Trust Region Policy Optimisation (TRPO) [65], among others. The current state-of-the-art in this field is the Proximal Policy Optimisation algorithm [66] (PPO).

PPO provides a more stable, generic and data-efficient algorithm for the resolution of diverse RL tasks, such as Atari games [34], robotic locomotion [21] and autonomous control [22]. The most significant contribution of PPO is the introduction of a `clip` function that constrains the update of the *surrogate objective*. The surrogate objective is a concept that PPO inherits from the TRPO. It is a criterion used to update the policy in terms of the *Advantage function* and the probability ratio of the selected action.

The original surrogate objective is displayed in Equation 2.12. The ratio function ($r_t\theta$), for a given time t, establishes how more or less probable is a_t given the current weights θ with respect to the old ones θ_{old}. The advantage function (A_t) denotes the divergence between the action-value and the state-value for a given action and state $A(s,a) = Q(a|s) - V(s)$. Using basic definitions from Section 2.2 and expanding them, the advantage function can be expressed as: $A(s,a) = r + \gamma V(S') - V(S)$.

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right] \tag{2.12}$$

It is clearly visible that L^{CPI} can prove to be highly unstable, for example in the earlier stages of the training process, where probabilities of actions could change drastically. To overcome this issue, PPO introduces a "clipped" version of this objective. The `clip` function, is a mechanism that sets an upper and lower bound for a given value.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[min(L^{CPI}(\theta), \texttt{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] \tag{2.13}$$

PPO sets as its surrogate objective Equation 2.13 where $L^{CLIP} \approx L^{CPI}$ when the ratio is close to 1. The clip function ensures that the ratio of the actions is always between $[1-\epsilon, 1+\epsilon]$, typically with a value of $\epsilon = 0.2$. The complete descriptions of the different steps comprising the PPO algorithm can be found at Algorithm 2.

From an empirical point of view, PPO exhibits better sample complexity properties, proving that it is an optimal choice for the autonomous driving problem. As will be discussed in the following chapter, the huge state and action dimensionality of this problem urges the need of learning as much as possible from a single interaction with the environment.

Algorithm 2 Proximal Policy Optimisation - Clip (adapted from [66])

```
Require: Initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
1: for episode=1, M do
2:     Collect trajectories $\{D\}_k = \tau_i$ by running policy $\pi(\theta_k)$
3:     Compute rewards-to-go $\hat{R}_t$
4:     Compute the advantage estimates $\hat{A}_t$ based on the current value function $V_{\phi_k}$
5:     Update the actor policy by maximising the PPO-Clip objective:
           $$\theta_k + 1 = \underset{\theta}{\text{argmax}} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}, \texttt{clip}(r_t(s_t), \epsilon, A^{\pi_{\theta_k}})\right)$$
       via stochastic gradient ascent.
6:     Fit value function by MSE:
           $$\phi_{k+1} = \underset{\phi}{\text{argmin}} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2$$
       via stochastic gradient descent
7: end for
```

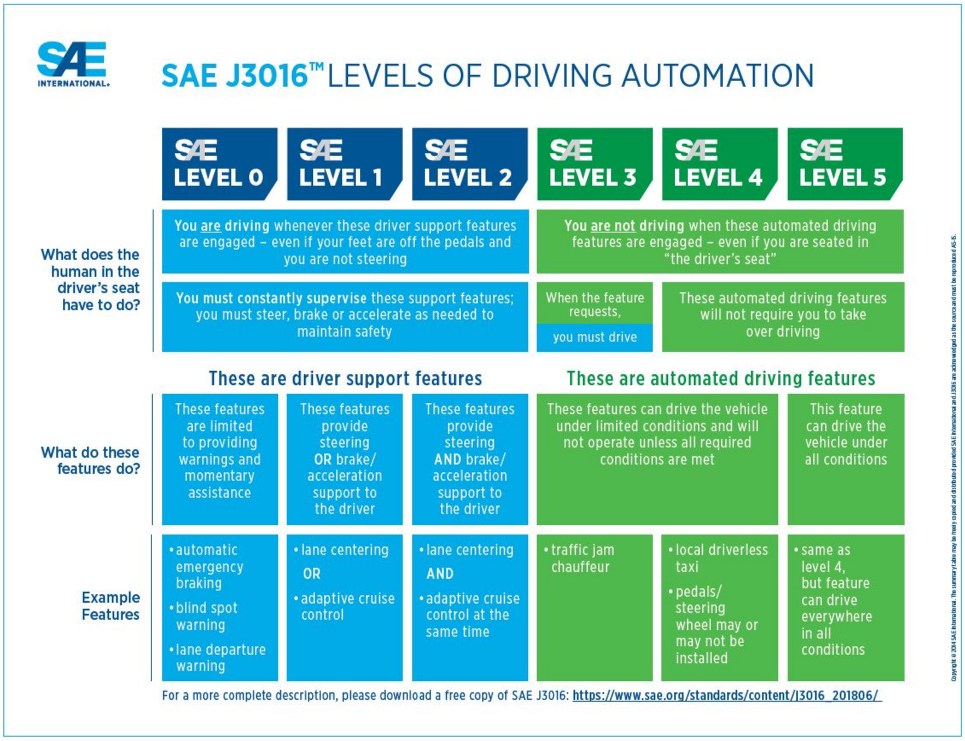
Chapter 3
Autonomous Driving

This chapter provides a comprehensive overview of the Autonomous Driving (AD) domain. It begins by defining the concept of vehicle autonomy and by stating the official levels of autonomy that can be found on street cars. Secondly, it outlines the various processes that comprise the autonomous vehicle pipeline, namely perception, planning, and control. Then, it acknowledges different approaches to the autonomous driving paradigm, starting with the classical mathematical perception and control models, continuing with the initial incorporation of neural networks, and finally describing end-to-end models that fusion all the components into a single entity.

3.1 Introduction to autonomous driving

The field of AD is a research area with the primary objective of developing and implementing fully autonomous self-driving vehicles that are capable of operating with minimal or no human intervention. In order to classify these vehicles according to their level of autonomy, the Society of Automotive Engineers (SAE) has established six distinct categories, which define the respective responsibilities of humans and machines and provide a standard for the automotive market. Figure 3.1 illustrates the different levels of autonomy, along with the extent of human responsibility in a selection of driving scenarios.

SAE level 0 represents the initial stage of the development of such systems, with only a limited number of sensors that are capable of computation. For example, these sensors are used to determine the safety distance between the vehicle and the car in front or to determine whether the vehicle is leaving the lane. SAE level 1 represents a further development, with the introduction of automated functions such as lane centring or cruise control. These functions require constant human supervision, as they only take partial control of the car. SAE levels 2 and 3 represent a threshold between autonomy and assistance. Both levels share the same capabilities, but at level 2, the human must be supervising the vehicle constantly, while at level 3, the vehicle is designed to drive autonomously under certain circumstances. If the conditions for autonomous driving are not met, the driver must resume control of the car. Finally,



SAE J3016™ LEVELS OF DRIVING AUTOMATION

	SAE LEVEL 0	SAE LEVEL 1	SAE LEVEL 2	SAE LEVEL 3	SAE LEVEL 4	SAE LEVEL 5
What does the human in the driver's seat have to do?	**You are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			**You are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			These are automated driving features		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering OR brake/ acceleration support to the driver	These features provide steering AND brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering OR • adaptive cruise control	• lane centering AND • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

For a more complete description, please download a free copy of SAE J3016: https://www.sae.org/standards/content/J3016_201806/

Figure 3.1: Levels of driving automation [55]

levels 4 and 5 represent the maximum level of autonomy. The number of vehicles equipped with this technology is limited, with only a few companies offering level 4 vehicles as taxicabs or merchant trucks. To date, no manufacturer has yet released a level 5 vehicle for general use on the road.

At this stage, it seems appropriate to pose the following question: if a car is indeed capable of adjusting its speed and location with respect to the lane dynamically, why is it so challenging to fully automate the driving task? The reason behind this situation is the intrinsic complexity of driving. Contrary to popular belief, the task is not as straightforward as it may appear. As illustrated in Figure 3.2, autonomous driving vehicles comprise a series of interconnected tasks and processes that must collectively function and coordinate in order to achieve SAE level 5.

Firstly, autonomous vehicles(AV) require a substantial amount of sensory data that accurately describes their surroundings and their current speed, as well as their position within the lane. Typically, AV are equipped with Global Positioning Systems (GPS) devices that provide precise information about the vehicle's location to a precision of a few decimetres. In order to obtain information about the vehicle's surroundings, AV incorporate two additional sensors. First, the RADAR provides information regarding the position and velocity of surrounding objects [30]. Secondly, the LiDAR[13] describes the shape and dimensions of the surrounding objects.

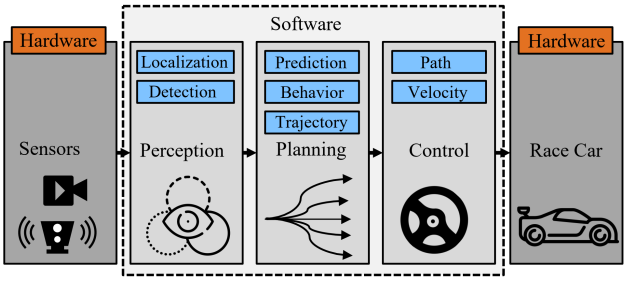


Figure 3.2: Autonomous driving pipeline [4]

All the information retrieved by the sensors must be perceived and processed in order to be used by the subsequent components. The perception module is composed of algorithms which analyse the incoming information, identifying objects and areas of free space. Furthermore, it determines the vehicle's location in relation to its surroundings. This process is crucial, as an error in perception could potentially result in a chain effect, leading to a potential accident. For example if a bridge over the highway is erroneously identified as a wall, the car could decide to apply the brakes in order to avoid the perceived "obstacle".

With regard to the sensory aspects, there is an underlying issue with the nature of the autonomous driving paradigm: the speed. The information retrieved from the sensors must be equally valid when the vehicle is travelling at low speeds in urban scenarios and at high speeds on highways or motorways. Consequently, the latency and precision of these components must be related to the desired maximum speed of the vehicle and the sensing range [18].

The next step in the AV pipeline is comprised in the planning box. This component of the pipeline provides the optimal path or trajectory to be followed. The planning step is usually divides into three distinct phases: global planning, local planning, and behavioural planning. Global planning refers to the trajectory that the vehicle must follow so the car arrives to a destination. An illustrative example of this plan could be the route proposed by a navigation system, such as the Google Maps application. In contrast, local planning offers greater granularity, as it operates under the umbrella of an optimal global trajectory. Its goal is to avoid obstacles and perform overtakes while being as close as possible to the optimal driving line. Finally, behavioural planning provides information about a higher-level mission, such as energy management, adaptation to climatic conditions, interaction with other vehicles, and so on.

The final software component of the autonomous car pipeline is the control block. which reduces the lateral and heading error in order to be close to the reference line (trajectory) and to reduce the velocity error adapting itself to the nature of the road and the current scenario. Control actions typically include steering, braking and accelerating, although in real-world scenarios there are additional tasks such as activating the turning light. Historically, control actions were modelled using complex mathematical and physical concepts that adapted the car velocity and steering taking into consideration the shape of the road [37, 36, 38]. Over time, these techniques were substituted by iterative learning control methodologies. Which became useful with the development of neural networks, since they rely on principles of learning from experience (i.e. repeating the same scenario over and over) in order to learn how to correctly control the car under different circumstances [8, 61, 63, 62].

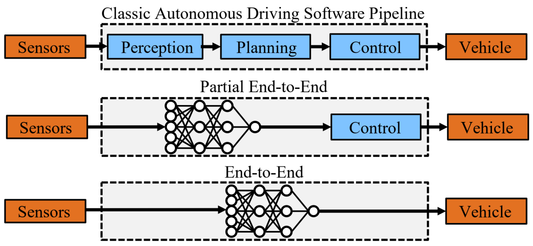


Figure 3.3: Different autonomous driving pipelines [4]

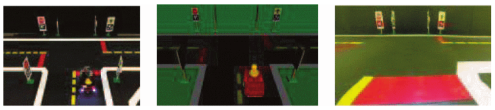
As Figure 3.3 illustrates, there has been a significant research in the autonomous driving field with the goal of avoiding the propagation of errors from one block to its successors. The underlying concept is to combine the components of the pipeline and train a neural network that excels in multiple or all of the blocks of the pipeline, capable of detecting correlations between detection, planning and control activities. A first approach was conducted using a single neural network to detect and plan the optimal trajectory, which will later be used as a reference for a separate control module (referred to as *partial End-to-End* systems)[82, 81, 83].

Finally, as RL algorithms were improved, researchers started to abandon the idea of a modular autonomous car and started to train a RL agent capable of driving in an *End-to-End* manner. In these approaches, the agent learns the driving task without the need of expert knowledge representation, receiving only sensory information [11, 23, 71]. Although this methodology produces, in quick time, a reliable and fast vehicle manoeuvre on different road conditions, it usually fails to generalise to different layouts [15].

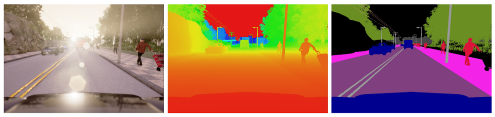
The objective of this thesis is to address the issue of generalisation by incorporating human behaviour and cognition into the *End-to-End* autonomous driving pipeline configuration.

3.2 Simulators

As mentioned in the previous section, the Autonomous Driving Systems (ADS) paradigm is too broad to address all the possible casuistics simultaneously. Instead, research efforts are often focused on specific driving scenarios, as for example urban driving, higher-level behaviour management, and even car racing. Given the high cost of real-world testing, mainly due to the cost of repairs in the event of a crash, researchers



(a) left physical framework, centre digital twin, right on-board camera



(b) left realistic view, centre enhanced depth sensor, right semantic segmentation

Figure 3.4: Comparison of the Duckietown (a) and CARLA (b) simulators

use realistic simulators of the above mentioned environments in order to develop and train their algorithms while minimising the cost of error.

