Politecnico Di Torino

Dipartimento di Elettronica e Telecomunicazioni

Corso di Laurea Magistrale in Ingegneria Elettronica

Master Degree's Thesis

Machine Learning and Big Data Processing in a Human-Vehicle Interaction System

Relatore: Danilo De Marchi

Correlatore: Manolo Dulva-Hina

Autore: Andrea Ortalda

Anno Accademico 2017-2018

A chi da una vita corre al mio fianco.

E che continuerà a farlo. Sempre.

Abstract

Machine Learning and Big Data processing are the key points in the development of an Advanced Driving Assistance System (ADAS) and in general of an Autonomous or Semi-Autonomous vehicle. With the rising of the Internet Of Things (IoT) every object in a road environment will be connected and will interact with all the other components in the scenario. In this scenario, intelligent vehicles will be the linking point for human beings with the general system. Taking all these data coming from the environment, the vehicle will acquire knowledge by means of Machine Learning techniques, in order to improve safety, that is the final goal of the ADAS. In this project all these considerations were taken into account: the creation of a Big Data simulated environment, the processing of these data by means of the Ontologies creation and finally the application of 2 ML techniques thought for obstacle identification, classification and avoidance. This thesis shows all the results achieved, highlighting the previous system problems with the corresponding solutions. As a research thesis, the contribution is related to the ADAS and ML anatomy comparison and analysis, Ontology creation for different scenarios and ML application for the obstacle context. This thesis was a contribution to the artificial intelligence sub-system of a complete ADAS. Final goal of the project is the real implementation of the complete Autonomous Driving system, where the AI sub-system will play a key-role together with a communication intra-vehicle one and a general networking based sub-system.

Ringraziamenti

In primis desidero ringraziare la Mimma e il Taddo per tutto il supporto che mi hanno sempre dato, aiutandomi in ogni aspetto della mia vita. Tutto ciò che sono oggi è solo grazie a loro. Sono stati molto più che un esempio da seguire: sono ciò che io miro a diventare nella vita. Un ringraziamento speciale a mio fratello *Bob* Stefano, che mi ha sempre sostenuto, sopportato e incitato dandomi i più preziosi consigli possibili. Grazie a *Marcolino*, il mio migliore amico, perché avere un gemello diverso nel mondo è qualcosa di unico e bellissimo. Ringrazio gli amici di una vita: *The Bobons* e le *Cacche* per il supporto datomi in questi anni di Politecnico, specialmente durante il mio Erasmus, perché trovare delle persone cos splendide su cui fare affidamento per ogni cosa possibile era al di là di ogni mia più rosea previsione. Grazie *Signo, Sala, Dario, Giamma, Mora, Spita, Febio e Riki.* Ringrazio immensamente i miei nonni *Maria e Mario* e *Angelo ed Elisa*: ogni volta che penso alla parola angelo penso a voi. Un ringraziamento grande come il bene che voglio loro ai miei zii *Laura e Maurizio*, *Fulvio e Carla*, *Aldo e Rosy* e *Nino e Monica*, che negli anni hanno sempre creduto in me crescendomi come se fossi un figlio ed essendo molto più che semplici parenti. Un grazie enorme alle *Mantwins* perché il legame che ci unisce rimane qualcosa di unico. Ringrazio gli scarsi *Gauzo e Savio*, gli asini *Mastro, Messi, Pado e Rinaldi*, i cuginetti *Lollino e Davidino*, il torinese acquisito *Samu*, la bionda *Ferre* e gli amici di Laigueglia *Mat, Ste, Ludo, Amber, Liuk, Silvia e Gigio*: mi avete riempito la vita e vi devo molto. Ringrazio i singori *Zielo e Longano*, *Davide Poggi*, *Andrea Copia* e *Gian* perché hanno reso i miei anni al Politecnico incredibilmente leggeri e pieni di divertimento. Un grazie particolare a *Triple G* e alla *Sarjenta Canti* per aver reso il mio primo semestre a Parigi qualcosa di unico: siete parte di me nonostante la lontananza che ci separa. Grazie alla *Pink Lady Gang: Wouter, Dane e Adrien*: siete stati come dei fratelli per me e spero di rivedervi presto. Grazie a tutti gli amici del residence *Restignat* e in generale a tutti coloro che ho conosciuto durante il mio fantastico periodo parigino. Grazie al prof. Danilo Demarchi per la pazienza e la disponibilità con cui mi ha seguito nel mio Erasmus e nella stesura della tesi e al prof. Manolo Dulva-Hina per tutti i risultati che abbiamo conseguito insieme. Grazie all'equipe dell'ECE de Paris e in particolare ad Aakash e a Francesco per i consigli e i pranzi insieme. Grazie anche al mon ami *Nicholas*, a cui auguro il meglio nella vita. Grazie a *Rebecca, Sha e vale*, perché sono state parti importanti della mia vita. Grazie a Martina e alla famiglia Buso per i bei momenti passati insieme e per aver accolto Stefano come un figlio. Ve ne sono davvero grato. Ringrazio inoltre i professori *Zamboni, Piccinini, Maddaleno, Venesio e Michelone* per avermi fatto capire cosa sia la stima nel mondo accademico. Grazie anche ai miei compagni del liceo per i 5 splendidi anni passati insieme.

Tutto questo non sarebbe stato possibile senza di voi.

Contents

1 Introduction **1**

1.1 Aims and Motivation 2

1.2 Document Structure 3

2 State Of The Art **5**

2.1 ADAS 5

2.1.1 Longitudinal Control 6

2.1.2 Lateral Control 6

2.1.3 Driving Vigilance Monitoring System 7

2.1.4 Parking Assistance 7

2.2 Machine Learning 8

2.2.1 Supervised Learning 10

2.2.2 Unsupervised Learning 10

2.2.3 Reinforcement Learning 12

2.2.4 Deep Learning 13

2.3 ADAS & Machine Learning 16

3 Methodology **19**

3.1 System Structure 19

3.2 Unity 3D 22

3.2.1 Machine Learning Agents 23
3.2.2 TensorFlow & TensorBoard 31
3.3 Ontology 33
3.3.1 Protégé 33
3.3.2 JAVA 34
3.4 Python 35

4 Initial project state 37
4.1 Unity Scenario 37
4.2 Ontology 38
4.3 Machine Learning 41

5 Application & Results 43
5.1 Introduction 43
5.2 Unity Environment 43
5.2.1 Road System 44
5.2.2 Moving Objects 51
5.2.3 Static Objects 57
5.3 General Ontology 61
5.4 Instantiated Ontology 62
5.4.1 General Road Object 62
5.4.2 Close Road Object 64
5.4.3 Obstacle 66
5.5 Supervised Learning 70
5.5.1 Decision Tree 71
5.5.2 K-Nearest Neighbours (KNN) 74

5.5.3 Random Forest 75
5.5.4 Multilayer Perceptron (MLP) 77
5.6 Reinforcement Learning 79
5.6.1 Issue On the Training Scene 80
5.6.2 The Reinforcement Learning scene 81
5.6.3 Results 83

6 Conclusions 91
6.0.1 Results 91
6.0.2 Future Works 93

A Building the Ontology: Data & Methodology 95

B Supervised Data: format & collections 97

C Agent Observations & Training Parameters 99
C.1 Observations 99
C.2 Training Parameters 100

List of Figures

2.1 Longitudinal and lateral control example 7
2.2 Driver vigilance monitoring system 8
2.3 Parking assistance system 9
2.4 Supervised Methods 11
2.5 Reinforcement Learning Model 13
2.6 General Neural Network 14

3.1 Global System Architecture 20
3.2 Unity Logo 23
3.3 ML-Agents structure 25
3.4 Multi-brain agents scenario 27
3.5 TensorBoard Graphic Example 32
3.6 Ontology represented with OntoGraf [1] 35

4.1 Starting Unity Scenario 38
4.2 Instantiated Vehicle Ontology 39
4.3 Instantiated Environment Ontology 40
4.4 Decision Tree for the event recognition 41
4.5 Reward 42

5.1 The new global road 44

5.2 Straight segment 45
5.3 Curved segment 45
5.4 Crossing segment 46
5.5 Traffic Light System 47
5.6 Speed signs 48
5.7 Crossing signs 49
5.8 Stop signs 49
5.9 Work signs 50
5.10 Car SE Path 54
5.11 Car NE Path 54
5.12 Car SO Path 55
5.13 Car NO Path 55
5.14 Pedestrian and generation parameters 56
5.15 Generic Rock 58
5.16 Static Car 58
5.17 Work On The Road 59
5.18 Tree On The Road 59
5.19 General Road Scheme 60
5.20 General Objects Ontology 61
5.21 Environment E and the entire road objects set R 63
5.22 The ontological representation of road object collection R . . . 64
5.23 Unity 3D Close Road Object scenario 65
5.24 Ontological representation of C 66
5.25 Unity 3D Obstacle scenario 68
5.26 Ontological representation of O 70

5.27 Obstacle classification using decision tree. 72
5.28 Obstacle classification using decision tree. 73
5.29 Decision Tree Score. 73
5.30 KNN predictions accuracy result 75
5.31 Random Forest Score 76
5.32 Random Forest Feature Importance 76
5.33 MLP Classifier 78
5.34 MLP Score 78
5.35 Reinforcement Learning scene 82
5.36 Cumulative Reward, carSj scene 85
5.37 Entropy, carSj scene 86
5.38 Cumulative Reward, carSj20 scene 87
5.39 Entropy, carSj20 scene 87
5.40 Cumulative Reward, carSTime25 scene 88
5.41 Entropy, carSTime25 scene 89

List of Tables

3.1 Unity key components 24

5.1 One Traffic Light System period (22 seconds) 47

6.1 Summary of ML results 92

List of Algorithms

1 The *ManageTrafficLight()* function 53

2 The *getObstacles()* function 69

3 The *getLane()* function 71

Chapter 1

Introduction

The Advanced Driver Assistance System (ADAS) is the system thought for helping the driver in the driving process and involved in safety improvement in a vehicle. This system could have several depths of invasion in the driving process, starting from the lightest way (simple warning message) and arriving to the autonomous driving system, where ideally the driver would not act. ADAS is composed of several sub systems, each one concerning a different aspect of the driving experience. For this reason, the Big Data problem raised recently in the ADAS design, considering the big impact of the Internet of Things (IoT), whose continuous amount of data is used by the ADAS subsystems. The term Big Data can be defined as data that becomes so large that cannot be processed using conventional methods, because of their characteristics: *Velocity, Variety, Value, Volume and Veracity.*

1.1 Aims and Motivation

In the last two decades, transportation systems have seen a big transformation from simple mechanical entities to advanced intelligent interconnected platforms. This transformation gave rise to ADAS that are mainly motivated by the following reasons:

- *Safety considerations:* in European Union, the year 2013 has registered 25,400 road traffic accident. Among people killed, 44.7% were car drivers or their passengers and 21.9% were pedestrians with a significant mortality rate gap between low, middle and high-income countries [2].
- *Ecological considerations:* experiments conducted using same vehicle, same cycle and similar traffic conditions showed a gap ranging from -11% to 14% L/100 km of fuel consumption between 33 different drivers.
- *Vehicle market considerations:* the needs of consumers have evolved. The consumers prefer functional source of transportation with options and features to enhance the driving experience.