Urban simulators are the most common and have attracted the most attention from researchers in these days [10, 43, 45, 86, 33]. The two most popular simulators are DuckieTown and CARLA. DuckieTown [57] is an open source platform developed for education and research in the field of ADS. The platform is lightweight and flexible, providing a wide range of configurable urban scenarios with minimal computational cost. The system is composed of autonomous vehicles, denominated Duckiebots, that drive through different cities (Duckietowns). Despite their relatively simple design, the Duckiebots capable of localising themselves within a global map, avoiding obstacles, pedestrians (Duckies) and other Duckiebots, as well as following traffic signals. In contrast, CARLA [17] represents a more realistic and computationally expensive urban simulator. The software has been developed using the Unreal Engine 4[20] physics and game-design engine, which provides realistic assets including streets, cars, pedestrians, illumination, climatic conditions, and so on. The platform supports a flexible specification of sensor suites with different signals available as for example GPS, speed, acceleration and detailed data collision. CARLA is also capable of detecting driving infractions like driving on the opposite lane or on the sidewalk.

Focusing on higher-level simulators, highway-env [39] is a simulator based on the OpenAI Gym [7] framework defined to model different driving scenarios such as car overtaking and lane merging on a highway; roundabout collision avoidance, lance changing and path following; a goal oriented car parking scenario; and intersection negotiation with dense traffic. This simulator presents all these environments in a two-dimensional view of the world with a top-down perspective. In this instance, the car controlled by the algorithm is highlighted in green, and the cars that correspond to the environment are coloured in blue. Highway-env has been used to ease the learn of decision-making and motion planning [76], as well as agent coordination [58]. The official GitHub page [1] contains a variety of illustrations demonstrating the

[1]`https://github.com/Farama-Foundation/HighwayEnv`

aforementioned examples.

Finally, there is the family of autonomous racing simulators. These kinds of platforms allow the researchers to push the driving capabilities of autonomous agents to the limit, by including high-speed sensory detection and complex physics dynamics. One example of this kind of system is the Learn-to-Race(L2R) simulator [26] which includes a framework that accurately models vehicle dynamics and racing conditions whilst providing a close-to-real visual representation. L2R provides multi-modal information, with several on-board cameras capturing data from different angles and a comprehensive array of inertial measurement sensors. Another example is F1Tenth [54]. As with DuckieTown, F1Tenth provides both a hardware and a software platform for the development and testing of ADSs. The term F1Tenth is derived from the 1/10th scale used to develop the vehicles. F1Tenth enables a safe and rapid experimentation of algorithms in real-world laboratory set-ups, and in ROS-based [73] virtual environments. F1Tenth is a widely used simulator that holds competitions all around the world where researchers can compare different perception, planning and control algorithms in real-world races[2]. It also serves as a baseline infrastructure to build more complex autonomous racing vehicles such as the Autonomous Indycar series[3]. The last racing simulator to be acknowledged is TORCS [85], the Open Racing Car Simulator. TORCS is a lightweight yet realistic simulator that has long been a benchmark in the field of ADS. As it has been selected as the simulator to be used in this work, it will be explained in detail in the following section.

3.3 TORCS

TORCS is an open-source racing car simulator that provides a three-dimensional visualisation, accompanied by a detailed physics engine and accurate car dynamics, including aerodynamics, traction, fuel consumption, and several other factors. The version of TORCS that is used in this work, corresponds to a proposed modification [44] which is structured as a client-server application, where bots run as an external process which is connected to the simulator via UPD[4] connections. This detached structure allows researchers to create bots in many different languages. The only limitation when it comes to communicating with TORCS is the synchronisation. Every 20ms the simulator sends to the bots the information gathered by the sensors. The simulator then waits 10ms for a response with the next action to be taken. In case that the bot does not reply within the stated time span, the last communicated action is executed. TORCS' communication architecture is displayed at figure 3.5.

TORCS offers a variety of driving scenarios, represented in different categories of race tracks. Firstly, it offers high-speed oval tracks in the style of the NASCAR[5] or IndyCar[6] competitions which can be used to emulate the behaviour of the car when driving at high speeds on wide roads with open and long turns. Secondly, a series of

[2]https://f1tenth.org/race.html
[3]https://www.indyautonomouschallenge.com/
[4]https://en.wikipedia.org/wiki/User_Datagram_Protocol
[5]https://www.nascar.com/
[6]https://www.indycar.com/

road tracks belonging to different environments are available. These include urban streets with multiple acceleration and braking zones, with the usual 90º turns; country roads which resemble the layout of national roads that connect towns or small cities, typically with a mid-range maximum authorised velocity and several chained left and right turns; and mountain roads with their characteristic low-speed turns and changes of slope. Finally, TORCS also contains different off-road tracks where the asphalt is covered in mud, dirt or snow. Such layouts facilitate the creation of scenarios that are specific to driving in different climatic conditions. Figure 3.6 shows the layout of each one of the tracks available in TORCS.

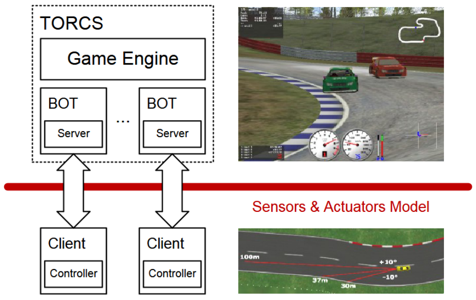


Figure 3.5: TORCS client-server architecture

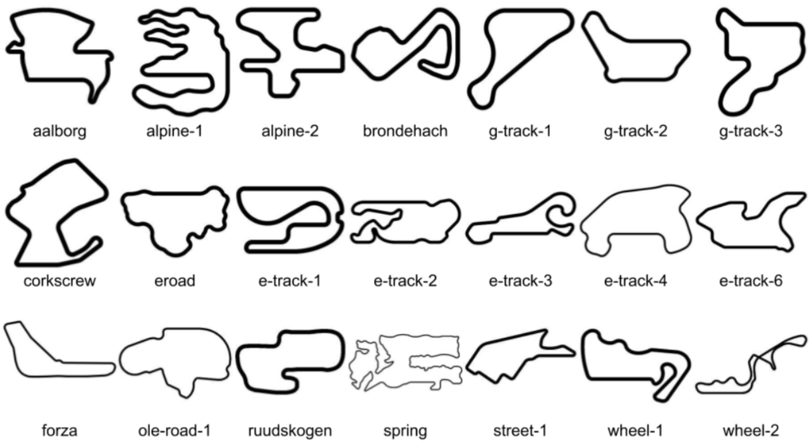


Figure 3.6: Layout of all tracks in TORCS [64]

With regard to the sensory aspect, TORCS provides each registered bot with the information read from each of the sensors detailed in Table 3.1, which details the value range and unit of measure for each sensor.

For further clarification regarding the recovered information, refer to the following

Name	**Range**	**Unit**
angle	$[-\pi, +\pi]$	(rad)
curLapTime	$[0, +\infty)$	(s)
damage	$[0, +\infty)$	(pts)
distFromStart	$[0, +\infty)$	(m)
distRaced	$[0, +\infty)$	(m)
focus	$[0, 200]$	(m)
fuel	$[0, +\infty)$	(l)
gear	$\{-1, 0, 1, .., 7\}$	-
lastLapTime	$[0, +\infty)$	(s)
opponents	$[0, 200]$	(m)
racePos	$\{1, 2, ...N\}$	-
rpm	$[0, +\infty)$	(rpm)
speedX	$(-\infty, +\infty)$	(km/h)
speedY	$(-\infty, +\infty)$	(km/h)
speedZ	$(-\infty, +\infty)$	(km/h)
track	$[0, 200]$	(m)
trackPos	$(-\infty, +\infty)$	-
wheelSpinVel	$[0, +\infty)$	(rad/s)
z	$(-\infty, +\infty)$	(m)

Table 3.1: TORCS sensor names and ranges with units of measure.

definitions:

- *angle*: Represents the direction the cat is facing with respect to the track axis.
- *curLapTime*: Is defined as the time elapsed since the last pass through the goal line.
- *damage*: Is a value that reflects the current flaws of the car produced by collisions with walls or other cars.
- *distFromStart*: Represents the approximate distance driven by the car measured from the finish line.
- *distRaced*: Represents the total distance covered by the vehicle since the beginning of the drive.
- *focus*: Is a vector or five range finder sensors, that act as a LiDAR, returning the distance between the track edges and the car within a range of 200 metres. The orientation of the range finder sensors is fully configurable. It should be noted that when the vehicle is outside the track, the obtained values are not reliable. Due to computational limitations, the values are only updated at a rate of once per second. Consequently, repeated calls within this timeframe may result in the delivery of potentially erroneous information.

- *fuel*: Is the current load of the gas tank.
- *gear*: Is the current gear that the car is driving on.
- *lastLapTime*: Corresponds to the time required by the vehicle to complete the previous lap.
- *opponents*: Is a set of 36 sensors that cover the entire surroundings of the car within a distance of 200m. Each of the sensors has a field of view of 10 degrees, and it is able to report the distance of the closest opponent within that field of view. In the event that no opponent is identified, the value 200 is returned.
- *racePos*: Represents the current position of the bot in the overall race order.
- *rpm*: reflects the current engine's revolutions.
- *speedX*, *speedY* and *speedZ*: Are the decomposition of the car's speed along each one of the three axes (longitudinal, transverse and vertical).
- *track*: Is composed by the same kind of range finder sensors than the *focus*. In this case, there are 19 sensors the definition of which is limited to the front area of the vehicle, specifically between -90º and +90º with respect to the car's visual centerline.
- *trackPos*: Represents the relative distance of the car with respect to the track centre-point. The value returned is 0 when the car is near the centre of the track. Values close to -1 indicate that the car is at the right side, while values close to 1 indicate otherwise. For values outside this range, it can be inferred that the car has left the drivable area.
- *wheelSpinVel*: Represents the current rotatory speed of the vehicle's wheels. The sensor response is a vector with four items.
- *z*: Corresponds to the distance of the car's centre of masses from the surface of the track along the vertical axis (z).

As a response to the input provided, bots are capable of performing up to seven different actions (Table 3.2), apart or simultaneously. The actions *accel*, *brake* and *clutch* correspond to virtual pedals where a value of 0 means no action and a value of 1 represents full pressing of the pedals. As these are continuous actions, they take into consideration all possible real values within the range $[0, 1]$. The *gear* action indicates the next gear to be set, while the *steering* action sets the turning wheel to the given value. Here, a positive value represents turning left, while a negative value means turning right. Finally, the configuration actions *focus* and *meta* are responsible for setting the angle of the five focus sensors and restarting the race, respectively.

TORCS has long been regarded as a leading platform for the development of control and planning algorithms for autonomous racing cars. Its popularity can be attributed to its lightweight characteristics and the variety of driving tracks and cars available. The following section presents a selection of the most significant contributions to control using Reinforcement Learning (RL) for autonomous driving in TORCS, offering a concise overview of the current state of the art.

Name	Range
accel	$[0, 1]$
brake	$[0, 1]$
clutch	$[0, 1]$
gear	$\{-1, 0, 1, .., 7\}$
steering	$[-1, 1]$
focus	[-90,90]
meta	$\{0, 1\}$

Table 3.2: Actions available in TORCS with their value range

3.4 Autonomous Driving in TORCS

TORCS has been employed as a testing ground for a multitude of control solutions for ADS. Its longevity supposes the existence of two distinct control paradigms: a traditional control approach based on mathematical models and a modern approach based on function approximations using neural networks. This section covers the most important representatives of each approach, outlining the methodologies employed, the results obtained and the margins for improvement.

3.4.1 Model-based systems

Prior to the computational feasibility of end-to-end neural networks, there was a significant effort in the autonomous driving research community to develop robust and generalised car driving controllers using mostly mathematical representation of turn curvature, kinematics, and physical models. In [52], authors propose a methodology for the internal representation of the track model using the position and distance covered by the four wheels. At each time t, the model computes the curvature of the road by dividing the distance between wheels (1.94 metres) by the proportion of the distance covered with the left wheels L_l with respect to the right wheels L_r (Equation 3.1). The wheels covering similar distances implies that the vehicle is driving in a straight line, so the value of the radius is effectively infinite.

$$r_t = \frac{1.94}{1 - \frac{L_l}{L_r}} \tag{3.1}$$

Despite the models' proximity to the actual representation of the track maps, the generated models had two major drawbacks, as illustrated in Figure 3.7. Firstly, there is the underlying assumption that exists a basic controller, sufficiently sophisticated to complete a lap on the track, and thus, generate the map. As evidenced by the second example (corresponding to the Aalborg track), this is not always achievable. Secondly, the curvature of the map may be approximate, but the starting and ending points may differ, as illustrated by the third case (E-road).

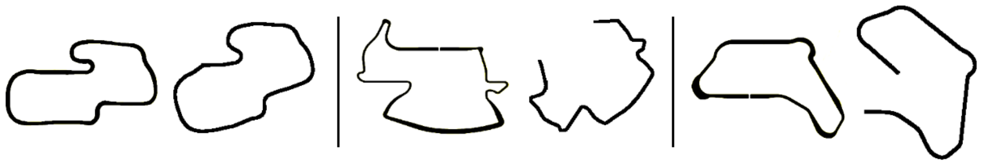


Figure 3.7: Mathematical map generation for tracks Ruudskogen (left), Aalborg (center) and E-road(right)

The authors compared the results obtained from the trajectory and control algorithms trained using the aforementioned mechanism with real data obtained from humans. As table 3.3 illustrates, the resulting algorithm not only demonstrated inferior performance compared to humans, but also resulted in greater vehicle damage over the course of the test.

	Avg. Lap (s)		**Damages**	
Track	**Human**	**Bot**	**Human**	**Bot**
Ruudskogen	74.23	100.89	17	1183
Aalborg	84.4	120.1	17	10090
E-road	73.71	123.09	0	6344

Table 3.3: Human-bot comparison for the controller developed in [52]

Despite these results, the researchers identified great potential in the concept of developing an internal representation of the current track. Consequently, several articles were published in which the authors attempted to overcome the limitations previously exposed by (i) dynamically computing the curvature of the turns in front as the agent drives, as described in [6, 59], and by (ii) using directly the information coming from the sensors and applying Fuzzy Logic to discretise the scenarios, as in [56, 9]. A direct comparison between these models and human performance can be observed int Figure 3.8. For the first time, a performance surpassing that of a human has been achieved on a track (C-Speedway). The results of the track tests demonstrate that the curvature prediction algorithms have the potential to perform at a level close to a human operator.

3.4.2 Neural-Based systems

Given the complexity of mathematically modelling the dynamics of the car and the margin for improvement observed in the previous section for some of the models, there has been a trend of gradually abandoning the mathematical formulation and adopting end-to-end approaches that directly map observations into actions, using a neural network within a RL framework.