In the data given by the European Automobile Manifacturers Association (ACEA), it is possible to see that the european automobile industry invests 50.1 billion in R&D annually [3], more than 5% of its total industry turnover. Many important innovators in this field, such FCA, Ford and GM, are at the top with respect to the others leading one. Actually, the automobile industry was in the second place in 2014 for the Research & Development [4], with almost 120 billions $. With the rising of Machine Learning [5] and Big Data

[6], Advanced Driver Assistance Systems (ADAS) have-taken a primal role in the safety industry [7]. Example like Tesla [8] and Waymo [9] self-driven vehicles show the future direction in this field, but at the current state of the art a medium-range vehicle has already some ADAS components embedded in it [10]. Considering that 95% of all accidents are caused by driver error, such as poor anticipation, inappropriate reaction to a hazard and violation of road traffic laws [11], driver's behavior, emotion and distraction analysis has acquired a starring role in the safety domain, giving to ADAS a key role in the automotive field.

1.2 Document Structure

This document is composed of 6 sections and 3 appendices. First section is the introduction, presenting main definitions and motivations. Second section is dedicated to the state of the art, analysing ADAS and Machine Learning anatomies in order to link them. Third section introduces the Methodology, describing all the software and tools used for the project development. Fourth section shows the initial project state, highlighting weaknesses and which part of the previous work was re-used. Section Five is the actual application of the software presented in section three, with the results achieved, Section six presents conclusions and future works. Finally, 3 appendices show the data formats (A and B), while C presents learning parameters used in the Reinforcement Learning application.

Chapter 2

State Of The Art

This chapter reports the work done in the ICACCE article [12], done at *ECE de Paris* with *Dr. Assia Soukane, Mr Moujahid and Prof. Hina.* It presents the ADAS and Machine Learning state of the art, showing the relations between the two anatomies.

2.1 ADAS

Since the development of the crash test in a vehicle, safety improvement is one of the main concerns in the automotive field [13]. In the USA, in 2017 over 37000 people died in road crash accidents [14] and around 90% of the current ones can be attributed to human errors [15]. These errors are normally undetected dangers due to the driver distractions or low reaction time. In such scenario, ADAS is becoming more and more important for big car manufacturers, especially for safety and emission improvement. For these two reasons, many manufacturers are developing an own ADAS. Driving vigilance monitoring system, lateral/longitudinal control and parking assistance are normally the four main components that modern ADAS [10] implement. Machine Learning techniques and Big Data processing in an IoT scenario are the key points in the ADAS development [16]. Main ADAS components description is given below in the following subsections.

2.1.1 Longitudinal Control

Longitudinal control is a sub-system sensor based whose goal is to detect danger situations in front of the car [17] and to control vehicle speed [17]. This is done thanks to sensors embedded on the front part of the car, allowing the system to calculate distance from other vehicles and their speed. The first version of this system was created in 1995 by *Mitsubishi* [18]. At the current state of the art, mid-range cars [19] are still not implementing evasive actions to avoid the dangers detected by this sub-system.

2.1.2 Lateral Control

Lateral control is strictly related to the Longitudinal one: it is a sub-system that uses embedded sensor to detect danger coming from lateral lanes. Figure 2.1 shows the two areas covered by the two sub-systems. Lateral control system is usually composed of 2 sub-systems: *change-lane*, whose goal is to detect danger due to the lane changing actions, and *lane-keeping*, whose goal is to keep the vehicle on the actual lane. As for the longitudinal control, this system is not taking evasive action in order to avoid the dangers.

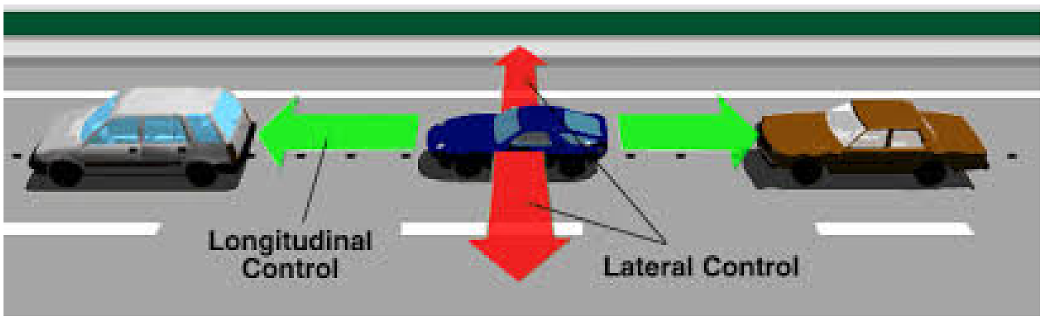


Figure 2.1: Longitudinal and lateral control example

2.1.3 Driving Vigilance Monitoring System

With the raise of the smart phones, distraction analysis is becoming one of the key aspect in the ADAS development [20]. A complete Driving Vigilance Monitoring System would be able to detect not only distraction due to the phone usage, but also stress level in a driver and fatigue situation. In addition, the system would also be able to detect if a driver is drunk or not. At the current state of the art, only high-end car are implementing a basic version of this system: Mercedes-Benz *attention assist* system [21] is an example. This system is based on real-time image recognition, thanks to cameras pointed on the driver face.

2.1.4 Parking Assistance

Parking Assistance system uses lateral and longitudinal sensors and cameras positioned on the vehicle back part to evaluate and operate the parking action. At the current state of the art there are already some working evasive systems, like BMW x6 and Audi A8 that are able to park itself without the driver actions. Mid-range car are only implementing cameras that help the

Figure 2.2: Driver vigilance monitoring system

driver in the action, without the artificial intelligence part. In Figure 2.3 it is possible the see an example of the Parking Assistance system.

2.2 Machine Learning

Machine learning is based on create knowledge based on the previous events encountered by the system [5]. This is thought to emulate the way that humans do: learning from past experience they can build knowledge and transmit it. The common division for the ML world is: Reinforcement Learning, Supervised Learning and Unsupervised Learning. Deep Learning is a particular part of the Machine Learning world: until 2014, it has been considered as a part of Supervised Learning, but Generative Adversarial Networks (GANs) and Variational Auto Encoders (AES) have challenged that view. At the current state of the art, Deep Learning is commonly seen as a set of tools of algorithms that can be used to solve significant problems in Supervised and

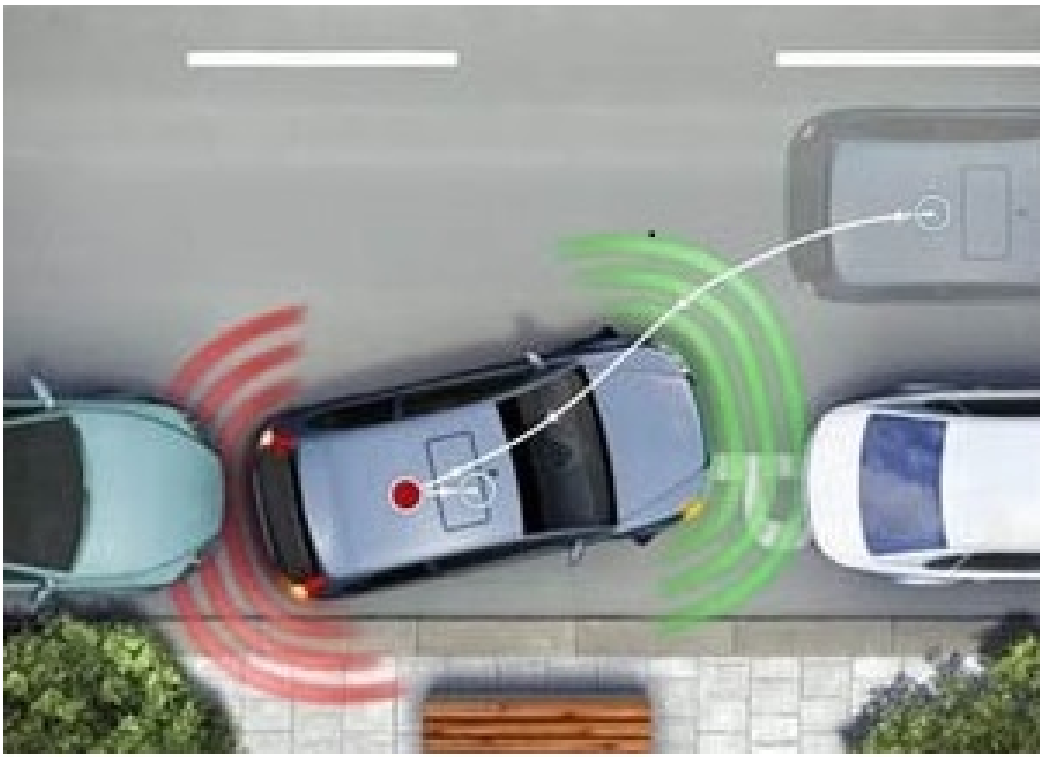


Figure 2.3: Parking assistance system

Unsupervised Learning. Machine Learning is a discipline strictly connected to Data Mining [22] and Big data [6]: nowadays intelligent systems have to manage large Volume, Velocity, Value, Veracity and Variety of Data (Big Data 5V) and find the useful ones in between them (Data Mining) in order to train the system in the best possible way.

2.2.1 Supervised Learning

Supervised Learning is based on knowing both data input and final output, creating a model that can predict a response, reasonable but uncertain, to new data that are coming. The best example is given in [5]: imagine a medic that wants to predict if a person is going to have a heart attack in a determined period of time: he could analyse previous patients and find similar cases. In this way he can be able to give a reasonable prediction (output) about the possibility of the heart attack, knowing the situation of the patient (input). For example age, family history, stress level and other analysis. A more mathematical definition is given in [23]: taking into account an input set, x, a determined output y is given by a function f. Supervised goal is to find another function h that give a similar output for the the given x, knowing in advance what will be the y value.

2.2.2 Unsupervised Learning

Unsupervised Learning basic concept is that it is not possible to know in advance what there is inside the data[5]. Unsupervised Learning is strictly related to the *clustering* and *data mining* concepts: in a unknown data set,

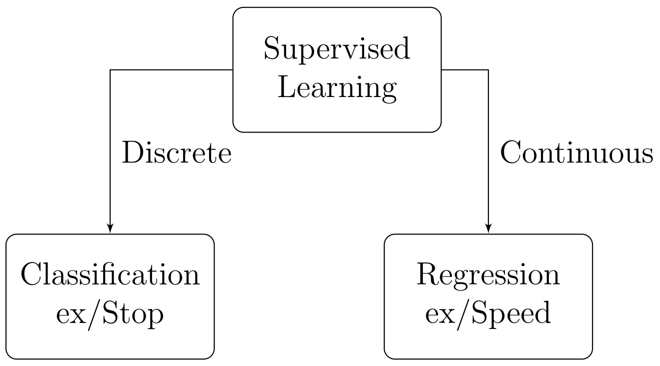


Figure 2.4: Supervised Methods

the machine has to find a possible classification way extracting common and useful features. Basic point and main difference with respect to Supervised is that the data structure is not known and Unsupervised final goal is basically to find one of the possible structures for the data. From a mathematical point of view the input data set X correspond to infinite f, so the goal is to find the best one to create a useful y set. An example of a unsupervised application is the description of what there is inside a jungle: tree, plants, animals and so on. One way of acting is to divide and cluster different kind of animals: spiders, monkeys, snakes and so on. Key point is that the system does not know what the jungle contains; the system will just see that the animal features are different with respect to the ones that characterize the plants: for example they can move; the decision of calling them "animals" is only one of the infinite possible classification method. It is because of all these considerations that it is important to talk about feature extraction: it is the science of extract and detect the most important and useful features in a scenario. At the end the system will find unseen patterns or labels, extracting common features between all the individuals that are being considered.

Taking a look at the previous example, the ability of moving is a feature for animals and humans that has been extracted for this application [24].

2.2.3 Reinforcement Learning

Reinforcement Learning is slightly distant from the other Machine Learning techniques. If SL and UL have something in common, RL is based on a different concept: in a particular ruled-world, named *environment*, an *agent* has to take some *actions*. These *actions* led it into a particular *state* with a numerical *reward* [25]. A good example of how reinforcement learning works is to think about a chess game: if the final goal is very clear, the agent does not exactly if its moves, named *action a*, are good or bad until he reaches the end of the game. He only knows that some actions a lead him to some situations of the environment, named *state s*, and every time he performs an action a that has lead him into a state s he obtained a *reward r*. Key point in RL is that the *agent* knows the rule of the *environment*, but does not know in advance which *actions* give positive *reward*. At the end of the training, the *agent* should be able to take the *actions* that will give it the highest *reward*.

From a mathematical point of view, Markov decision process (MDP) are the best way to formalize a RL problem [25]. Markov decision process allows to describe and represent decision-making problems and solutions. Basically, a MDP is composed of 5 elements:

- **S** is the finite set of states.
- **A** is the possible set of action available for a state **S**.

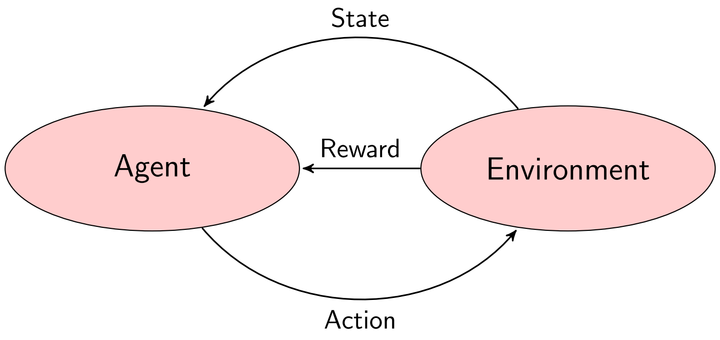


Figure 2.5: Reinforcement Learning Model

- $\mathbf{T}(\mathbf{S},\mathbf{A},\mathbf{S'});\mathbf{P}(\mathbf{S'}|\mathbf{A},\mathbf{S})$ T represents the environment model: it is a function that produces the probability P of being in state *S'* taking action A in the state S.

- $\mathbf{R}(\mathbf{S},\mathbf{A},\mathbf{S'})$ is the reward given by the environment for passing from *S'* to S as a consequence of A

- $\pi(\mathbf{S};\mathbf{A})$ is the policy of the state i.e. the solution of the problem that takes as input a state S and gives the most appropriate action A to take.

Other 2 key property in the MDP are the *Markovian property* and the *stationarity property*. The former is simply assuring that previous states do not influence current state S. The latter instead assures that during the process these properties remain the same.

2.2.4 Deep Learning

Deep Learning is a way to create and represent artificial intelligence. The most used techniques in Deep Learning is the Artificial Neural Network

(ANN). ANN are inspired by the human and animal brain structures, i.e. a network that is composed by many neurons that communicate with each other. An ANN could be composed of several layers, depending on the model depth or complexity. The depth of a model can be usually defined in two different ways: the former consists in evaluating the longest path during the computation, the latter in considering the depth of the relation between layers [26]. Usually in the ANN world a network depth is evaluated on the worst critical path or counting the number of hidden layers. In between the input and output layers, several intermediate layers could be present, which are responsible of the deep learning algorithm implementation. A general NN is shown in Figure 2.6

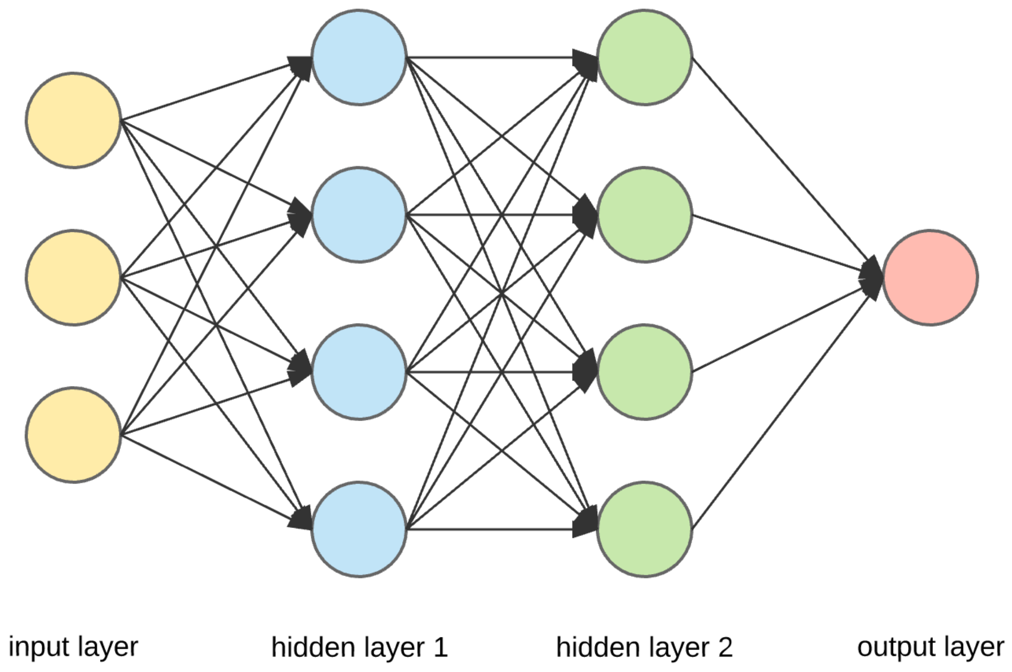


Figure 2.6: General Neural Network

Neural network output computations are based on the conception of weights and activating function [27]. The activating function is a concept strictly related to the threshold one: if the computation of the input gives a value greater than the threshold, the activation function will give a result, otherwise another one is produced. The activation function could be complex of very simple, depending on the network application. Weights play a key role in the output computation, because the network has to take into account more than one inputs and some of these may have a greater impact on the final computation. This is also a key point in the learning process of the network: if the input produces an incorrect output, the correspondent weight is reduced. This process is also known as "back propagation". From an algebraic point of view, defining $\mathbf{w_i}$ as the weight that the node i has and $\mathbf{x_i}$ as the ith input:

$$output = \begin{cases} 0 \text{ if } \sum_i x_i w_i \leq threshold \\ 1 \text{ if } \sum_i x_i w_i > threshold \end{cases} \tag{2.1}$$

The most used example of neural networks application is about image recognition: for example we can give many image examples about a vehicle and the machine will learn to recognize other vehicle images without knowing how a vehicle works or what are its main components. The network learn from past experience (example images) changing the input weights when a new image is given as input.

2.3 ADAS & Machine Learning

- Supervised Learning application in ADAS is strictly related to longitudinal and lateral control. These two systems in fact face classification problems, such as incoming cars from other lanes or sudden deceleration actuated by vehicle in front of the current car. Significant examples of SL in ADAS can be found in [28] and [29]. Parking assistance system and driving vigilance monitoring are more works of Reinforcement and Deep Learning, also if Supervised and Unsupervised can be used as preprocessing phase for featuring extraction and reduction.

- Deep Learning main application is related to the world of the image processing and recognition. Using ANN, Deep Learning algorithm are able to analyse and classify different type of roads and situation: for these reason the Driving Vigilance Monitoring System uses DL algorithms for the driver face analysis. DL can also play a key role in the parking action: back cameras can be used to evaluate in a precise way the distances of the parking spot, building a much clearer scenario.

- RL is used wherever a decision has to be taken. For this reason, at the current state of the art it is only involved in the parking action, since it is the only system that implements evasive actions, but longitudinal and lateral control are moving in the same direction. The parking action can be formalized as a MDP: *environment, state, agent and reward* are respectively the parking spot, the vehicle position, the vehicle itself and the correct positioning during the action.