In [24] the authors present a comparison of the performance of two different Deep

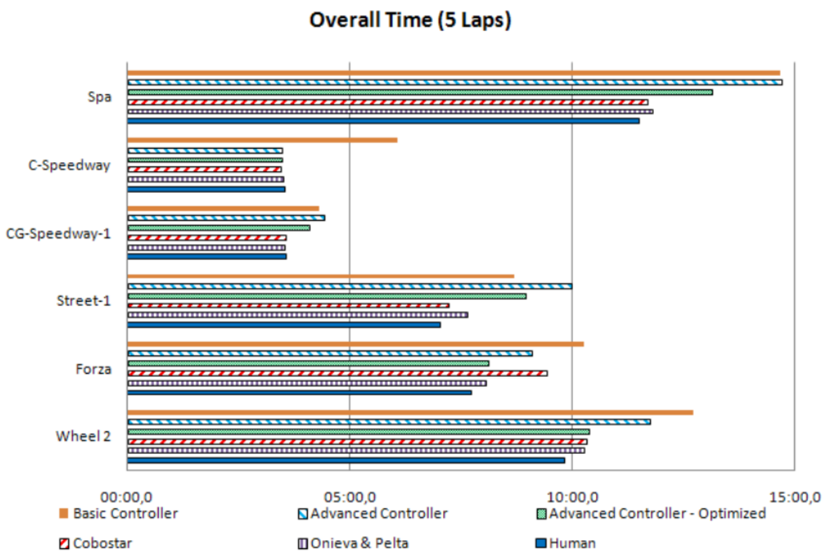


Figure 3.8: Comparison between models: Onieva & Peralta ([56]), Cobostar([9]) and authors' controllers ([59])

Reinforcement Learning (DRL) algorithms . The Soft Actor-Critic (SAC) algorithm is subjected to a comparative analysis against a variant of the Deep Q-learning algorithm explained in Section 2.3. This variant has been adapted to handle continuous action spaces by applying a categorisation process in the output [3].

Both RL agents are trained using the same input information. Of all the sensory available in TORCS, the authors decide to use only the *angle* between the car and the road, the *track* "LiDAR", the position of the car relative to the width of the road, the *speed* decomposed in each one of the axes, the rotational speed of the four wheels and finally the *RPM* of the engine.

The reward function has been defined in terms of a multi-component function (Equation 3.2). In this sense, the agent is positively rewarded for achieving high speeds while following the track direction, and it is penalised for driving towards the edges of the road or for applying excessive lateral force.

$$R = V_x \cos\theta - |V_x \sin\theta| - |2V_x \sin\theta \; trackPos| - V_y \cos\theta \tag{3.2}$$

The symbols V_x and V_y represent the longitudinal and lateral velocity, respectively. The angle between the car and the track axis is denoted by θ. Finally, the normalised distance between the centre of the track and the car is referred to as *trackPos*.

The training phase has been conducted utilising all the available tracks, with the objective of achieving track generalisation. The tracks are cyclical in nature, with the sequence of tracks changing after five training episodes. Once the final track has been completed, the training process is resumed with the initial track.

As an exploration mechanism, the article proposes the maximisation of the entropy for the SAC, and the application of some noise to the output of the DQN. However, there is one specific action that the authors acknowledge to need for a dedicated exploration procedure. For this case, a stochastic braking mechanism has been employed, using a gaussian distribution to determine when the actual actions should be overridden and use the brake instead. With this, the aim is for the agent to learn the braking zones to reduce speed before entering the turns.

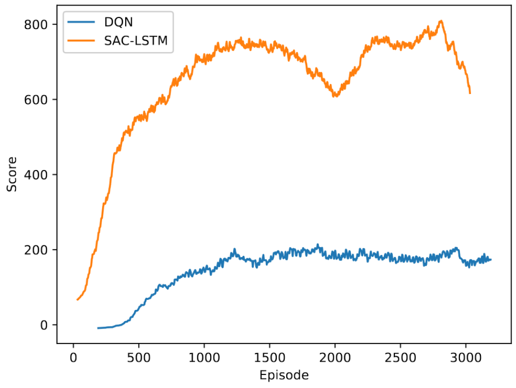


Figure 3.9: Reward comparison between SAC and DQN

Although the authors do not specify the track or combination of tracks where the evaluation has been performed, it is evident that there is a clear trend indicating that the SAC algorithm outperforms the DQN. However, the only results presented are dependent on the reward function. This makes them incomparable with other systems that solve the same problem, but with a different reward function. Nevertheless, the article includes a video[7] demonstrating a qualitative comparison of the performance of both algorithms. It can be observed that the car is continuously making right and left turns, not being able to drive in a straight line. Notwithstanding, the vehicle appears to be stable when turning, which provides insight into the reason for this slaloming in straight lines. This may be to detect turns in advance and always be prepared to take them.

Following the same idea, in these two other works [78, 29], the authors employ the DDPG algorithm (Section 2.4) to train a different ADS. Both works make use of the same four sensors: *angle*, *track* LiDAR, *trackPos*, and the *Speed* decomposed into the three axes. In [78], the authors vie for implementing a more complex weighted reward function (Equation 3.3), whereas in [29] authors decide to use a simpler representation of the same function composed of only three components (Equation 3.4)

[7]`https://youtu.be/f82EBvPKyDI`

$$R = V_x \cos(\theta) - \alpha V_x \sin(\theta) - \gamma |trackPos| - \beta V_x |trackPos| \tag{3.3}$$

$$R = V_x \cos(\theta) - |V_x \sin(\theta)| - |V_x trackPos| \tag{3.4}$$

Each one of the systems has been trained in different tracks. In [78] the authors make use of only the Aalborg track. The agent is first trained alone, to learn the dynamics and the layout of the track, and then it is tested in a competition on the same track with other 9 TORCS default bots. In [29], on the other hand, the authors select the CG-SpeedWay track for training the agent and use Track-E and Highway-Track to validate the resulting behaviour.

As with the previously commented work, there is no standard evaluation process for these other two. The results depend on the evaluation tracks and the reward function, which are different. For this reason, the performance of the models cannot be compared. However, as the authors of [78] provide a video of the evaluation phase[8], it is possible to draw certain conclusions. The record shows less critical zigzag manoeuvres at straight lines, which reinforces the previous assumption that this behaviour is caused by a forceful anticipation of turns. As this agent has only been trained in one track, the anticipation becomes less important. It is also visible, at the right-bottom axis, that the agent is not braking at all. It manages the speed by acting only with the accelerator pedal, which is clearly a sub-optimal behaviour.

In our work, we address the problems identified in the aforementioned works. Firstly, we define a new RL schema for the Actor-Critic algorithm that incorporates human behaviour and cognition helping the agent to reduce the action search space by enhancing the information of the critic. Secondly, we develop a generalised driving model able to adapt, not only to unseen tracks but also to cars that have not been driven before. Finally, we provide a standard and reward independent measure that represents the performance of our model. Establishing a comparable benchmark for the development of ADS in TORCS.

[8]https://www.dropbox.com/s/balm1vlajjf50p6/drive4.mov?e=1&dl=0

Chapter 4

Human-Centred Autonomous Driving

This chapter details the main goal of this thesis, that is including human expertise in terms of behaviour and cognition within the actor-critic (AC) schema. It begins by introducing the concept of Inverse Reinforcement Learning, explaining the methodology and some issued derived from its use. After that, our proposal to enhance the AC schema is detailed, justifying the lack of huge biases in the agent behaviour and its better performances compared with other expert-based supervision systems. Then, the need to emulate human perception is introduced, explaining the human perception process along with key concepts as the *persistence of view.*

4.1 Integrating Human Behaviour

As it has been demonstrated in the previous chapter, Actor-Critic (AC) algorithms [35] have become state-of-the-art when approaching an Autonomous Driving System (ADS) problem modelled under the Reinforcement Learning (RL) paradigm. Their exceptional performance is attributed to the close relationship and constant feedback between the two components. Firstly, the actor retrieves a collection of sequences of states, actions and rewards (usually referred as trajectories) by making use of its learned policy, which determines the optimal action based on the critic's evaluation of the reward to be obtained in future states. Secondly, the critic processes the obtained trajectories and refines its evaluation of the future rewards, based on the actor retrieved experience, to offer more accurate predictions in the next iteration.

It is evident that, under this paradigm, both component strongly rely on each other. While the actor needs accurate predictions from the critic in order to perform informed decisions regarding the most appropriate course of action. The critic requires a diverse and informative trajectories to better predict the future and thus, force the actor to improve. If one of the two components fails in their task, the other is strongly affected. In the event that the actor fails to adequately explore the state space, the critic is unable to provide accurate information regarding other regions of the state space with a higher value. Otherwise, if the critic is unable to accurately predict the expected reward in future states, the actor may end up in an undesired state, which could ultimately lead to a deterioration in the policy.

As observed in Section 3.4.2, the exploratory nature of AC algorithms is not enough by itself. In the case of ADS, where the state space is vast, some actions may not be explored due to the formulation of the problem, unless a specific exploration mechanism is developed. In [78], for instance, the authors do not provide a specific exploratory factor, which results in the agent never learning to brake. This is because the definition of the problem inherently rewards movement (in the form of higher speeds) and braking is in direct opposition to this. This results in the agent reaching a conservatory speed that allows it to navigate corners by only lifting the acceleration pedal. Instead of reaching high speeds, braking to take turns, and then reaching high speeds again, which would be an optimal behaviour. In [24] braking is learned by adding a special *ad-hoc* mechanism. By cleverly forcing the agent to brake 10% of the time, the authors ensure that the agent learns to brake efficiently before entering the corners. Nevertheless, despite managing the speed correctly, this agent fails to drive smoothly in straight lines, as it is constantly making left and right turns.

To be still with the steering wheel while speeding up in a straight line is a phenomenon that humans are aware of, but which machines must learn to emulate. This behaviour is particularly challenging to address, as agents learn through trial and error based on a reward function that must remain as simple as possible in order to preserve the agent's freedom for exploration. Consequently, penalising steering will inevitably affect the agent's ability to take turns. This behaviour, instead of being an exploration issue, it is an issue of knowledge transfer. There is a dearth of implicit and explicit information that could assist the agent in learning this and many other human behaviours.

To address this issue and at the same time increase the convergence of RL agents, there has been a great deal of interest in including human behaviour and experience within the RL paradigm. This is referred to as Inverse Reinforcement Learning (IRL) [1] and defines a Markov Decision Process (MDP) that does not define an explicit reward function. In contrast to the conventional approach of learning the reward function from examples provided by the agent as a consequence of its interaction with the environment, IRL learns directly the reward function by processing samples that are generated by experts on the task at hand. This represents a shift in paradigm from the traditional approach of “learning from interaction” to “learning from demonstrations”. Consequently, IRL generates a policy that emulates the expert's behaviour by identifying the relationship between the states visited and the actions taken.

In the context of ADS, the approach has been used to initialise the behaviour policy (actor) with an offline dataset composed by samples produced by human interaction with the simulator [70]. In this manner, the agent is able to learn a robust policy without having to begin from scratch. Initially, the agent learns a policy via IRL, and then, through the incorporation of a limited exploration mechanism into the AC schema, it is able to further refine its policy. Due to the properties of the MDP, it can be ensured that the policy resulting from this additional training phase will be equally good or better than the initial policy.

Despite the benefits that using an initialised policy provides in terms of convergence, it also entails a massive impact on the agent behaviour. As the exploration factor applied to the posterior RL training is reduced drastically, the behaviour of the agent is highly biased by the samples provided in its initialisation. If this samples contain undesirable practices that should be avoided, there is a possibility that the agent may be unable to address this in its policy.

In response to this problematic issue, a more recent study [84] uses a different methodology, which established a constant supervision of the agent's actions by a human expert. The system is designed in such a way that human supervisors are allowed to override the actor's actions with alternative ones of their choice in order to avoid undesired scenarios. By doing this, the agent learns from its interaction with the world in a natural manner, without being heavily influenced by human behaviour and, at the same time, assuring that certain behaviours are avoided. However, this process is very costly. First, in terms of financial resources, as it needs the presence of an expert throughout the whole training process. And second, in terms of time, as the expert must be able to process what is happening in the simulator in order to act and avoid certain situations. Therefore, the time of simulation must be equal to the real time.

This thesis presents a novel approach to the actor-critic method that aims to combine the benefits of IRL and human supervision. As the critic forms its predictions based on the samples produced by the actor, the critic relies on the actor's exploration capabilities to provide precise estimations about the future rewards. Therefore, the more heterogeneous the trajectories are, the more accurate the predictions will be.

As previously discussed, a common approach to promote trajectory diversity is boosting the exploration capabilities of the actor, which has been shown to yield good results for most problems. However, in tasks where the consequences of previous actions have a significant impact on future states, *ad hoc* procedures may generate poorly informed random samples, derived from early or inaccurate actions. The driving task represents a good example of this kind of task. Braking one meter earlier or later can result in many different outcomes and there are cases in which any other action taken after that cannot change the outcome of the situation. In such instances, the random exploration mechanism, such as Gaussian noise, may have a significant impact on the critic's estimations, resulting in a deviation in the appraisal of the trajectories and hindering the convergence of the actor to an optimal policy.

This thesis proposes a modification of the AC architecture that feeds the critic with samples enriched with human expertise, in order to overcome the issue of limited exploration capabilities and reduced training phase times. The incorporation of human trajectories into the AC schema introduces implicit behaviours and knowledge that can be challenging to translate into a reward function or a complex controller. Consequently, the critic not only has available recent states visited by the actor, but also better or worse states visited by humans, along with the subsequent actions performed to take advantage of the situation or to solve it.

Figure 4.1 depicts the proposed learning schema. The principal contribution is illustrated at the bottom of the image, where there is depicted the process of retriev-

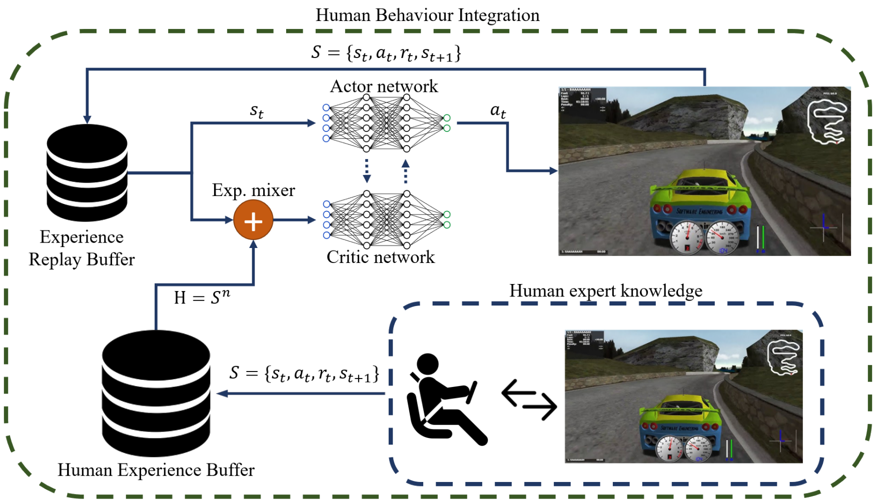


Figure 4.1: AC learning schema integrating Human Behaviour

ing the human knowledge in the form of trajectories resulting from different driving attempts performed by humans. The trajectories, denoted by $S = \{s_t, a_t, r_t, s_{t+1}\}$, are stored in the Human Experience Buffer (HEB hereafter).