- As mentioned in the introduction, with the raise of the IoT, the Big Data concept has assumed a key role in the ML applications. For this reason, feature selection is a mandatory pre-processing phase, as happened in Arash Jahangiri and Hesham A. Rakha [30] work or in C. Miyajima et al. [31] one. UL finds its main applications in this field, since its goal is to label and cluster huge amount of raw data. It can find usages in every ADAS sub-system, since at the current state of the art each of them has to manage large volume of data with different characteristics.

Chapter 3

Methodology

3.1 System Structure

As already mentioned in chapter 1, this project is a contribution to the *ECE de Paris* research project *Autonomous Smart Secured Interactive Automobile (ASSIA)*. It is developed in collaboration with *Laboratory of Insitut Universitarire de Technologie of Versailles* (LISV) and has as contributors PhD researchers and master degree's students. The entire project is about the construction of a complete intelligent ADAS, looking at new fields of the technology like 5G communications intra-vehicles and Machine Learning. This work is a contribution to the construction of the artificial intelligence part.

The scheme representation of the global ADAS architecture is shown in Figure 3.1. In particular, this work will focus in the area delimited by the dotted rectangle, which is the artificial intelligence part in the global system.

The entire ASSIA system, interacting with the environment, is then composed of 4 main components:

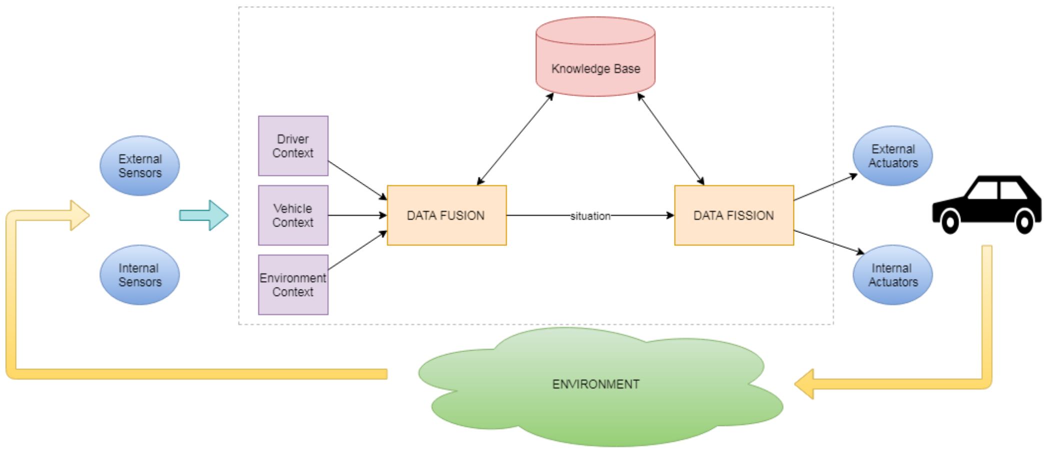


Figure 3.1: Global System Architecture

- Sensors: A sensor for definition is a ”transducer that transforms a physical property into an electrical measure, which one desires to digitize.” The sensors are the components that generate the system input data set, capturing informations about the environment. This work will not focus on real sensor analysis, but on simulated one. Since the creation of instantiated ontologies, supervised and reinforcement applications need data, this work will consider the input from sensor as given, working on simulated one. More detail on the data processing and collection are given in the 3 appendix:

  - A: concerning the ontology data processing.
  - B: concerning the Supervised Learning data processing.
  - C: concerning the Reinforcement Learning agent observation .

- Context: All the data received can be divided into 3 different contexts, representable by means of Ontologies. This work will main focus on the *environment context*, creating the specific ontologies thanks to the sensor data. The three contexts are:

  - *Driver context*: it stores all the information about the driver situation, as stress level, mental state and fatigue level. This work will not focus on this context, but in [32] a related work was done at *ECE*.
  - *Vehicle context*: it stores all the data concerning the vehicle itself, where the ASSIA system is installed. Main components here are vehicle velocity, distances from other objects, acceleration and break pedal usage.
  - *Environment context*: this is the biggest and most various data set. It contains all the informations coming from the external world and IoT: communication intra-vehicles, other objects positions and physical properties.

- Knowledge Base: a *Knowledge Base (KB)* is where the system knowledge is sorted, in order to be used by the computer for the applications. The main difference with respect to a normal database is that , starting with general possessed data, it will be able to merge them creating new knowledge. All of this can be done thanks to the reasoning process existing in the system. The KB communicates with two system components:

  - *Data Fusion*: it consists in the combination of different data coming from different sensors and contexts. This phase is mainly a work of classification: this work will be focused on a Supervised Learning application where the obstacles are classified looking at physical properties.

  – *Data Fission*: once the data fusion phase is completed, its outputs will be the fission input. In this phase the system, with the KB help, will decide and actually build the best actions, sending signals to the actuators in order to perform these actions. In this work this phase is performed by mean of Reinforcement Learning, where the obstacle avoidance is performed.

- Actuators: These components are where the decisions taken in the *Fission* phase are actually performed, influencing the environment. Steering, accelerate and break actions are the best examples. As for the sensor, this work will interact with a simulated environment, in which the car will detect, classify and avoid an obstacle. The actions performed would be the same of a future real-case scenario, in which the agent will be able to accelerate, steer and break in order to avoid obstacles.

3.2 Unity 3D

Unity is a cross-platform game engine developed by Unity Technologies [33], which is primarily used to develop both three-dimensional and two-dimensional video games and simulations for computers, consoles, and mobile devices. In the research field softwares like unity are used to simulate systems before the real implementation.

Figure 3.2: Unity Logo

Unity 3D possibility to write into JSON standard *csv* files, collecting real world data like velocity or size, allows to create good simulations and testing environment. In table 3.1 is it possible to see the most important components usually used when a scene is created.

3.2.1 Machine Learning Agents

Unity Machine Learning Agents (ML-Agents) is an open-source Unity plug-in that enables games and simulations to serve as environments for training intelligent agents. Since the Supervised Learning can be done off-line, analysing the data collected in advance, the Reinforcement Learning is most of the time an on-line application, because the actions taken by the agent have to receive a reward in real time in order to update the state.

The ML-Agents is composed of 6 components, that allow the on-line implementation of an intelligent systems:

1. Academy.
2. Brain.
3. Agent.

Component	description
Collider	Allows physical collisions with other objects and the possibility to detect them. Useful for the creation of moving object interactions.
Rigibody.velocity	In a 3D scene, allows to have a real-time 3-dimensional Vector storing the object velocity.
Transform	Define the position, the rotation and the scale of a game object in the scene. Each of this component is a 3-D Vector (x-y-z).
C# Script	Allows to define variables, influence other components and communicate with other game objects.
Cameras	Can be attached to an object in order to follow it. A scene can have several cameras, so that it is possible to monitor different game objects.
Child/Parent	A game object can have a child, becoming its parent. This follows the common class/subclass relation in an object oriented environment.
Tag	Allows to group game objects, in order to collect them in the scripts

Table 3.1: Unity key components

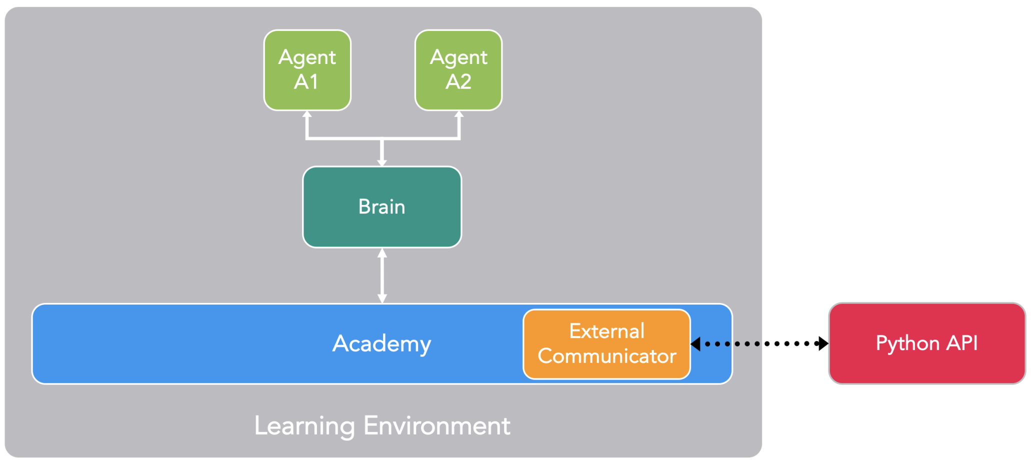


Figure 3.3: ML-Agents structure

4. Python API: It is, with the external communicator, the component that allows the communication between the learning algorithms and the learning environment components.

5. External communicator: Unity provides a socket implementation for the communication with the Python API. It is embedded in the Academy object and it is responsible for the learning data exchange with the API, sending the decision made by the brain and receiving the learning algorithm updates.

6. Environment: It is the totality of the Unity scene, composed of both ML-Agents components and normal Unity editor game object. This big environment allows the learning of a self-created Unity project, whose application can be very different.

Academy

The Academy is the object in charge of managing the other two key objects: the brain and the agents. Its main roles are to initialize the environment, reset the scene when needed and manage the changes in the scene. For example in a scenario in which when an agent reaches the target, this should be randomly reallocated; this action should be managed by the Academy. The Academy manages also the scene dimensions where training, the quality of the graphic and the frame number. Its main methods are:

- *InitializeAcademy()* : method for any logic normally performed in the standard Unity *Start()* or *Awake()* methods. These methods allow the event to start and the object management.
- *AcademyReset()* : method called when the environment has to be reset.
- *AcademyStep()* : function that is called at every simulation step, before any agents are updated.

It is important to notice that a scene can have multiple brains and agents, but it has always only one Academy, that handles all the brains and agents interactions and behaviour. In Figure 3.4, the multi-brain agents scenario is shown.

It is important to notice that a brain can be attached to different agents, while there is only a single academy, that has a specific external communicator that communicates with only a Python API. The brains moreover can be of different types: a combination of external and internal brains is

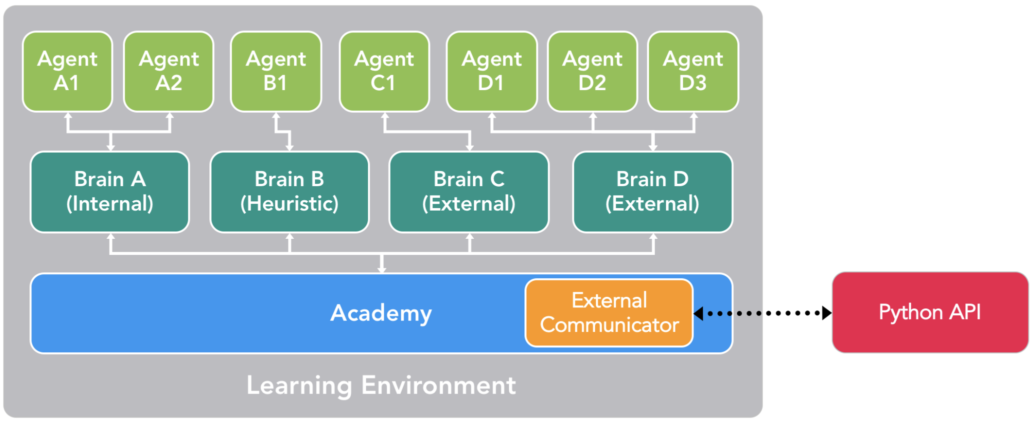


Figure 3.4: Multi-brain agents scenario

possible, allowing the training phase while an internal brain is run to show the training result.

Brain

The brain is the component that encapsulates the decision making process. It decides what action the agent has to take, thanks to the data coming from the agent and the learning algorithm. It has 3 really important parameters:

1. Vector Observation: It is a Vector containing the observation made by the agent. The brain will analyse these observations and take actions thanks to these data. Observations usually are distances from objects, agent velocity and other physical measures.

2. Vector Action: It contains all the possible actions that can be given to an agent. For example a car agent could have 3 actions: steer, break or accelerate. The brain at every step will tell to the agent how much it has to steer, break or accelerate according to its decisions.

3. Type of brains: Currently there are 4 brain types:

    - Player: No training phase has been done. It allows the player to take actions in order to test the reward function, the agent observations and if the academy reset the scenario in the correct way. In this phase, the learning algorithm is not playing any role.
    - External: In order to train the agent, the brain has to be set as External. That means that the brain will take the rewards from an external learning algorithm, sending to the agent the action taken looking at the rewards. Setting the brain as external allows the brain to receive the vector observation from the agent, following the general scheme of a reinforcement learning scenario. An External brain simply passes the observations from its agents to an external process and then passes the decisions made externally back to the agents.
    - Internal: Once the training phase is completed, ML-Agents plug-in creates a scheme using TensorFlow [34] (an open-source software developed by Google Brain). The internal brain will read the training results and actually take decision, sending the actions by the agent. An Internal brain uses the trained policy parameters to make decisions (and no longer adjusts the parameters in search of a better decision).
    - Heuristic: It is an experimental brain that allows the programmer to code by hand an agent's decision making process. An Heuristic brain requires an implementation of the Decision interface to

which it delegates the decision making process.

Agents

An agent is the Reinforcement Learning actor. Its main role in a ML-Agents scene is to collect observations from the environment and pass them to the brain object. The most important coding part in the agent creation is the reward function, that allows to estimate the value of the agent's current state toward accomplishing its tasks. An agent passes its observations to its brain. The brain, then, makes a decision and passes the chosen action back to the agent. Agent scripts are based on two key functions:

- *CollectObservation()* : To make decisions, an agent must observe the environment to determine its current state. A state observation can take the following forms:
  - Continuous Vector : A feature vector consisting of an array of numbers.
  - Discrete Vector : An index into a state table (typically only useful for the simplest of environments).
  - Visual Observations : one or more camera images. For this type of observation the creation of a camera object attached to the agent is required.

  It is important to notice that for more stable training processes a normalization of the observation is needed. This means that the values should be [-1, 1]. For example a possible code for observe a ball velocity could be:

```
public GameObject ball;


public override void CollectObservations()
{
    Vector3 speed = ball.transform.
    GetComponent<Rigidbody>().velocity;
AddVectorObs(speed.x / maxVelocity);
AddVectorObs(speed.y / maxVelocity);
AddVectorObs(speed.z / maxVelocity);
}
```

Normally the observations are distances between objects, velocity, collider bounds and object positions.

- *AgentAction(float[] vectorAction)* : the vector *vectorAction* contains all the actions that an agent can take and it can be of two types:
  - Continuous : When an agent uses a brain set to the Continuous vector action space, the action parameter passed to the agent's *AgentAction()* function is an array with length equal to the Brain *Vector Action Space Size* property value. The individual values in the array have whatever meanings that the programmer ascribe to them. If one element in the array is assigned as the speed of an agent, for example, the training process learns to control the speed of the agent though this parameter.
  - Discrete : When an agent uses a brain set to the Discrete vector

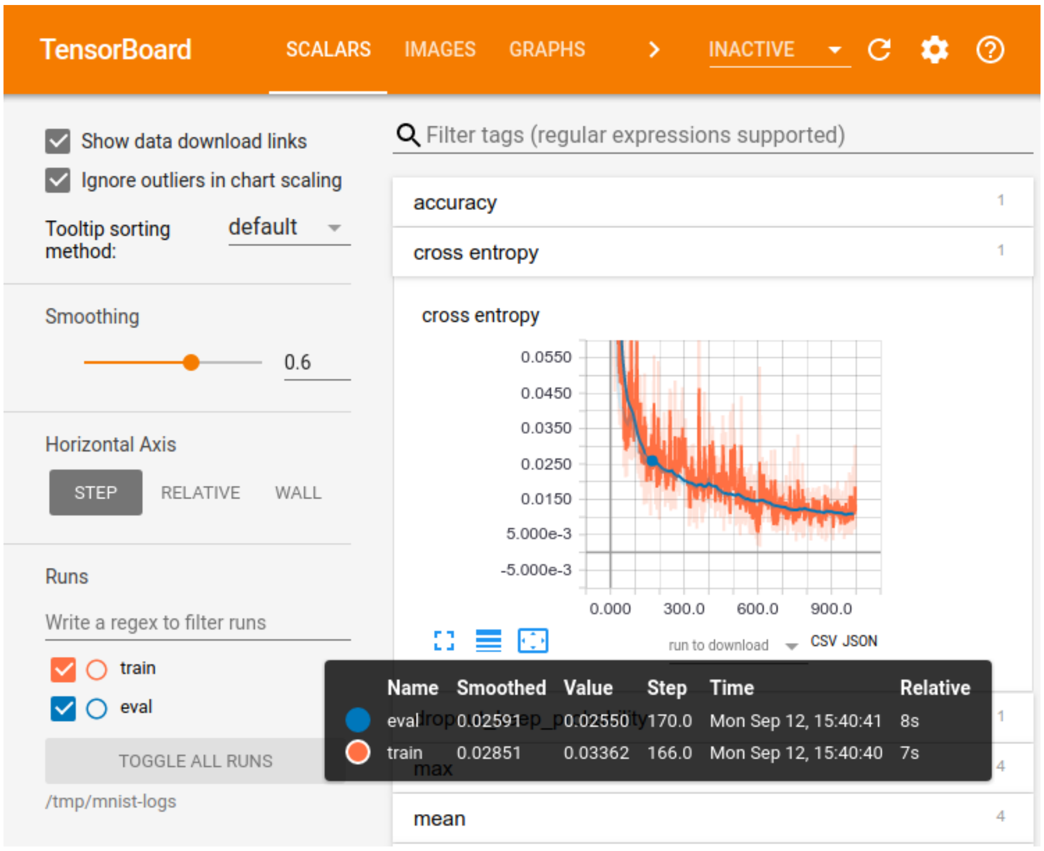
action space, the action parameter passed to the agent's *AgentAction()* function is an array containing a single element. The value is the index of the action in the table or list of actions. With the discrete vector action space, *Vector Action Space Size* represents the number of actions in your action table.

Actions are influenced by the most important parameters in the reinforcement learning scenario: the *Reward*. It is possible to allocate rewards to an agent by calling the *AddReward()* method in the *AgentAction()* function. The reward assigned in any step should be in the range [-1,1], otherwise this could lead to an unstable training. The reward value is reset to zero at every step.

3.2.2 TensorFlow & TensorBoard

TensorFlow [35] is an open source software library for high performance numerical computation. It has its main usage in the ML and DL world, originally was created by the Google's AI organization. The flexible numerical computation core is used across many other scientific domains, as Big Data and Data Mining. As Machine Learning is becoming more and more important, TensorFlow is following the same trend, with several innovating companies that started to use the software: Goggle, AMD and Intel are some examples. The ML-Agents plug-in uses this software to compute the learning results, analysing at each step the parameters of the application.

TensorBoard is a TensorFlow tool for graphical visualization. TensorBoard operates by reading TensorFlow events files, which contain summary data that could be generated when running TensorFlow. In figure 3.5 is it possible to see a fully configured scenario for the graphical visualization of the learning system entropy.



✓ Show data download links

✓ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis [✓ STEP] RELATIVE WALL

Runs

✓ train

✓ eval

	Name	Smoothed	Value	Step	Time	Relative
●	eval	0.02591	0.02550	170.0	Mon Sep 12, 15:40:41	8s
●	train	0.02851	0.03362	166.0	Mon Sep 12, 15:40:40	7s

Figure 3.5: TensorBoard Graphic Example

TensorBoard is a powerful tool, in which it is possible to plot simple 2-D graphics showing the evolution of the learning parameters, but also more complex graphs showing the system structure and debug it. In addition, it is possible to create Histogram displaying how the distribution of some Tensor in the chosen TensorFlow graph has changed over time. It does this by showing many histograms visualizations of the tensor at different points in time.

3.3 Ontology

Ontology is an object-oriented way of representing an environment. It is a tool used not only for scientific systems, but also for other domains. Philosophy, artificial intelligence, systems engineering, information management, computational linguistics, and cognitive psychology could use ontologies for the *knowledge representation* [36]. *Knowledge representation* is a field of artificial intelligence that tries to deal with the problems that surround the system design and the usage of formal languages suitable for capturing human knowledge. Any artificial intelligence is dependent on knowledge, and thus a representation of that knowledge in a specific form is mandatory. The usage of ontologies is an understandable, fast and efficient way of doing that. Ontologies found their most famous application in Semantic Web, where Web pages are annotated by ontology-based meta-data. The W3C Web Ontology Language (OWL) is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things. In this project OWL will be used first for the generic environment representation, second to build more specific ones useful for the obstacle detection.