The nature of the HEB is not distinct from that of the actor experience replay buffer. Consequently, the HEB may be either a static dataset of human trajectories generated in the past or a dynamic dataset that is constantly being filled with more human interactions. It is important to emphasise that the contents of the HEB must be diverse. In order to provide a more comprehensive understanding to the critic, it is essential that the HEB contains a balanced representation of both positive and negative trajectories. In other instances, the inclusion of this information may result in a positive or negative bias in the critic's decision, which in turn will affect the actor's policy.

For the precise case of this thesis, a custom interface has been developed to record the data produced by humans, as TORCS does not support this natively. Making use of this interface, three different human drivers have interacted with TORCS producing a total amount of more than 16,000 samples. The HEB thus is, for this case, a static replay buffer.

In addition to the previously described HEB, during the training phase the actor will interact with the environment, populating the Experience Replay Buffer (ERB from now on) with the trajectories resultant from its exploration. Once a sufficient number of training episodes have been completed and the ERB is full, the actor waits for the critic to produce its estimations on the generated trajectories. It is at this point when the critic will integrate the uninformed data in the ERB with a limited number of consecutive samples extracted from the informed human experience stored in the HEB. This mixing step is depicted in Figure 4.1 as the *experience mixer*. In the event that the HEB is a static buffer, as it is the case, it is essential to select the number of human samples to be extracted from it with great care. Neural networks are susceptible to overfitting when trained on samples that are frequently revisited. Consequently, a huge number of human samples constantly repeated in evaluation phase could produce a poor generalisation to unseen states. Furthermore, the retrieval of an excessive number of samples from HEB may result in a disproportionate credit being assigned to human samples relative to the exploration of the actor, thereby creating a discrepancy between the actions of the agent and the predictions of the critic. To avoid this issue, it is recommended to use sizes of H smaller than a tenth of the size of the ERB.

The proposed human-in-the-loop scheme presents a more efficient and cost-effective approach than the alternative of a human supervisor monitoring the agent's actions in real-time without losing the correction factor. Furthermore, this architectural approach is less invasive than initialising the actor with offline human trajectories, as it does not impede the exploratory nature of the actor. Which, instead, learns with the guidance of its own learned policy avoiding possible biases produced by the human initialisation.

4.2 Integrating Human Cognition

As it has been previously explained in Section 2.2, state representation is a crucial process in RL. States are comprehended by substantial information that the agent will use to tell apart between scenarios. In order to avoid situations of ambiguity where two different environment states share the same representation, the information included in the state representation must be carefully selected. Moreover, an incorrect representation can also affect the agent's performance in a negative way. In the field of ADS, car-dependant information such as the car weight, the engine's rpm or the angular velocity of the wheels can affect to the generalisation of the agent's driving capabilities to other cars. This casuistry makes state definition a complex task, in which the included information must be minimum to avoid biases or adjustments to certain properties whilst being detailed enough to avoid situations of ambiguity.

However, in the same manner that state definition can affect in a negative way the performance of the agent, it can also improve the agent's performance if enhanced with implicit information. This thesis proposes the adaptation of the agent's perception to make it more akin to humans'. Following this principle, agents are endowed with the ability to integrate retinal input of a visual scene with prior knowledge to make inferences similar to human decision-making.

Most of the sensory information provided by TORCS corresponds to static measurements. However, there is one particular sensor that can be configured to suit our needs. Recalling from the explanation in section 3.3, the *track* sensor is a collection of 19 range finder vectors that return the distance between the car and the edges of the road within a range of 200 metres in the direction of the given vectors. By default, the direction of the vectors is evenly distributed between -90º and 90º in intervals of 10º. For more clarification, we refer the reader to Figure 4.3.

This is evenly-spaced configuration is the one utilised in the works detailed in the previous section. Under this setting, the agent gives equal importance to what is ahead and what is beside. Paying attention to regions of the space that are not physically reachable due to the car turning radius [31].

By taking a loot at figure 4.2, our motivation to include human's visual cognition into the agent state representation becomes clear. The human field of view depicted in the figure shows that humans place most of their attention to what is within 20º of the visual centerline. Is in this *focus recognition area* where we can recognise and differentiate shapes, colours, words, and so on, with high precision. Beyond 30º, humans are not able to tell apart shapes, only colours, loosing the ability to identify objects. Applying the notions derived from this diagram to the driving context, this means that humans constantly look ahead on the road to be able to respond smoothly to the upcoming situations. We do not focus on what is aside of the car unless in certain punctual cases.

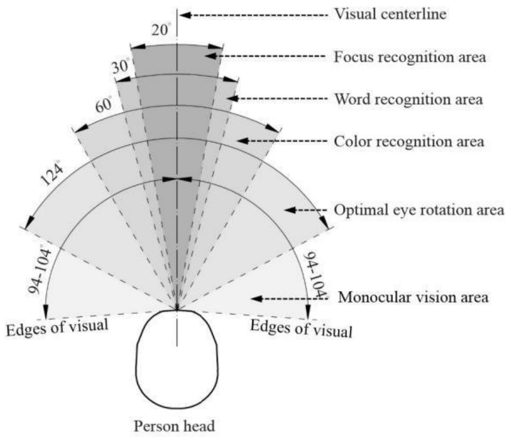


Figure 4.2: Human sight range, extracted from [79]

Hence, we replaced the default angles of the 19 range finder sensors with values that emulate human vision by (i) giving the most importance to what is around the visual centerline and placing five sensors around it (-1.5, -0.5, 0, 0.5, and 1.5); (ii) applying complementary attention to what is inside the focus recognition area placing the majority of the sensors (-11, -7, -4, -2.8, 2.8, 4, 7 and 11); and (iii) recovering a few concepts from the colour and word recognition areas with the six remaining sensors (-45, -32, -23, 23, 32 and 45). Figure 4.3 shows a comparison between the original placement of sensors (Figure 4.3(a)) and the human-centred configuration (Figure 4.3(b)). With this new representation of the field of view, we aim to dismiss information that is no longer relevant due to the car kinematics without losing representation capabilities.

Even with the newly proposed sensor configuration, there is still margin for improvement regarding the state representation. In some problems, temporal dependen-

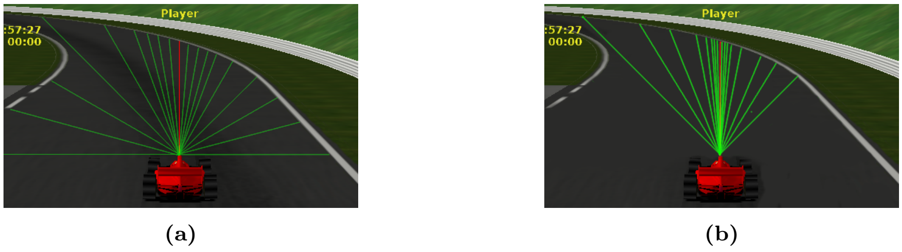


Figure 4.3: Track sensory visualisation: (a) Default. (b) Human-like

cies play a crucial role in understanding the environment. The Pong game represents a clear example of this situation. In [2], authors train a RL agent that is able to play by processing only snapshots captured from the game. As authors acknowledge, a single snapshot (frame) may not be enough to understand the current scenario. Figure 4.4(a) shows the input that the agent will receive. In it there is no information about neither the actual movement of the agent, nor the opponent or the ball. This supposes ambiguity between states, as the ball may be moving in any direction, and consequently the position of the agent may be correct or incorrect. To address this issue, authors propose the inclusion of the last N frames into the state representation (*frame-stacking*) to provide the agent with this temporal dependencies and thus, embedding information about trajectories (Figure 4.4(b)), which could not be sensed otherwise.

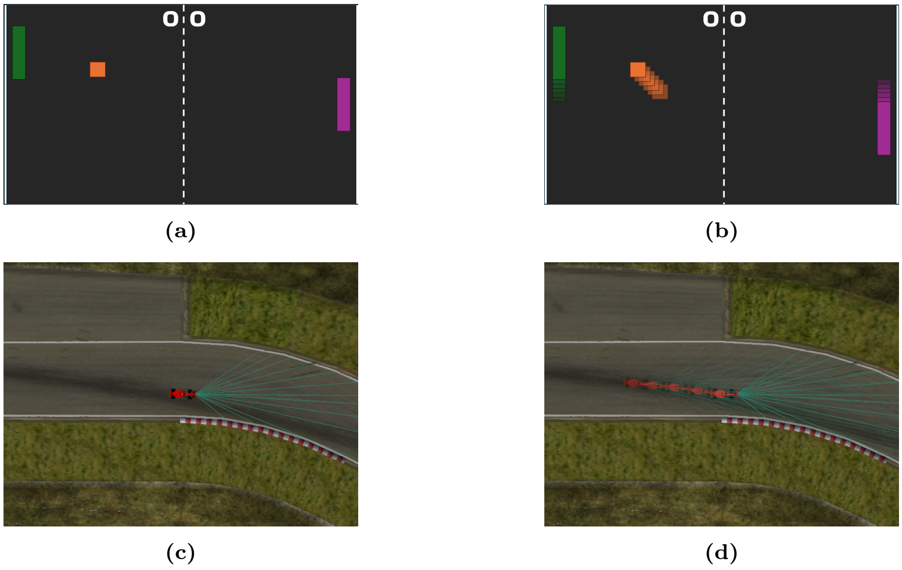


Figure 4.4: Visualisation of the frame-stacking effect

Applied to the ADS problem, some similarities can be drawn. With a single observation, the agent receives its distance with respect to the centreline, but it has no information about whether it is deviating or approaching to the centre. A similar case can be observed with the speed. The agent knows its current speed, but without knowledge about the previous speed it cannot know if it is in a speeding-up process or braking. Moreover, by stacking the information of the track sensor, the agent can built an internal representation of the curvature of the road while driving. This internal map may help the agent model turns in advance, and thus adapting the speed accordingly.

Form an anthropological point of view, the process of stacking frames is natural. Humans can perceive movement thanks to a phenomenon called *persistence of view*. This optical illusion occurs when the perception of an object persists for some time after the rays of light proceeding from it have ceased to enter the eye, causing the next perception to mix with the previous one. Persistence of view helps the brain in computing trajectories and sense speed [42].

By endowing the agent with human cognition capabilities, in terms of mimicking the human sight properties and persistence of view, we expect the agent to create an internal representation of the road modelling the curvature of the turns whilst approaching them. In terms of translation into TORCS, stacking n observations will give the agent information about the trajectory over the last $10n$ milliseconds, as it is the refresh rate for the sensors.

Chapter 5

HuBeC-Driving

This chapter details the formal representation of all the components comprising the Markov Decision Process (MDP) used to formalise this problem within the Reinforcement Learning (RL) framework. Moreover, the HuBeC-Driving (Human Behaviour and Cognition) implementation details are explained, applying the theoretical concepts previously introduced in Sections 4.1 and 4.2. Which modify the Actor-Critic (AC) schema by adding human samples and make use of the human perception mechanism as a reference for modelling the state representation respectively.

5.1 Problem Definition

In this section, we detail each one of the components comprising the MDP. Which, recalling from Section 2.2, is defined as a tuple $MDP = \langle S, A, R, T, \rho \rangle$, where S represents the set of states, A the set of available actions, R the reward function defined in terms of states and actions ($R : S \times A \rightarrow \mathbb{R}$), T the transition function between states ($T : S \times A \rightarrow S'$) and ρ the distribution over initial states.

The upcoming paragraphs detail the mathematical definition of each one of the aforementioned components (i.e. States, Actions and Reward function) along with an exhaustive explanation of the motives behind the utilised formalisation.

5.1.1 States

State definition is a key part of the MDP formalisation. It represents the information to be processed by the agent in order to decide an action to take. State representation must minimise ambiguities without loosing generalisation capabilities.

In this thesis we aim to develop an agent able to generalise its driving to different environments, comprising both unseen tracks and cars that were not driven before, all with a human-centred point of view.

From the large catalogue of sensory information available in TORCS (Table 3.1), some of the sensors are not available to the human from the driving position (e.g. the tire's angular velocity). Moreover, using too car-dependent inputs takes us away from a generalist model for on-road driving. For these reasons, we only selected a set of sensors that are accessible to the human while driving and that are shared by any car for the state representation. Table 5.1 shows the selected sensors and their unit range.

Name	Range (unit)
angle (α)	$[-\pi, \pi](rad)$
speedX (v_x)	$(-\infty, +\infty)(km/h)$
speedY (v_y)	$(-\infty, +\infty)(km/h)$
speedZ (v_z)	$(-\infty, +\infty)(km/h)$
track (t)	$[0, 200]^{19}(m)$
trackPos (t_p)	$(-\infty, +\infty)$

Table 5.1: TORCS sensors and unit values

The **angle** sensor is the direction in which the car faces at a certain location of the track; **speedX**, **speedY** and **speedZ** are the velocity at each degree of freedom (i.e., the velocity in the longitudinal axis, in the lateral axis and in the vertical axis, respectively); The **trackPos** returns the normalised distance between the car from the centre of the road. A value close to 1 means the car is on the left side of the road whilst close to -1 means otherwise. $|t_p| > 1$ means the car is off the track. Finally, the **Track** sensor, as detailed in Section 4.2, is composed by 19 range finder vectors that return the distance of the car from the edges of the road. To avoid issues produced by the different magnitude scales, all values have been normalised and defined in the range $[0, 1]$.

The sensors of Table 5.1 collect data from a single observation of the environment to be used as input for a classical AC algorithm. However, as stated at Section 4.2, in the Human Cognition model, we propose the stacking of the last n observations ($n > 1$) into a single one and its use as input to the RL agent. So that it can compute the observations correlation along with its internal state representation.

For this reason, states are defined in terms of *frames*. The formal representation of a state is shown in Equation 5.1, where f represents a single frame made up of the components listed in Table 5.1. The superscript d represents the number of frames that will persist on the agent's sight. It must be a natural number, and $d = 1$ corresponds to a case where no frame is stacked (i.e. a classical state representation).

$$\begin{aligned} f &= \langle \alpha, v_x, v_y, v_z, t, t_p \rangle \\ S^d &= f^d \end{aligned} \tag{5.1}$$

As it will be discussed later in Chapter 6.3, we present results with S^1 for the baseline and Human behaviour models, and with S^5 for the Human Cognition and the HuBeC-Driving models. We have also experimented with $d = 3$ and $d = 7$, but none of them has reached the minimum required results to be considered a viable option.

5.1.2 Actions

Among the seven available actions in TORCS (Table 3.2), two of them (named focus and meta) are used to configure a sensor and restart the race, respectively. Thus, they do not have a real impact on driving and can be safely ignored.

Out of the five remaining actions, we have discarded the clutch and gear actions. Recalling from Section 3.3, they correspond to the virtual clutch pedal and the next gear to introduce respectively. Both actions are highly correlated, as a new gear cannot be introduced without fully pressing the clutch pedal. This correlation is really difficult to be learned by a RL agent, since the majority of the time both actions are inactive.

Moreover, they are also extremely car-dependant as the gear-changing process is subjected to the current speed and RPMs of the vehicle's engine. Engines with different sizes, fuel types, number of valves, and so on, have a different regime of RPM, thus have a different gear progression. This way, in order to avoid high correlation between actions and generalise the driving behaviour to multiple cars, these two actions have been replaced by an automatic process. Which leaves us with three actions: accelerate, brake and turn the steering wheel.

In daily non-racing driving, it is unlikely to press both the accelerator and the brake pedals at the same time. For this reason, we combined both actions into a single one, called *ab*, which ranges from -1 to 1, where negative values represent slowing down and positive ones, speeding up. With this fusion, the continuous action space is reduced in favour of a faster convergence. Equation 5.2 shows the final actions and their range, where *ab* stands for the combined action slow down and speed up, and *st* stands for steering.

$$A = \{ab \in [-1, 1], st \in [-1, 1]\} \tag{5.2}$$

5.1.3 Reward Function

Rewards play a key role in the learning process. As in the case of states, the reward function must be carefully defined. It transmits to the agent the ultimate goal to pursue, differentiating good from undesired behaviours. The reward function must then, guide the agent through the action space, identifying actions that lead to states with positive amounts of reward. However, the reward function must be general enough so the capability of the agent to explore the search space is not diminished. In the case of ADS, reward functions are usually complex, involving various components that model the desired behaviour of the Vehicle.

One common issue that can be observed in ADS, is the existence of a local minimum in the reward space at which agents reach a speed that is high enough to earn significant rewards, but low enough to allow the car to be controlled solely by lifting the throttle pedal (i.e. without braking). As rewards are usually measured in terms of speed, and braking reduces it drastically, agents following a greedy policy tend to refrain from braking.

Another known issue arises when the agent is driving in a straight line. In this scenario, the agent tends to steer a few toward left and right making the car wiggle.

This phenomenon is called *slaloming* and it is recurrent in ADS [23]. One possible explanation for this phenomenon can be found in the **Track** sensor, which can be understood as the vehicle's way of perceiving the shape of the road. In long straights, these sensors may reach the maximum covered distance (200m) without finding an edge. This added to the huge penalty received by the agent for driving off-track, may be producing the slaloming, as the agent tries to anticipate corners.

With the design of our reward function, we aim to address these two issues. Firstly, we mitigate the avoidance of braking by adding a component that equally rewards both speeding up and braking. Secondly, we add a penalty component to mitigate the *slaloming* problem by minimising the difference between angle values. In addition to the previous two, we also added an extra reward-shaping component to minimise the lateral velocity applied to the car when it turns. Which should lead the agent to enter turns more steadily, reducing its speed in favour of safety.

$$R_1 = 2|\hat{v}_x^i - \hat{v}_x^{i-d}| + \hat{v}_x^i(\cos\alpha^i - \sin|\alpha^i|) - \hat{v}_y^i \cos\alpha^i - |\alpha^i - \alpha^{i-d}| \qquad (5.3)$$

Equation 5.3 shows the mathematical formula of the reward function, where $\hat{v}_x^i$ denotes the normalised value of the speed along the x axis at time i whereas $\hat{v}_x^{i-d}$ represents the same but d frames before. The components of the formula are, respectively: the increment or decrement of speed between frames in absolute value; the movement along the road in the track reference system; the lateral velocity penalty; and the variation of the angle in absolute value.

5.2 Implementation

This section covers the details of implementation of the HuBeC-Driving proposal. Firstly, the topology of the neural networks are covered, describing a key temporal limitation. Next, an additional exploration mechanism will be commented. Finally, the driving tracks used during the training phase will be described.

5.2.1 TORCS Modifications

The source code for TORCS is available at SourceForge [1]. This corresponds to the release 1.3.7, which is the last version available. Some modifications have been made to the source code to facilitate the training process. Firstly, the initial three-second countdown before starting the race has been removed. Secondly, a restart mechanism for external bots has been implemented, as it was previously only available to TORCS bots and human players. Then, the human controller has been adapted to record human driving trajectories. Finally, the TORCS configuration has been changed to enable its use from the command line (which allows reaching a simulation time x128 times faster than real time) and to use custom configuration files to select different

[1] `https://sourceforge.net/projects/torcs/`

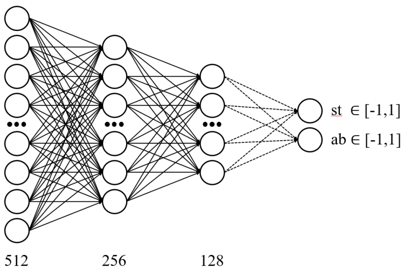
tracks during training. The project containing this modifications is available at the author's GitHub Page [2]

5.2.2 Network Topology

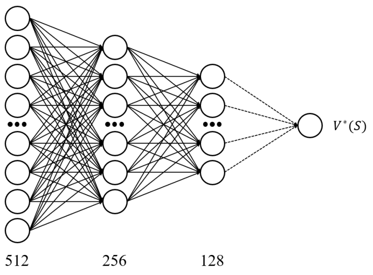
As explained before, HuBeC-Driving builds on actor-critic algorithms. From this family, Proximal Policy Optimisation (PPO) [66] is state of the art for its robustness and generalisation capability due to a careful update of the policy weights. It avoids huge oscillations in the policy specially at early stages of the learning process. Also, unlike DQN [50], PPO can be used in problems with continuous action space. For those reasons, PPO is the selected learning algorithm to solve this problem.

In working with TORCS, synchronisation between the environment and the agent is a crucial factor. As detailed in Section 3.3, the agent has a 10 ms deadline to respond after receiving an observation. Given the difficulty of recovering from a timeout situation (as actions taken in the past can strongly affect present states), the decision-making process in both training and inference times must take less than this limit as specified in the TORCS manual. To comply with this limitation, we chose a fully connected feed-forward architecture for both $Q^*(s, a)$ (actor) and $V^*(S)$ (critic) networks. Both networks share the same specifications, with three hidden layers each having 512, 256 and 128 neurons, respectively. Between each one of the hidden layers, there has been applied a Dropout [72] regularisation (with probability $p = 0.1$) and a ReLU [53] activation function.

With the aforementioned network topology, the elapsed time between action decisions is around 0.8ms when using TORCS in textual mode and 4.5ms when using TORCS in render mode. This is 12.5 and 2.22 times less respectively than the 10ms limit specified.



(a) Actor



(b) Critic

Figure 5.1: Visualisation of the actor and critic network topologies

It should be noted that the output layer changes depending on the network. The actor network outputs the actions to take, and it is composed of two neurons with a hyperbolic tangent activation function to ensure the range limitations defined in

[2]`https://github.com/migarbo1/TORCS`

Equation 5.2 (Figure 5.1a). More specifically, the output of this network is the mean of a multivariate normal distribution that determines the action to sample, which is a common practice in policy learning with continuous action spaces. The covariance matrix of the distribution is a diagonal matrix which is fixed and will be used for exploration purposes.

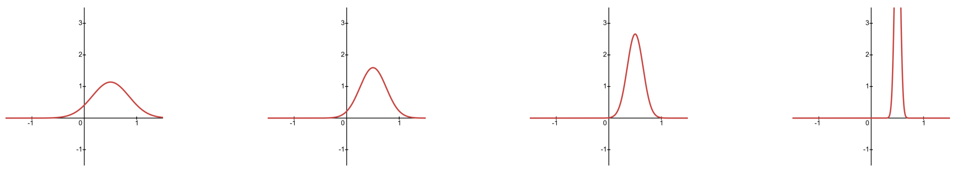
Regarding the value function (i.e. The critic), as its purpose is to approximate the raw accumulated reward that can be obtained from a certain state onwards, the neural network of the critic will only have a single output neuron with no activation function, as the reward does not have an upper nor lower bound limits.

5.2.3 Exploration Factor

In small problems with a limited number of states and actions, most RL algorithms can find the optimal policy with the algorithm's own exploration factor. However, the huge size of the solution space in the AD problem demands the need to add an exploration mechanism supplementary to the one of the PPO algorithm, ensured by the limited update of the policy.

The ϵ-greedy algorithm, as explained in Section 2.3, assists to learning methods, such as Q-learning or DQN, in exploring the action space of the problem. Its mechanism is simple yet effective. With probability $p = 1 - \epsilon$ the agent select the action resulting from the inference process. And with probability $p = \epsilon$ the agent selects the next action to take at random between all the possible ones. However, the ϵ-greedy algorithm is only applicable to problems with discrete action spaces. This is because, in the case of a continuous action space, the number of actions to pick at random is infinite.

In this proposal, actions are obtained using a multivariate normal distribution with mean in the actor network's output and a fixed covariance matrix. With this setting is crucial to fill the covariance matrix with the right value. High values of the action range (Equation 5.2) can result in actions that differ from the agent's intention (e.g. braking instead of speeding up); but small values can over-specify the agent's range of acting, hindering exploration.



(a) $\Sigma = 0.35$ **(b)** $\Sigma = 0.25$ **(c)** $\Sigma = 0.15$ **(d)** $\Sigma = 0.05$

Figure 5.2: Visualisation of the effect of Σ in action selection for a mean of 0.5

We can handle this trade-off between exploration and exploitation by using a *covariance annealing* process. This implies setting a high covariance value (Σ) at the start of the training and gradually decreasing it as the process comes to an end. For

ADS, we adopted a linear decrease proportional to the percentage of the steps that have been completed. This way, a high level of exploration is achieved at the early stages of the learning process whilst reducing the variability of actions towards the end.

Figure 5.2 presents a visual example of the procedure for an action with mean $ab = 0.5$ (i.e. press the acceleration pedal half way). At the beginning of the training phase (Figure 5.2a), the probability distribution is almost flat, so it is fairly possible that the agent accelerates way more or less than desired. However, as the training phase comes to an end (Figures 5.2c and 5.2d) the probability of performing an action substantially different from the mean (i.e. the output of the actor network) is almost zero.

For the particular case of this thesis, the initial value is set to $\Sigma = 0.35$ and it is decreased linearly until it reaches a floor value of $\Sigma = 0.05$, which is also used at inference time.

5.2.4 Training Tracks

Just as datasets can induce bias in models, in ADS the roads used during the training phase affect how the model will behave later on other roads. The goal of this thesis is to train an agent able to learn a policy that is able to generalise to unseen tracks.

A similar goal is pursued in [23], where the authors attempt to achieve it by making use of all available tracks during the training phase. More precisely, to prevent the agent from over-adjusting its driving to a specific layout, the training track is changed in a cyclic manner after five training epochs (i.e. starting from the first track, the agent performs five epochs and changes the environment to the next one. When this procedure has been completed on the last track, it starts again from the one used at first). Nevertheless, this approach cannot be considered a genuine pursuit of generalisation, given that all the tracks driven during the testing phase are initially employed during training. Consequently, the agent never faces novel and unseen states.

Rather than training with all tracks at once, in this thesis we aim to achieve generalist driving by carefully selecting a subset of tracks specifically oriented to the training phase. To this end, it is important to ensure that the selected tracks are fairly representative, featuring different aspects and difficulties of driving so that the agent can generalise its knowledge to unseen situations.

Every road, not only in TORCS but in general, can be described using a finite set of characteristics, such as high speed turns, sharp corners, the presence or absence of long straight lanes, wide or narrow edges; etc. In this thesis, we worked in the selection of a subset of four roads that encapsulate all the casuistry that drivers can face in regular on-road driving. Figure 5.3 displays the chosen tracks.

To justify the selection of the roads depicted in Figure 5.3, below there is a description of their main characteristics:

- **Wheel-2** is the TORCS representation of an official F1 circuit called Suzuka. It is composed of 4 high-speed turns in a row followed by a *hairpin* turn and a long straight. Wheel-2 is a very complete track in terms of complexity.

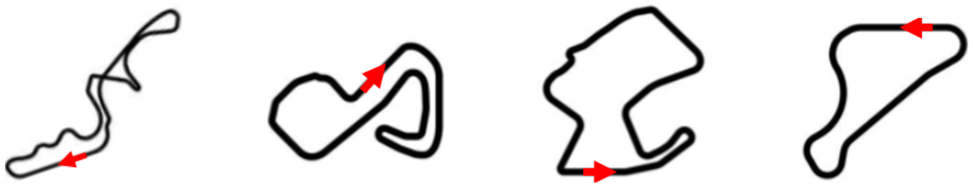

(a) Wheel-2 **(b)** Brondehach **(c)** Corkscrew **(d)** G-track-1

Figure 5.3: Subset of selected tracks for training . The red arrow indicates the position of the finish line and the direction of the track.

- **Brondehach** has the typical ninety-degree turns found on city streets, as well as long corners (similar to roundabouts), short straights and a hairpin bend.
- **Corkscrew** is a country road with blind turns and constant acceleration and braking zones. It resembles to the roads connecting distant towns.
- **G-track-1** is an oval track that can be regarded as a highway, with open wide turns and plenty of driving space.

The first three circuits were reserved for training and G-track-1 was used as a validation environment for choosing the model with the best performance. The training procedure is carried out as follows: the agent is trained for a certain number of timesteps (18 million) given a set of configurable parameters. During this phase, at the beginning of each epoch, the track is chosen randomly. After three epochs, the agent is evaluated. When the obtained performance is better than the previous one, the policy is saved. By following this procedure, we aim to prevent the agent from adapting its driving to a single driving track.