3.3.1 Protégé

Protégé is a free, open-source ontology editor and framework for building intelligent systems [37]. It is a software used to build and represent ontology in OWL. The Protégé most used tools are:

- OWLViz: used for representation has graph.

- OntoGraf: used to instantiate class objects with individuals, subclasses and properties
- VOWL: Visual Notation for OWL Ontologies (VOWL) provides graphical representations for elements of the OWL Web Ontology Language.

The OntoGraf tool allows to create specific ontologies that represent specific environments with defined individuals belonging to generic classes. Every individual will then have some private and unique properties, instantiated thanks to the data collected. In Figure 3.6 it is possible to see an instantiated ontology with OntoGraf, taken by the article for the KEOD conference [1]. Here the *Rock* object is a *subclass* of the generic *class* thing. *Rock* class has a unique individual with unique properties; in this case

- HasLanePosition
- HasPhysics

are the two simple properties characterizing the object, with some individuals belonging to these properties.

3.3.2 JAVA

JAVA is a general purpose, object oriented programming language [38]. In this project NetBeans Integrated Development Environment (IDE) for Java is used for the formal creation of the ontologies, through the OWL specification. The program written in JAVA will be able to read the *csv* file produced in Unity and extract the data contained in it. Thanks to these data, the

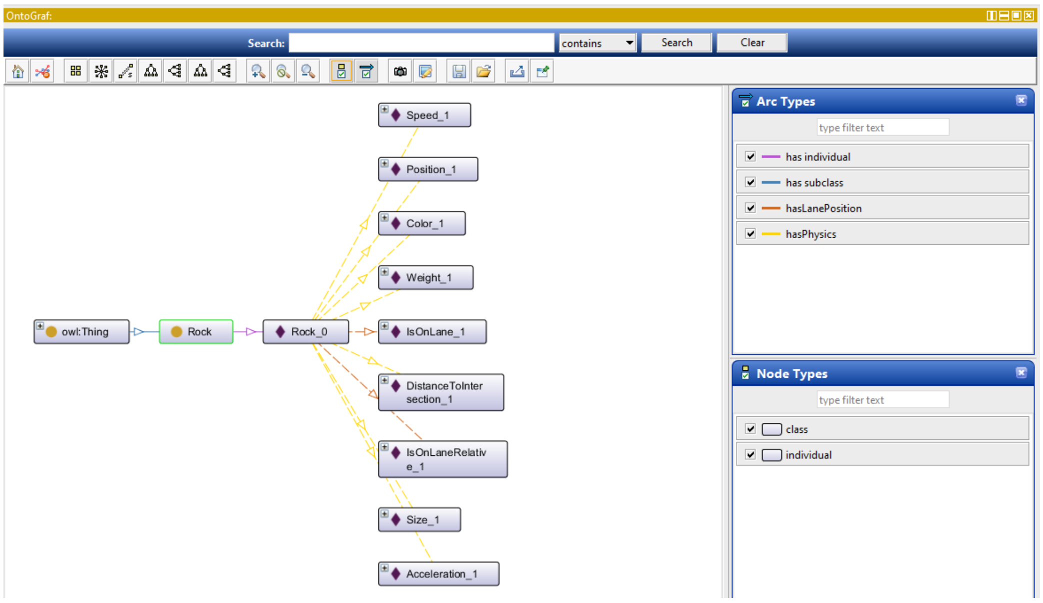


Figure 3.6: Ontology represented with OntoGraf [1]

program will create an ontology for each frame caught in Unity. The program has a server-client implementation for future on-line application, such as Reinforcement Learning. The server reads the *csv* file and send a single frame, while the client is able to extract data from the frame and create the correspondent ontology.

3.4 Python

Python is an interpreted high-level programming language for general-purpose project. It is based on the the idea that the programming languages have to be intuitive, simple and understandable. In the 1999 document *The Zen of Python*, these concept are highlighted:

- Beautiful is better than ugly.

- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.

Despite the simplicity goal, python is a powerful language with libraries dedicated to Machine Learning applications. The most important library in this domain is *Scikit-Learn*, usable mainly for data mining and data analysis. It implements several learning algorithms for *Classification, Regression and Clustering* such as *Support Vector Machine (SVM), Decision Tree and Random Forest. Scikit-Learn* is mainly used for Supervised Learning and Unsupervised Learning applications, while for Reinforcement Learning it does not exist a dedicated library because of the application variety and software usage.

Chapter 4

Initial project state

This work is a contribution in the ASSIA intelligent system, whose starting state is presented in this chapter, together with the issues that it was presenting. As for the whole work, this chapter is composed of 3 parts, in order to highlight the different progresses achieved: Unity Scenario, Ontology and Machine Learning.

4.1 Unity Scenario

The unity scenario was composed of a global road, one static car, one moving car, two pedestrian crossings and 2 stop signs. A stylized scheme is shown in Figure 4.1.

The moving car was only able to turn to its left and then go straight, behaving in a non natural way after the turn. The controlled vehicle was complete for the application, since it was able to record and write the data in a *csv* file. Since the application was the event recognition and classification,

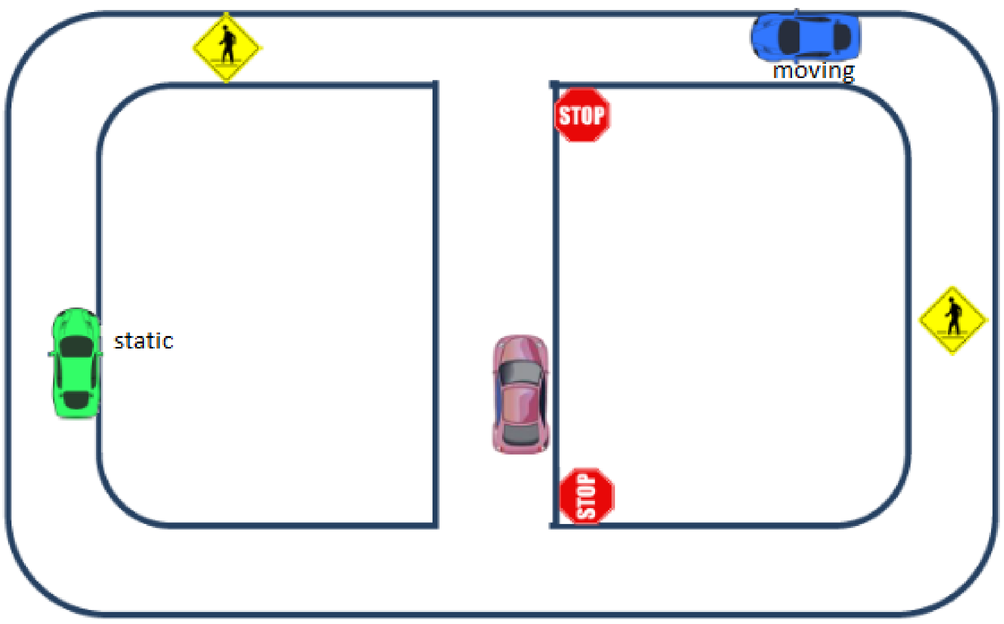


Figure 4.1: Starting Unity Scenario

the most important data, written also in the *csv* file were:

- Global road building, including in which road segment the car was.
- Vehicle physical properties, such as velocity, position and orientation.

The main issue in this part was the lack of road objects, the road dimension and the strange behaviour of the moving car. In addition, some changes had to be done in the data collection phase, since the system was not including an object collection part. However, the work done for the road building an data recording was used in this work application.

4.2 Ontology

The global structure of the JAVA application for the ontology creation was completed. In the same way that this project is thought, the single ontology instantiation and global creation were separated. The main issue were the coding errors in the JAVA programs, with the server-client application that did not work in the correct way. The ontology merging process developed to create general ontology was re-utilised, with improvement regarding the road object. The instantiated ontology creation process was fixed and amplified, adding the creation of the needed individuals. In Figure 4.2 it is possible to see the instantiated ontology used in [39] for the vehicle and in Figure 4.3 for the environment.