Finally, it is important to state that all training and test processes have been done using the TORCS replica of a F1 car. Although it may seem odd to focus on every day driving with a formula one, actual automobile industry receives the majority of its improvements (in terms of engines, brake systems, automatic transmissions, etc.) from the different Motorsport competitions.

Additionally, from the driving perspective, F1 cars have the most complex dynamics. They have a thousand horsepower engine inside a car that weights less than a tone, brakes are also really sensitive, and the relationship between both of them and the steering wheel must be coordinate in order to avoid spinning along the track. By forcing the agent to learn how to drive an F1 car, we make it face the most difficult scenario of driving, so any other car will represent an easier instance of the same problem.

5.2.5 Gathering Human Data

When working with human samples, it must be ensured that the data is recorded in an environment where humans fells comfortable. Also, in order to keep the process as close to reality as possible, the mapping between the actions and the controllers of the simulator must be direct. In the context of AD, this means that we have to provide users with both software and hardware that make them feel like they are driving a real car.

As stated before, the dynamics of a Formula 1 car are really complex, for this reason we have build a series of driving assistance features that will help the volunteers to control the car. In this sense, the user is provided with an additional traction control algorithm to ensure that the wheels do not loose traction. Also to avoid the driver from spinning at corner exits due to the amount of horsepower. Finally, an Anti Block System (ABS) has been developed for the brakes, in order to ease the braking events. With these modifications, the software is ready.

The utilised hardware platform is a set of Logitech G920 Steering wheel and pedals with a force feedback mechanism [3]. It applied some vibrations and resistance to the steering wheel when driving at high speeds or when colliding with an obstacle. Both peripherals plus the chair emulating a racing car seat, create an immersive set-up.

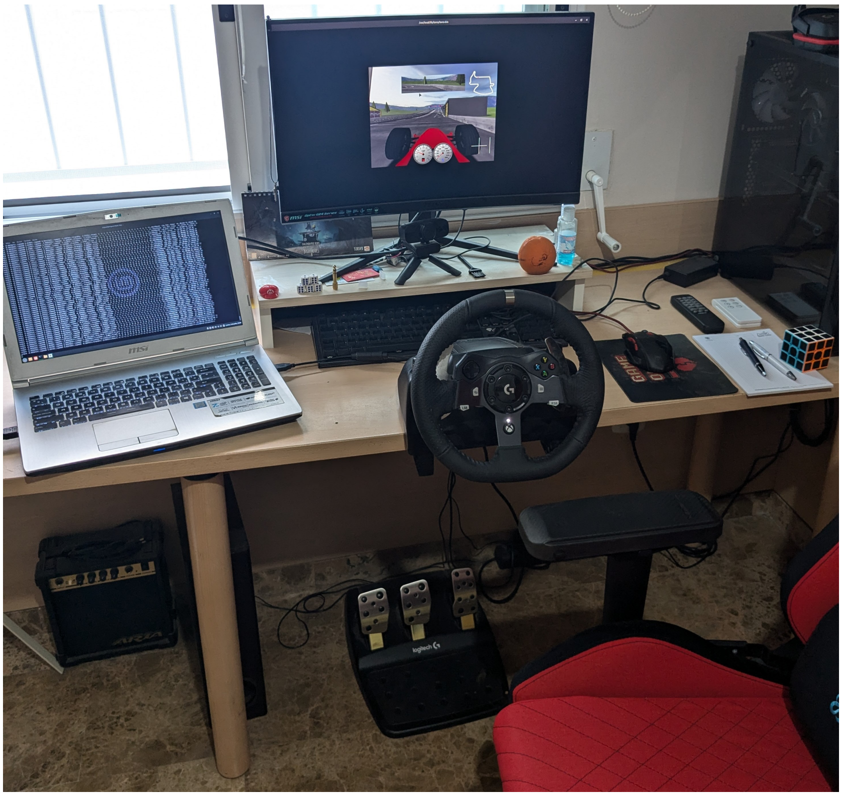


Figure 5.4: Physical environment to utilised to record the human data

[3]`https://www.logitechg.com/es-es/products/driving/driving-force-racing-wheel.html`

To gather the data, the TORCS internal human controller has been modified to add additional sensors and loggers. With them, the emulator stores the states that the humans visit with the same observation formalisation that has been detailed in Section 5.1.1. Along with states, the system records the action performed by the users. As they can both press the acceleration and brake pedals at the same time, a mechanism has been developed to determine the actual action that is transmitted to the car. This supposes that only 2 actions are stored, st and ab, just as defined in Equation 5.2.

For the gathering process, a call for volunteers was conducted within relatives and friends of the author. As a result, 5 different people, both men and women between 22 and 26 years, have recorded three laps on each of the training tracks (i.e. *Brondehach*, *corkscrew* and *Wheel-2*) and three additional laps on the *Aalborg* track, because, as will be explained later on this thesis, shows a complex layout in which RL agents struggle to perform. As a result, more than 600000 state-action pairs have been obtained from the process.

Chapter 6
Evaluation and results

This chapter outlines the specific evaluation details. Specifically, the nomenclature utilised to describe the distinct developed models and the parameters utilised during training to ensure replicability standards are met. Furthermore, a detailed comparison between the developed models is provided. Explaining the benefits of each one of the approaches.

6.1 Evaluation

As previously stated in Section 3.4, topic diversity in research projects using TORCS reflects the lack of a unified evaluation criteria to compare the proposed models in terms of general performance. The absence of a benchmark makes it challenging to quantify the potential gains that could be achieved through the implementation of the proposed solutions or architectures. This casuistry forces researchers to develop their own baseline model in order to establish a ground truth. The purpose of which is to determine whether the proposed modifications represent genuine improvements.

In line with the aforementioned considerations, in this thesis we have trained a baseline model using a standard PPO implementation. This approach allows us to quantify the impact of each modification on performance. Then, we have gradually added the human-centred enhancements detailed in previous sections, resulting in three more models The objective of this thesis is to analyse the performance of each model across all roads, with the aim of achieving consistency and safety on each track. The specific details of each of the models are provided below.

- **Baseline**: This corresponds to a standard implementation of the PPO algorithm, with S^1 (i.e. no frame-stacking) and standard sensor placement (i.e. evenly spaced between -90º and 90º, as in Figure 4.3a).
- **Human Cognition** (HuC): This corresponds to a standard implementation of the PPO algorithm, but adapting the sensors to human perception. This model utilises frame-stacking with 5 frames (i.e. S^5) and a human-like vision (Figure 4.3b).

- **Human Behaviour** (HuBe): This corresponds to an implementation of the enhanced PPO schema detailed in Section 4.1 providing online human feedback during training (as depicted in Figure 4.1). This model uses a standard sensor placement (i.e. evenly spaced between -90º and 90º, as in Figure 4.3a) and S^1 (i.e. no frame-stacking).

- **Human Behaviour and Cognition** (HuBeC): This model combines both proposals. First, it is implemented using the enhanced PPO schema detailed in Section 4.1. Second, it uses frame-staking with 5 frames, as well as a human-like placement of sensors (Figure 4.3b).

This thesis employs three distinct metrics to assess the efficacy of the developed models. In order to provide results and conclusions that are both relevant to this work and to any ADS developed in TORCS, the selected metrics are independent of the reward function and measure the results produced in an objective manner.

Firstly, we report the average distance driven by the agent across all tracks, along with the average speed. The combination of these two metrics provides provide a comprehensive overview of the agent's performance understood in terms of consistency across different roads.

This is achieved by measuring the speed at which the agent drives at any point on the track during a single lap (i.e. the telemetry). Telemetry is employed in motorsport competitions to ascertain the lap times of the drivers, with a view to identifying potential gains or losses in time.

The three metrics (i.e. average distance, average speed and telemetry) represent general lectures from sensors that are independent of the reward function utilised, the architecture, the state representation or the precise implementation of the agent. Consequently, the findings presented in this thesis can be utilised as a reference point in the development of ADS in TORCS.

6.2 Replicability

During the development process, in an effort of transparency, we have ensured that all the proposed models and their results can be easily replicated. Every RL training was conducted on a machine with a Nvidia GeForce RTX 3090 GPU, a 12th Gen Intel(R) Core(TM) i9-12900KF CPU and Ubuntu 22.04 LTS operating system. The models were trained for 18 million steps with the listed hyperparameters: $\gamma = 0.99$; five updates per iteration, $clip = 0.2$, $entropy\_coef = 0.05$ and $\lambda = 0.98$.

In the validation phase, performance is evaluated through the analysis of episodes. An episode is considered complete when the agent drives off the road, stops the car, drives in reverse, collides with a wall, or reaches the maximum number of steps (set to 50,000). The mean distance and speed are calculated across 10 distinct episodes in each of the 19 tracks. The telemetry is obtained by selecting the fastest lap completed by the vehicle during the episode.

The complete set of proposed model weights, the TORCS Python interface, and the training and evaluation algorithms can be accessed on the author's GitHub page[1].

6.3 Results

This section presents the outcomes of the various developed models. This section presents a detailed comparison of the performance of the models in terms of the average distance driven across all tracks and the average speed at which the agent has been driven. Furthermore, this section presents a comprehensive analysis of the driving behaviour exhibited by the optimal model, which has been contrasted with the average human action.

6.3.1 Overall model comparison

With the aforementioned configuration, the four specified methods have been trained for 18 million steps. Subsequently, an evaluation phase has been conducted, during which the driving consistency of all agents has been tested across 10 episodes (i.e. 50000 steps maximum) in all tracks. Additionally, the average speed for each track has been recorded during the evaluation phase, providing a more comprehensive interpretation of the results. Figure 6.1 illustrates the average speed per track, while 6.1 presents a comparison of the models in terms of the average distance covered in each track.

In order to provide a more insightful comparison between models, a categorisation process has been developed which classifies the 19 tracks into three different categories. The C1 category of tracks comprises narrow, curvy roads with slow speed turns. These require the agent to exercise precise control of the car in order to complete a lap. C2 tracks present wider roads with open turns and long straights. The most crucial factor on these tracks is speed. Finally, C3 tracks present a combination of the two previously described features. The roads within this category exhibit both long straights with open turns and narrow and closed corners. The successful management of braking and speeding-up events is crucial to perform optimally in these tracks.

As it can be observed in Table 6.1, the baseline PPO algorithm produces some discrete results. The algorithm is able to complete a lap, on average, in 13 of the 19 tracks. Nevertheless, in only eight instances is it able to complete more than one lap. The model tends to reach high speeds (as evidenced by the blue line in Figure 6.1) in order to obtain higher rewards. However, the model lacks effective control of the vehicle, resulting in frequent collisions and, consequently, a reduction in the consistency of the average distance. This is clearly evident when comparing the performance of the agent in both **C1** and **C2** tracks. In the first category (i.e. the one requiring the most car control to complete a lap), the agent completed only one lap in the majority of cases. The only exception was the e-road track. However, in **C2** tracks (i.e. where higher speeds are of greater importance than car control), the agent consistently completes laps. This pattern is also reflected in the **C3** category, where

[1]`https://github.com/migarbo1/autonomous-racing-in-torcs`

the agent's performance varies depending on the track. In some instances (such as *brondehach* or *Corkscrew*), the agent is unable to complete a single lap. Conversely, in other scenarios (e.g. *g-track-3*, *wheel-1*) the agent demonstrates proficiency in navigating the track.

Cat.	**Name**	**Length**	**Baseline**	**HuC**	**HuBe**	**HuBeC**
C1	aalborg	2.58	0.22	0.22	***0.44**	*0.40
	alpine-1	6.35	12.32	**23.25**	21.97	20.44
	alpine-2	3.77	2.54	8.13	6.89	**15.80**
	eroad	5.38	20.77	28.73	28.99	**29.93**
	e-track-2	4.20	2.21	2.69	2.28	**2.91**
	e-track-6	3.26	7.56	19.50	20.27	**30.74**
	ole-road-1	6.28	7.10	6.94	**27.79**	18.94
	street-1	3.82	2.86	**2.95**	2.87	2.87
C2	e-track-4	4.44	21.84	34.57	35.05	**36.06**
	forza	5.78	17.29	16.85	25.92	**32.40**
	g-track-1	2.05	20.75	29.31	**31.36**	30.740
	g-track-2	3.18	22.42	31.73	**34.17**	33.98
	ruudskogen	3.27	14.40	22.34	26.00	**27.70**
C3	**brondehach**	3.91	4.05	18.84	*7.63	***29.76**
	corkscrew	3.60	2.73	2.99	*4.77	***14.15**
	e-track-3	7.04	5.91	5.69	23.43	**25.73**
	g-track-3	2.84	23.00	21.22	27.51	**28.85**
	wheel-1	4.32	20.89	30.12	**32.25**	26.60
	wheel-2	6.20	12.17	11.82	*9.82	***32.36**
	Total Average		11.63	16.73	19.44	**23.18**

Table 6.1: The mean distance covered across all tracks in metres is presented here. It should be noted that the names in bold indicate that the layout has been used for training purposes, while the results accompanied by an asterisk (*) indicate the appearance of the layouts on the recorded human data.

The second column of Table 6.1 presents the results obtained by the model using the modifications detailed in Section 4.2 referring to include a human-like perception of the environment (HuC model). The incorporation of temporal dependencies into the perception system, as provided by the frame-stacking methodology, resulted in enhanced car handling across **C1** tracks. This model achieves a performance that is almost double that of the baseline in three of the eight tracks within this category (e.g *alpine-1*, *alpine-2*, *e-track-6*). Despite showing a considerably lower speed (approximately around 30km/h less than the baseline, as seen in the orange line in Figure 6.1). This consistency is also reflected in the performance of the model in the other two categories. In four of the five **C2** tracks, the HuC model covers 33% more distance than the baseline despite the considerable gap in terms of average speed. It is evident that the HuC model displays a safe driving style, which allows it to generalise more effectively to diverse track types.