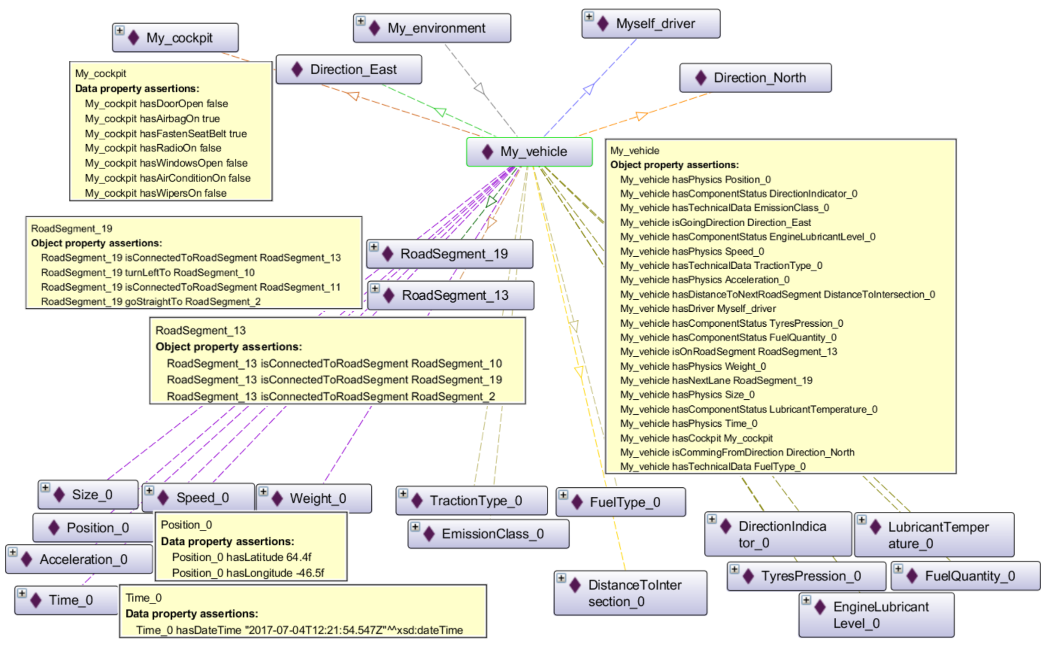
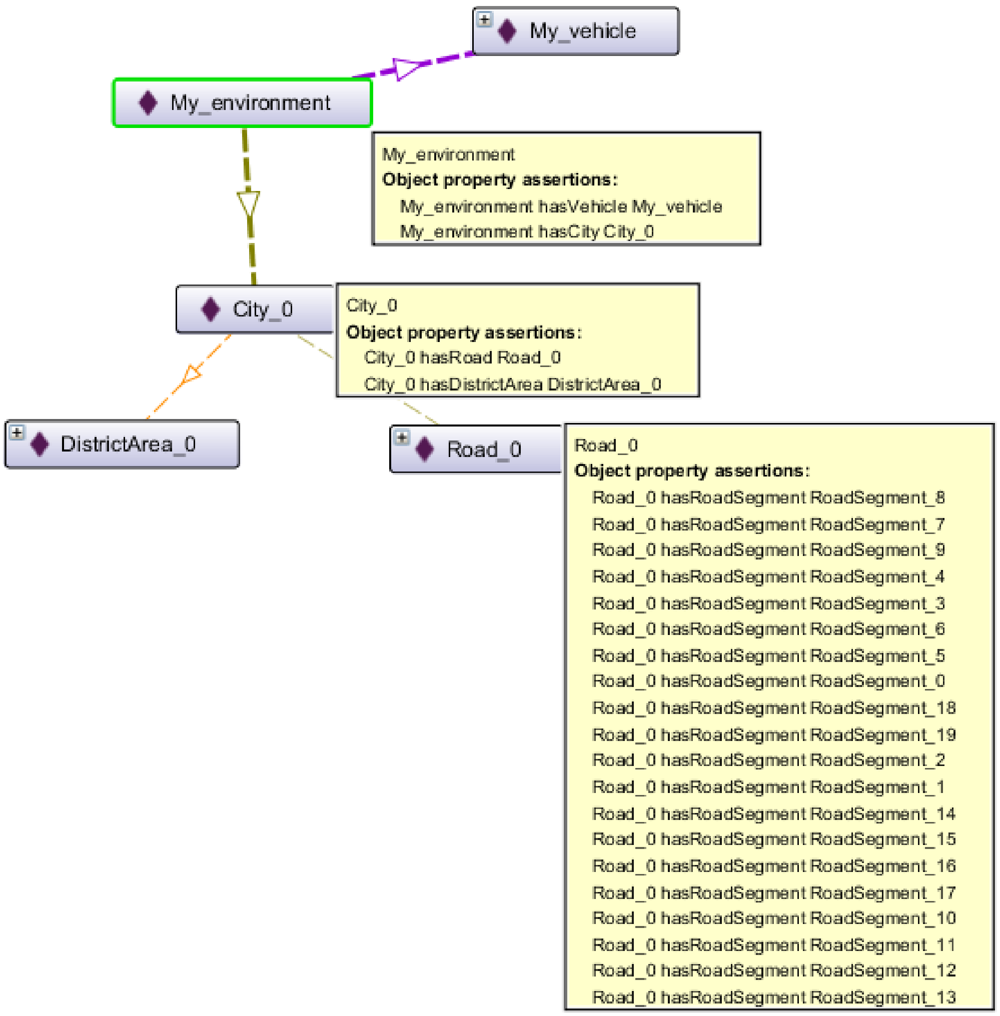

My_cockpit
My_environment
Myself_driver
Direction_East
Direction_North
My_vehicle
My_cockpit
Data property assertions:
My_cockpit hasDoorOpen false
My_cockpit hasAirbagOn true
My_cockpit hasFastenSeatBelt true
My_cockpit hasRadioOn false
My_cockpit hasWindowsOpen false
My_cockpit hasAirConditionOn false
My_cockpit hasWipersOn false
My_vehicle
Object property assertions:
My_vehicle hasPhysics Position_0
My_vehicle hasComponentStatus DirectionIndicator_0
My_vehicle hasTechnicalData EmissionClass_0
My_vehicle isGoingDirection Direction_East
My_vehicle hasComponentStatus EngineLubricantLevel_0
My_vehicle hasPhysics Speed_0
My_vehicle hasTechnicalData TractionType_0
My_vehicle hasPhysics Acceleration_0
My_vehicle hasDistanceToNextRoadSegment DistanceToIntersection_0
My_vehicle hasDriver Myself_driver
My_vehicle hasComponentStatus TyresPression_0
My_vehicle hasComponentStatus FuelQuantity_0
My_vehicle isOnRoadSegment RoadSegment_13
My_vehicle hasPhysics Weight_0
My_vehicle hasNextLane RoadSegment_19
My_vehicle hasPhysics Size_0
My_vehicle hasComponentStatus LubricantTemperature_0
My_vehicle hasPhysics Time_0
My_vehicle hasCockpit My_cockpit
My_vehicle isCommingFromDirection Direction_North
My_vehicle hasTechnicalData FuelType_0
RoadSegment_19
Object property assertions:
RoadSegment_19 isConnectedToRoadSegment RoadSegment_13
RoadSegment_19 turnLeftTo RoadSegment_10
RoadSegment_19 isConnectedToRoadSegment RoadSegment_11
RoadSegment_19 goStraightTo RoadSegment_2
RoadSegment_19
RoadSegment_13
RoadSegment_13
Object property assertions:
RoadSegment_13 isConnectedToRoadSegment RoadSegment_10
RoadSegment_13 isConnectedToRoadSegment RoadSegment_19
RoadSegment_13 isConnectedToRoadSegment RoadSegment_2
Size_0
Speed_0
Weight_0
TractionType_0
FuelType_0
Position_0
Position_0
Data property assertions:
Position_0 hasLatitude 64.4f
Position_0 hasLongitude -46.5f
EmissionClass_0
DirectionIndicator_0
LubricantTemperature_0
Acceleration_0
TyresPression_0
FuelQuantity_0
DistanceToIntersection_0
Time_0
Time_0
Data property assertions:
Time_0 hasDateTime "2017-07-04T12:21:54.547Z"^^xsd:dateTime
EngineLubricantLevel_0

Figure 4.2: Instantiated Vehicle Ontology

It is important to notice that the ontology describing the vehicle was much more complete with respect to the one describing the environment. As far as this work will be focused on the obstacle detection, classification and avoidance the vehicle ontology will not be modified, but the environmental one will be expanded with individuals and properties.



My_environment
Object property assertions:
My_environment hasVehicle My_vehicle
My_environment hasCity City_0

City_0
Object property assertions:
City_0 hasRoad Road_0
City_0 hasDistrictArea DistrictArea_0

Road_0
Object property assertions:
Road_0 hasRoadSegment RoadSegment_8
Road_0 hasRoadSegment RoadSegment_7
Road_0 hasRoadSegment RoadSegment_9
Road_0 hasRoadSegment RoadSegment_4
Road_0 hasRoadSegment RoadSegment_3
Road_0 hasRoadSegment RoadSegment_6
Road_0 hasRoadSegment RoadSegment_5
Road_0 hasRoadSegment RoadSegment_0
Road_0 hasRoadSegment RoadSegment_18
Road_0 hasRoadSegment RoadSegment_19
Road_0 hasRoadSegment RoadSegment_2
Road_0 hasRoadSegment RoadSegment_1
Road_0 hasRoadSegment RoadSegment_14
Road_0 hasRoadSegment RoadSegment_15
Road_0 hasRoadSegment RoadSegment_16
Road_0 hasRoadSegment RoadSegment_17
Road_0 hasRoadSegment RoadSegment_10
Road_0 hasRoadSegment RoadSegment_11
Road_0 hasRoadSegment RoadSegment_12
Road_0 hasRoadSegment RoadSegment_13

Figure 4.3: Instantiated Environment Ontology

4.3 Machine Learning

In the previous work done on the ASSIA system the two Machine Learning applications were made on the event recognition, classification and action decision. The Supervised Learning showed good result in the event classification: the *turn left, turn right, stop and normal* actions were classified with a mean score of 94.56% for the Decision Tree and 94.07% for the KNN. In Figure 4.4 it is possible to see the final decision tree. For these reason the SL application in this work was done taking in consideration these results, even if at the end the code was build from scratch.

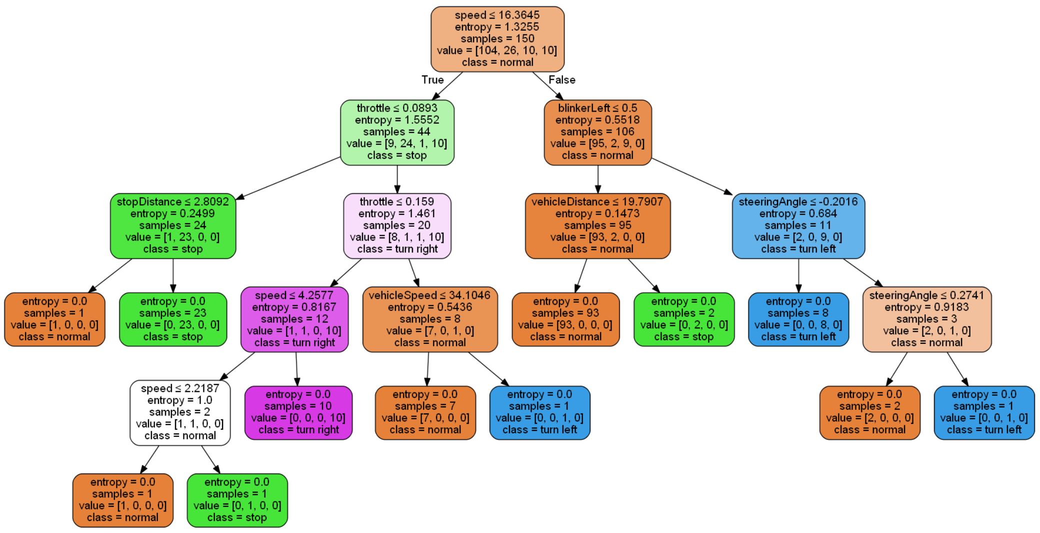


Figure 4.4: Decision Tree for the event recognition

Instead the Reinforcement Learning application showed bad results, with basically a not working neural network. The reinforcement learning application was build from scratch using the ML-Agents plug-in. Figure 4.5 shows the reward function results.