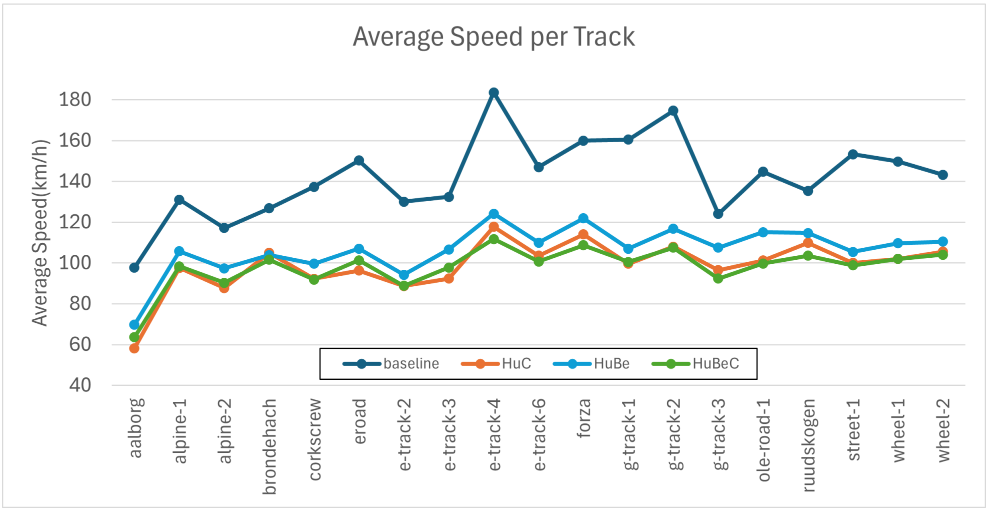


Figure 6.1: Average speed per track across 10 evaluation episodes.

In the third column, the reader will find a detailed account of the advantages gained from the incorporation of the novel PPO architecture (HuBe). As outlined in Section 4.1, this model incorporates a novel dedicated replay buffer for human trajectories. This can be understood as the incorporation of implicit human knowledge into the agent. From the performance obtained by the HuBe agent, two major conclusions can be drawn. Firstly, the incorporation of human samples recorded in the Aalborg track has proven beneficial to the agent in navigating narrow and winding layouts. As a consequence of this modification, the HuBe model is able to achieve a performance comparable to that of the HuC model in **C1** layouts. Moreover, it can be demonstrated that the incorporation of a human-enhanced replay buffer has not resulted in a significant bias in the agent's behaviour. This is clearly evident when comparing the performance of the baseline and HuBe models in the tracks where human experience has been recorded (e.g. *aalborg*, *brondehach*, *corkscrew* and *wheel-2*). In all cases, the performance of the two models has been found to be similar. The HuBe model achieves slightly higher speeds than the HuC model, which, when combined with the consistency acquired, results in superior performance than the HuC model over the **C2** tracks.

Finally, column four presents the results obtained by combining the two human-centred approaches (HuBeC). This model demonstrates superior performance compared to the other models in 12 out of the 19 models, exhibiting clear superiority across all three categories of tracks. In **C1**, the processing of human samples with the proposed central focus and temporal dependency yields the best performance in four of the six tracks where all agents perform at least one lap. In **C2**, the reduction in speed experienced by HuBeC, in comparison to HuBe, as a result of the incorporation of human cognition, has a slight impact on the performance observed in tracks within this category. Nevertheless, the consistency of HuBeC enables it to achieve the best performance on three of the five tracks and a performance approaching the best on two of them. Finally, the dominance in terms of car control and consistency is also reflected to the category **C3**. It is on this tracks where the combination of the two human-centred proposals is the most effective. In the three tracks used for both during training phase and human data collection (i.e. *brondehach*, *corkscrew* and *wheel-2*), the agent has demonstrated an enhanced ability to interpret input, resulting in a performance increase of over 200% on average on these tracks. Consequently, the proposed modifications to the PPO algorithm, as outlined in this thesis, have resulted in a 93.30% improvement in its performance when applied to the autonomous driving problem in TORCS. This has enabled the development of a generalist policy that is capable of driving safely and successfully in previously unseen tracks.

6.3.2 Telemetry

In addition to measuring performance in terms of distance covered, this thesis also compares performance in terms of driving behaviour. For this purpose, the speed of the agents was measured at each point of the lap (a telemetry study) on the *wheel-2* track (see Figure 5.2a). This track was selected for the analysis because it has three distinct sectors that can be easily distinguished. As seen in Figure 5.2a, the track consists of a series of seven consecutive turns, followed by a hairpin and then a long straight.

Figure 6.2 shows two distinct telemetry lines, reflecting the outcome of human driving (red line) and HuBeC driving (Green line). To generate the human data, we have selected the best lap on *wheel-2* of each one of our volunteers and we have computed the average speed at each position of the track. To generate the HuBeC data, an inference execution of the model was run over the given track for a total of 50,000 time steps. From the various laps completed, the fastest lap is selected. From this, we extract the speed and distance at each point on the track.

In Figure 6.2, the Y axis represents the speed in km/h. The X axis represents the distance, in metres, from the finish line and ranges from 0 (start) to 6,199m (just before reaching the finish line). The vertical dotted lines indicate the location of a turn. The finish line is located at the bottom left straight as shown in Figure 5.3, and the direction of the track is clock-wise.

As can be observed, the average human approaches the first turn at a significantly higher speed than the HuBeC model. Additionally, humans tend to apply more force when braking, resulting in a slower speed at the entrance of the corners. In contrast, the HuBeC model enters the first turn at a higher speed and brakes just before turn two. Both humans and HuBeC approach the next chain of five turns (from 800m to 1500m) in a similar manner, maintaining a similar speed and applying a degree of braking at turn six (approximately 1300m from the start). Upon exiting the chained turns, humans tend to accelerate more than the agent, which maintains a conservative speed. At turns eight and nine (located 2100m and 2300m from the start), it can be observed that there is minimal difference between the human and agent behaviours, as both lines exhibit a similar shape. However, once again, at the exit of the turn

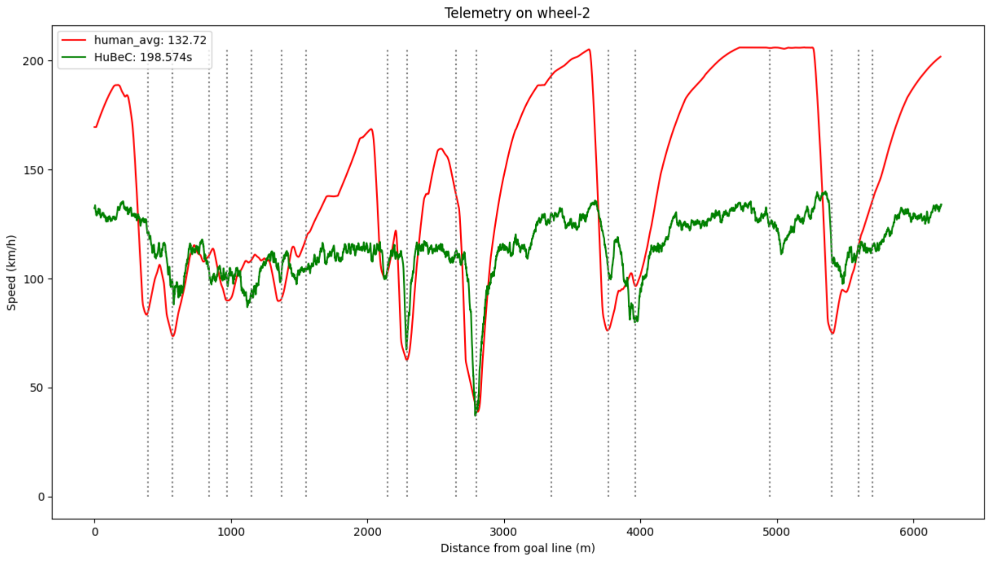


Figure 6.2: Telemetry comparison between the average human best lap and a lap completed by HuBeC over the *wheel-2* layout (Figure 5.3a).

nine, humans tend to accelerate more than the agent. This implies that humans must brake earlier (approximately 150 metres) than HuBeC when approaching turn 11 (2850 metres from the start), the slowest section of the track. In the next straight, the previously observed behaviour is again evident. As they approach the entrance to turns 13 and 14, humans reach higher speeds, which forces them to brake earlier and more extensively. This phenomenon is observed once more in the final straight and the final two corners of the circuit (5700m and 5800m from the start).

In general terms, it can be stated that in the search for a generalist and safe driving, the HuBeC agent tends to achieve a speed that is high enough to obtain significant rewards, while ensuring safety on the majority of tracks. Consequently, the HuBeC model exhibits deficiencies in its behaviour at corner exits. Nevertheless, in the context of braking events, the model demonstrates a degree of consistency with human standards.

Chapter 7
Conclusions and future work

This section details further researches that have been conducted after the completion the aforementioned work. Here will be detailed an additional experimentation that could not be included in previous sections due to a lack of time. In addition, some ideas of future lines of research are presented.

7.1 Conclusions

This thesis presents the development of a robust autonomous driving system (ADS) capable of adapting its driving style to different road layouts, ensuring safe and comfortable operation in previously unseen scenarios. In the course of this process, a review of several ADS platforms was conducted with the objective of identifying the optimal solution that balances realism with computational cost efficiency.

With regard to the development of ADS in TORCS, it has been acknowledged that there is a lack of a common criterion for the evaluation of research proposals. Consequently, reward-independent metrics have been employed in order to facilitate the utilisation of the results as a benchmark in the context of ADS development in TORCS. Moreover, we have gained insight into and implemented a state-of-the-art RL algorithm, such as PPO, which has subsequently been enhanced by the introduction of a dedicated human replay buffer. This has enabled the algorithm to obtain informed samples and assist it in identifying the optimal policy.

Finally, we have substantiated our proposals with an exhaustive evaluation phase, during which we have compared both the overall performance of the model and the precise behaviour of the agents. The results of this research provide a foundation for further investigation into the potential for incorporating implicit human knowledge into ADS without introducing significant biases in the resulting policy.

7.2 Future Work

The contents of this thesis represent the initial stages of the pursuit of a PhD thesis. In this sense, this work provides a foundation for future developments and improvements upon the already favourable results.

7.2.1 Ongoing research

Recently, our research has concentrated on a more detailed examination of the manner in which information is represented within the aforementioned simulator, as well as the degree of liberty that the agent should be afforded when undertaking actions. This has involved a detailed analysis of the trade-off between the expressiveness that a more complex action space provides, and the increased complexity and time required for the algorithm to converge.

In this context, we have evaluated the consequences of implementing all three selected actions simultaneously (i.e. acceleration, deceleration and steering), thereby affording the agent greater autonomy and control over the vehicle. Equation 7.1 illustrates the contrast between the action space delineated in this thesis (up) and the proposed novel action space (bottom).

$$\begin{aligned} A =& \{ab \in [-1,1], st \in [-1,1]\} \\ A' =& \{a \in [0,1], b \in [0,1], st \in [-1,1]\} \end{aligned} \tag{7.1}$$

In addition, we have studied novel approaches to the reward function. In more precise terms, we have conducted a review of the utilisation of a novel paradigm. In contrast to the speed-based reward function (as employed in this thesis), an alternative approach is proposed: a distance-based reward function. Equation 7.2 utilises the distance covered between the previous state (dr^{i-d}) and the current one (dr^{i}) to establish the basis of the reward. The distance traversed between steps is weighted by the trajectory followed and the distance from the centreline. As lectures between states are conducted at regular intervals (every 20 ms), the velocity of the car is implicitly represented, as higher speeds will result in greater distances being covered between states. However, by avoiding the explicit declaration of the speed within the reward function, we aim to help the agent understand that faster only means better under certain circumstances.

$$R_2 = (dr^{i} - dr^{i-d}) \cos \alpha^{i} (1 - |t_p|) \tag{7.2}$$

In a close future, we aim to asses the results obtained from these proposals and evaluate how their combination with the improvements presented in this thesis can increase the performance of the obtained models.

7.2.2 Future research

In the mid-range period, it would be of interest to adapt the proposals set out in this thesis to other environments, such as F1_tenth, as it is based on ROS. This will facilitate the translation of the proposals to the field of robotics. Nevertheless, the adaptation to ROS is not straightforward, as it necessitates a novel paradigm (i.e. all the sensors and actuators are asynchronous).

Furthermore, it would be beneficial to investigate additional techniques, such as pre-training the critic network to enable precise prediction of the value function, $V^*(S)$, from the outset of the training process. In this setting, the algorithms could enhance their convergence ratio, thereby reducing the number of steps required to obtain an optimal policy.

Finally, the field of model learning could be explored. In this context, during the training phase, the agent not only learns the optimal policy (Q*) and the value function (V*) but also the dynamics of the environment ($S \times A \rightarrow S', R$). In this sense, the agent is then able to utilise its internal model of the environment to perform simulated decisions that do not have an impact on the environment.

ACKNOWLEDGEMENTS

Firstly, I would like to start by acknowledging the effort and dedication shown by the supervisors. Eva and Ángel, thank you so much for your insights, recommendations and patience when facing undesired results and for the commitment that you have proven in the various deadlines for conference papers. I really hope this is the beginning of a long and fruitful journey together.

Also, I could not be grateful enough for the time that Elena, Alejandro, Gerard, Mario and Laura have dedicated to it. Their support to this project goes beyond the data gathering process, they have been a key pillar during all the stressful situations.

Last but not least, I would like to express my thanks to my family. Their interest and curiosity have been overwhelming. Also their unconditional support, as they have suffered from the tight deadlines and cheered for the good results. Under any circumstance they have been side-by-side, contributing with care and helping to clear the mind. Indeed, sometimes attention is all you need ;)

(This thesis has been partially founded by the project TED2021-131295B-C32 and the VALGRAI foundation)

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 1, New York, NY, USA, 2004. Association for Computing Machinery.

[2] Aakriti Adhikari and Yang Ren. Rl-pong: Playing pong from pixels. *arXiv preprint arXiv:1903.00374*, 2021.

[3] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 449–458. PMLR, 06–11 Aug 2017.

[4] Johannes Betz, Hongrui Zheng, Alexander Liniger, Ugo Rosolia, Phillip Karle, Madhur Behl, Venkat Krovi, and Rahul Mangharam. Autonomous vehicles on the edge: A survey on autonomous vehicle racing. *IEEE Open Journal of Intelligent Transportation Systems*, 3:458–488, 2022.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[6] Mohammad Reza Bonyadi, Zbigniew Michalewicz, Samadhi Nallaperuma, and Frank 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.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[8] Maximilian Brunner, Ugo Rosolia, Jon Gonzales, and Francesco Borrelli. Repetitive learning model predictive control: An autonomous racing example. In *2017 IEEE 56th annual conference on decision and control (CDC)*, pages 2545–2550. IEEE, 2017.

[9] Martin V. Butz and Thies D. Lonneker. Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 317–324, 2009.

[10] Long Chen, Yuchen Li, Luxi Li, Shuangying Qi, Jian Zhou, Youchen Tang, Jianjian Yang, and Jingmin Xin. High-precision positioning, perception and safe navigation for automated heavy-duty mining trucks. *IEEE Transactions on Intelligent Vehicles*, 2024.

[11] Eugenio Chisari, Alexander Liniger, Alisa Rupenyan, Luc Van Gool, and John Lygeros. Learning from simulation, racing in reality. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8046–8052. IEEE, 2021.

[12] Chiho Choi, Joon Hee Choi, Jiachen Li, and Srikanth Malla. Shared cross-modal trajectory prediction for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 244–253, 2021.

[13] R. T. H. Collis. Lidar. *Appl. Opt.*, 9(8):1782–1788, Aug 1970.

[14] Christoph Dann, Yishay Mansour, Mehryar Mohri, Ayush Sekhari, and Karthik Sridharan. Guarantees for epsilon-greedy reinforcement learning with function approximation, 2022.

[15] Tim De Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. Integrating state representation learning into deep reinforcement learning. *IEEE Robotics and Automation Letters*, 3(3):1394–1401, 2018.

[16] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[18] Davide Falanga, Suseong Kim, and Davide Scaramuzza. How fast is too fast? the role of perception latency in high-speed sense and avoid. *IEEE Robotics and Automation Letters*, 4(2):1884–1891, 2019.

[19] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

[20] Epic Games. Unreal engine 4.

[21] Siddhant Gangapurwala, Alexander Mitchell, and Ioannis Havoutis. Guided constrained policy optimization for dynamic quadrupedal robot locomotion. *IEEE Robotics and Automation Letters*, 5(2):3642–3649, 2020.

[22] Wei Guan, Zhewen Cui, and Xianku Zhang. Intelligent smart marine autonomous surface ship decision system based on improved ppo algorithm. *Sensors*, 22(15):5732, 2022.

[23] Kıvanç Güçkıran and Bülent Bolat. Autonomous car racing in simulation environment using deep reinforcement learning. In *2019 innovations in intelligent systems and applications conference (ASYU)*, pages 1–6. IEEE, 2019.

[24] Kıvanç Güçkıran and Bülent Bolat. Autonomous car racing in simulation environment using deep reinforcement learning. In *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pages 1–6, 2019.

[25] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[26] James Herman, Jonathan Francis, Siddha Ganju, Bingqing Chen, Anirudh Koul, Abhinav Gupta, Alexey Skabelkin, Ivan Zhukov, Max Kumskoy, and Eric Nyberg. Learn-to-race: A multimodal control environment for autonomous racing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9793–9802, October 2021.

[27] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John Agapiou, Joel Z Leibo, and Audrunas Gruslys. Deep q-learning from demonstrations. In *Annual Meeting of the Association for the Advancement of Artificial Intelligence (AAAI)*, New Orleans (USA), 2018.

[28] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, et al. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17853–17862, 2023.

[29] Zhiqing Huang, Ji Zhang, Rui Tian, and Yanxin Zhang. End-to-end autonomous driving decision based on deep reinforcement learning. In *2019 5th International Conference on Control, Automation and Robotics (ICCAR)*, pages 658–662, 2019.

[30] Muhamamd Ishfaq Hussain, Shoaib Azam, Farzeen Munir, Zafran Khan, and Moongu Jeon. Multiple objects tracking using radar for autonomous driving. In *2020 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, pages 1–4. IEEE, 2020.

[31] Juraj Jagelčák, Jozef Gnap, Ondrej Kuba, Jaroslav Frnda, and Mariusz Kostrzewski. Determination of turning radius and lateral acceleration of vehicle by gnss/ins sensor. *Sensors*, 22(6), 2022.

[32] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[33] Gregory Kahn, Pieter Abbeel, and Sergey Levine. Badgr: An autonomous self-supervised learning-based navigation system. *IEEE Robotics and Automation Letters*, 6(2):1312–1319, 2021.

[34] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[35] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[36] Krisada Kritayakirana and J Christian Gerdes. Autonomous vehicle control at the limits of handling. *International Journal of Vehicle Autonomous Systems*, 10(4):271–296, 2012.

[37] Krisada Kritayakirana and J Christian Gerdes. Using the centre of percussion to design a steering controller for an autonomous race car. *Vehicle System Dynamics*, 50(sup1):33–51, 2012.

[38] Vincent A Laurense, Jonathan Y Goh, and J Christian Gerdes. Path-tracking for autonomous vehicles at the limit of friction. In *2017 American control conference (ACC)*, pages 5586–5591. IEEE, 2017.

[39] Edouard Leurent et al. An environment for autonomous driving decision-making, 2018.

[40] Yiyi Liao, Jun Xie, and Andreas Geiger. Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3292–3310, 2022.

[41] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[42] Bing Liu, Shunying Zu, Hong Wang, Jing Xia, and Naikan Ding. Influence mechanism of visual perception of edge rate lines cycle length on driver's speed. *Transport*, 36:1–8, 08 2020.

[43] Yang Liu, Dingkang Yang, Yan Wang, Jing Liu, Jun Liu, Azzedine Boukerche, Peng Sun, and Liang Song. Generalized video anomaly event detection: Systematic taxonomy and comparison of deep models. *ACM Computing Surveys*, 56(7):1–38, 2024.

[44] Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. Simulated car racing championship: Competition software manual. *arXiv preprint arXiv:1304.1672*, 2013.

[45] Tiange Luo, Chris Rockwell, Honglak Lee, and Justin Johnson. Scalable 3d captioning with pretrained models. *Advances in Neural Information Processing Systems*, 36, 2024.

[46] Spyros Makridakis. The forthcoming artificial intelligence (ai) revolution: Its impact on society and firms. *Futures*, 90:46–60, 2017.

[47] Jianchang Mao and Anil K Jain. Artificial neural networks for feature extraction and multivariate data projection. *IEEE transactions on neural networks*, 6(2):296–317, 1995.

[48] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[49] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[50] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[51] Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, 1990.

[52] Jorge Muñoz, German Gutierrez, and Araceli Sanchis. A human-like torcs controller for the simulated car racing championship. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 473–480, 2010.

[53] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[54] Matthew O'Kelly, , and Rahul Mangharam. F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, 2019.

[55] On-Road Automated Driving (ORAD) Committee. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, jun 2018.

[56] E. Onieva, D. A. Pelta, J. Alonso, V. Milanes, and J. Perez. A modular parametric architecture for the torcs racing engine. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 256–262, 2009.

[57] Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Yajun Fang, Daniel Hoehener, Shih-Yuan Liu, Michael Novitzky, Igor Franzoni Okuyama, Jason Pazis, Guy Rosman, Valerio Varricchio, Hsueh-Cheng Wang, Dmitry Yershov, Hang Zhao, Michael Benjamin, Christopher Carr, Maria Zuber, Sertac Karaman, Emilio

Frazzoli, Domitilla Del Vecchio, Daniela Rus, Jonathan How, John Leonard, and Andrea Censi. Duckietown: An open, inexpensive and flexible platform for autonomy education and research. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1497–1504, 2017.

[58] Zhenghao Peng, Quanyi Li, Ka Ming Hui, Chunxiao Liu, and Bolei Zhou. Learning to simulate self-driven particles system with coordinated policy optimization. *Advances in Neural Information Processing Systems*, 34:10784–10797, 2021.

[59] Jan Quadflieg, Mike Preuss, Oliver Kramer, and Günter Rudolph. Learning the track and planning ahead in a car racing controller. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 395–402, 2010.

[60] Ashish Rana and Avleen Malhi. Building safer autonomous agents by leveraging risky driving behavior knowledge. In *2021 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*, pages 1–6. IEEE, 2021.

[61] Ugo Rosolia and Francesco Borrelli. Learning model predictive control for iterative tasks: A computationally efficient approach for linear system. *IFAC-PapersOnLine*, 50(1):3142–3147, 2017.

[62] Ugo Rosolia and Francesco Borrelli. Learning how to autonomously race a car: a predictive control approach. *IEEE Transactions on Control Systems Technology*, 28(6):2713–2719, 2019.

[63] Ugo Rosolia, Xiaojing Zhang, and Francesco Borrelli. Robust learning model predictive control for iterative tasks: Learning from experience. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1157–1162. IEEE, 2017.

[64] Anirban Santara, Sohan Rudra, Sree Aditya Buridi, Meha Kaushik, Abhishek Naik, Bharat Kaul, and Balaraman Ravindran. Madras: Multi agent driving simulator. *Journal of Artificial Intelligence Research*, 70:1517–1555, April 2021.

[65] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[66] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[67] Klaus Schwab. *The fourth industrial revolution.* Crown Currency, 2017.

[68] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[69] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[70] Yuda Song, Yifei Zhou, Ayush Sekhari, J Andrew Bagnell, Akshay Krishnamurthy, and Wen Sun. Hybrid RL: Using Both Offline and Online Data Can Make RL Efficient. *arXiv preprint arXiv:2210.06718*, 2022.

[71] Yunlong Song, HaoChih Lin, Elia Kaufmann, Peter Dürr, and Davide Scaramuzza. Autonomous overtaking in gran turismo sport using curriculum reinforcement learning. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 9403–9409. IEEE, 2021.

[72] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[73] Stanford Artificial Intelligence Laboratory et al. Robotic operating system, 2018.

[74] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[75] Domen Tabernik and Danijel Skočaj. Deep learning for large-scale traffic-sign detection and recognition. *IEEE transactions on intelligent transportation systems*, 21(4):1427–1440, 2019.

[76] Xiaolin Tang, Bing Huang, Teng Liu, and Xianke Lin. Highway decision-making and motion planning for autonomous driving via soft actor-critic. *IEEE Transactions on Vehicular Technology*, 71(5):4706–4717, 2022.

[77] J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[78] Sen Wang, Daoyuan Jia, and Xinshuo Weng. Deep reinforcement learning for autonomous driving, 2019.

[79] Zhisheng Wang, Yukari Nagai, Dan Zhu, Jiahui Liu, and Nianyu Zou. Based on creative thinking to museum lighting design influences to visitors emotional response levels theory research. In *IOP Conference Series: Materials Science and Engineering*, volume 573, page 012093. IOP Publishing, 2019.

[80] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[81] Trent Weiss and Madhur Behl. Deepracing: A framework for autonomous racing. In *2020 Design, automation & test in Europe conference & exhibition (DATE)*, pages 1163–1168. IEEE, 2020.

[82] Trent Weiss and Madhur Behl. Deepracing: Parameterized trajectories for autonomous racing. *arXiv preprint arXiv:2005.05178*, 2020.

[83] Trent Weiss, John Chrosniak, and Madhur Behl. Towards multi-agent autonomous racing with the deepracing framework. In *International Conference on Robotics and Automation (ICRA)-Workshop on Opportunities and Challenges with Autonomous Racing*, 2021.

[84] Jingda Wu, Zhiyu Huang, Zhongxu Hu, and Chen Lv. Toward Human-in-the-Loop AI: Enhancing Deep Reinforcement Learning via Real-Time Human Guidance for Autonomous Driving. *Engineering*, 21:75–91, 2023.

[85] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 4(6):2, 2000.

[86] Gioele Zardini, Nicolas Lanzetti, Marco Pavone, and Emilio Frazzoli. Analysis and control of autonomous mobility-on-demand systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 5:633–658, 2022.

[87] Wei Zhou, Dong Chen, Jun Yan, Zhaojian Li, Huilin Yin, and Wanchen Ge. Multi-agent reinforcement learning for cooperative lane changing of connected and autonomous vehicles in mixed traffic. *Autonomous Intelligent Systems*, 2(1):5, 2022.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

APPENDIX

SUSTAINABLE DEVELOPMENT GOALS

Degree to which the work relates to the Sustainable Development Goals (SDGs).

Sustainable development goals	**High**	**Medium**	**Low**	**Not applicable**
SDG 1. **No poverty.**				**X**
SDG 2. **Zero hunger.**				**X**
SDG 3. **Good health and well-being.**		**X**		
SDG 4. **Quality education.**				**X**
SDG 5. **Gender equality.**				**X**
SDG 6. **Clean water and sanitation.**				**X**
SDG 7. **Affordable and clean energy.**			**X**	
SDG 8. **Decent work and economic growth.**		**X**		
SDG 9. **Industry, Innovation and Infrastructure.**	**X**			
SDG 10. **Reduced Inequality.**				**X**
SDG 11. **Sustainable cities and communities.**	**X**			
SDG 12. **Responsible consumption and production.**				**X**
SDG 13. **Climate action.**			**X**	
SDG 14. **Life below water.**				**X**
SDG 15. **Life on land.**			**X**	
SDG 16. **Peace and justice strong institutions.**				**X**
SDG 17. **Partnerships to achieve the goal.**				**X**

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Reflexion on the relation of the TFG/TFM with the SDGs and with the most related SDG(s).

This work fits with the United Nations' Sustainable Development Goals (SDGs). In particular, with SDG 9, on "Industry, Innovation and Infrastructure", which aims to "Build resilient infrastructure, promote inclusive and sustainable industrialisation and foster innovation ". And with SDG 11, on "Sustainable cities and communities", which aims to "Make cities and human settlements inclusive, safe, resilient and sustainable".
Within SDG 9 and 11, and other SDG less related (such as SDG 3 and 8) the most related targets to this work are:

- *3.6 By 2030, halve the number of global deaths and injuries from road traffic accidents.*
- *8.2 Achieve higher levels of economic productivity through diversification, technological upgrading and innovation, including through a focus on high-value added and labour-intensive sectors*
- *9.4 By 2030, upgrade infrastructure and retrofit industries to make them sustainable, with increased resource-use efficiency and greater adoption of clean and environmentally sound technologies and industrial processes, with all countries taking action in accordance with their respective capabilities.*
- *11.2 By 2030, provide access to safe, affordable, accessible and sustainable transport systems for all, improving road safety, notably by expanding public transport, with special attention to the needs of those in vulnerable situations, women, children, persons with disabilities and older persons.*

This work lays the foundations to deepen the research in safe and robust autonomous driving in unseen roads. Reducing the risk of accidents (target 3.2). From an industrial point of view, autonomous transport of goods, either between cities in autonomous trucks or between countries in autonomous boats, will both help small businesses in the process of reaching farther clients (8.2) and help the industry increase the sustainability of the merchant lines (9.4). Finally, vehicle autonomy can be applied to the local transport system, either in buses and trains (both regular and underground) which could increase the amount of service available to users and reduce delays and cancellations (11.2).