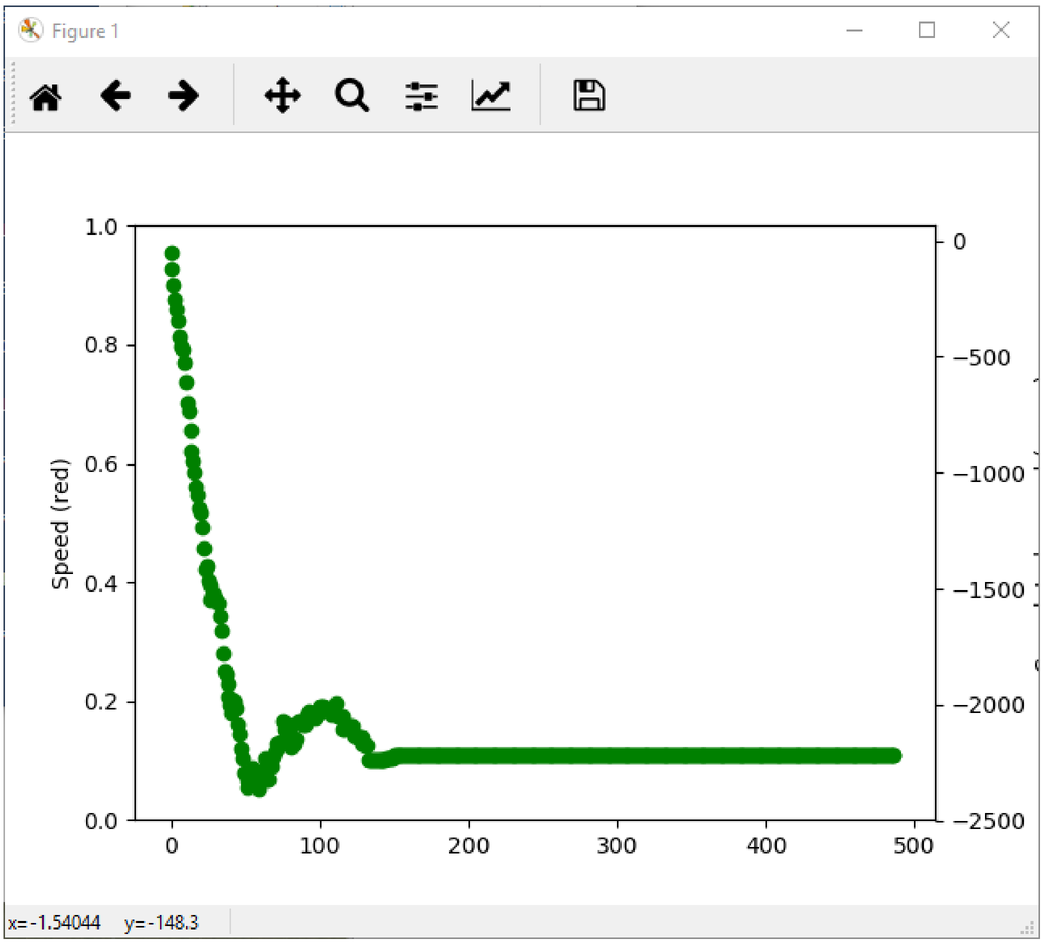


Figure 4.5: Reward

Chapter 5

Application & Results

5.1 Introduction

In this chapter the application of the tools presented in chapter 3 is shown in details, along with the results. It is divided in 3 sections: first section is dedicated to the modification applied to the Unity environment, second one dedicated to the ontologies instantiation and finally the two machine learning applications.

5.2 Unity Environment

This section presents all the addition made in the environment, including static objects, moving objects and the global road system. Once the scenario is completed, the data collection made in Unity is presented, since it represents the start for the ontological creation and for the Machine Learning application.

5.2.1 Road System

These modifications were applied in order to create a more realistic scenario. The creation of a traffic light system, a more complete road and the implementation of many road signs were the choice made to achieve that.

Road

The first action was the amplification of the road. Since the goal was to have a bigger scenario that embedded a traffic light system and moving cars, the road was basically doubled, with the addition of 2 *Horizontal segments* and 3 *Vertical* ones. In order to connect them, a *Road Center* with 4 entries was build, in parallel with two *Road Center* with 3 entries, put in the other two intersections. Figure 5.1 show the final road, highlighted in green/orange.

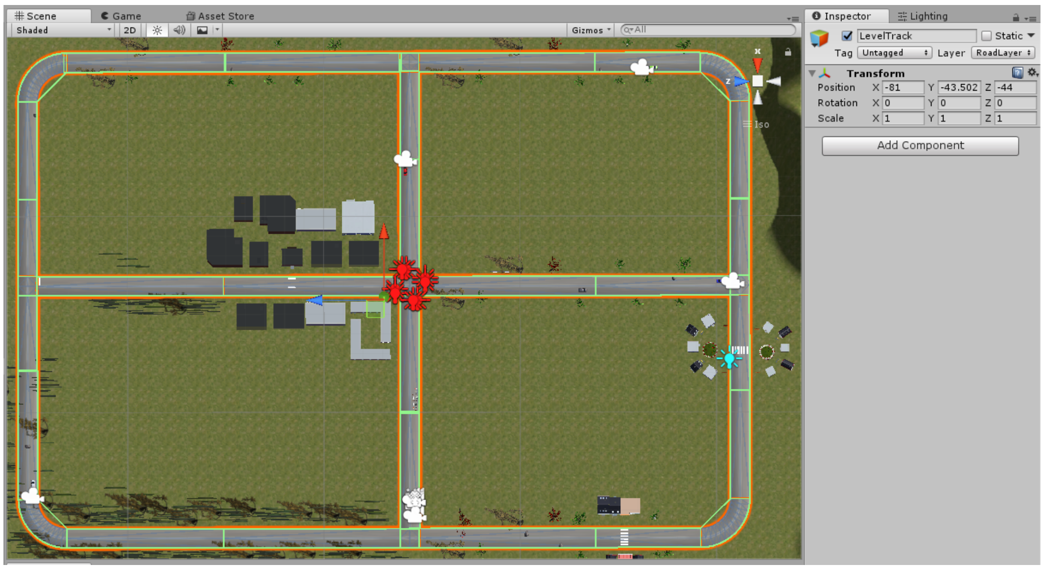
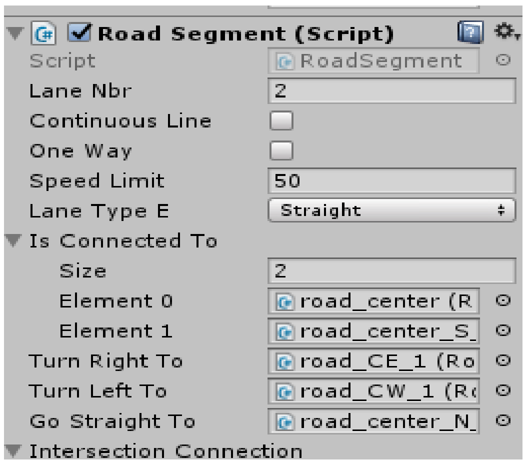


Figure 5.1: The new global road

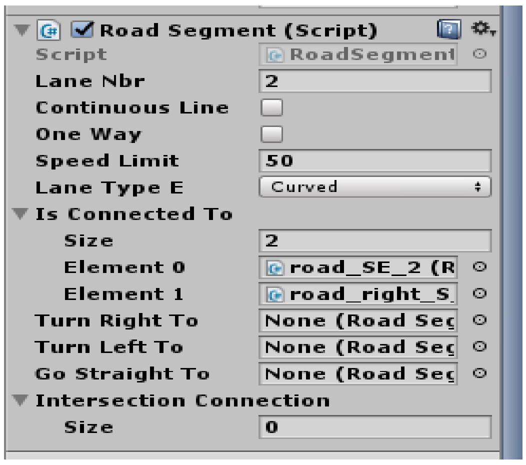
The intelligent vehicle has to build in his KB a map representing the road, so every road segment has a script attached that allows to build the connection to other game objects. The vehicle will be able to build the correspondent map thanks to the *TAG: Road* attached to every segment. Figures 5.4, 5.2 and 5.3 show the three different types of road segments: crossing, straight and curved.



Road Segment (Script)

Script	RoadSegment
Lane Nbr	2
Continuous Line	☐
One Way	☐
Speed Limit	50
Lane Type E	Straight
Is Connected To	
Size	2
Element 0	road_center (R
Element 1	road_center_S_
Turn Right To	road_CE_1 (Ro
Turn Left To	road_CW_1 (R(
Go Straight To	road_center_N_
Intersection Connection	

Figure 5.2: Straight segment

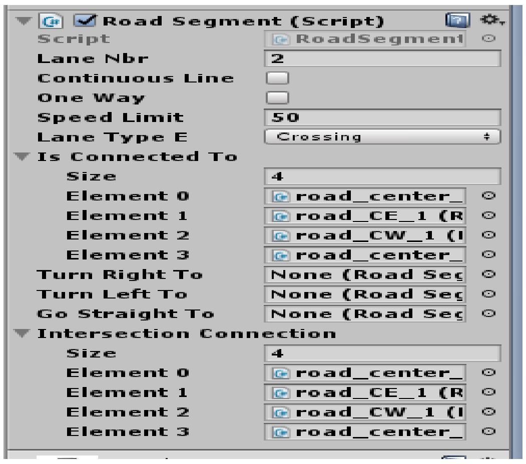


Road Segment (Script)

Script	RoadSegment
Lane Nbr	2
Continuous Line	☐
One Way	☐
Speed Limit	50
Lane Type E	Curved
Is Connected To	
Size	2
Element 0	road_SE_2 (R
Element 1	road_right_S_
Turn Right To	None (Road Seg
Turn Left To	None (Road Seg
Go Straight To	None (Road Seg
Intersection Connection	
Size	0

Figure 5.3: Curved segment

It is possible to see that the straight segment has as parameters the two elements which they are connected with, plus the possibility to turn in the two directions and to go straight, while the curved one has only the connections.



```
Road Segment (Script)
Script                  RoadSegment
Lane Nbr                2
Continuous Line         ☐
One Way                 ☐
Speed Limit             50
Lane Type E             Crossing
Is Connected To
    Size                4
    Element 0           road_center_
    Element 1           road_CE_1 (R
    Element 2           road_CW_1 (I
    Element 3           road_center_
Turn Right To           None (Road Seg
Turn Left To            None (Road Seg
Go Straight To          None (Road Seg
Intersection Connection
    Size                4
    Element 0           road_center_
    Element 1           road_CE_1 (R
    Element 2           road_CW_1 (I
    Element 3           road_center_
```

Figure 5.4: Crossing segment

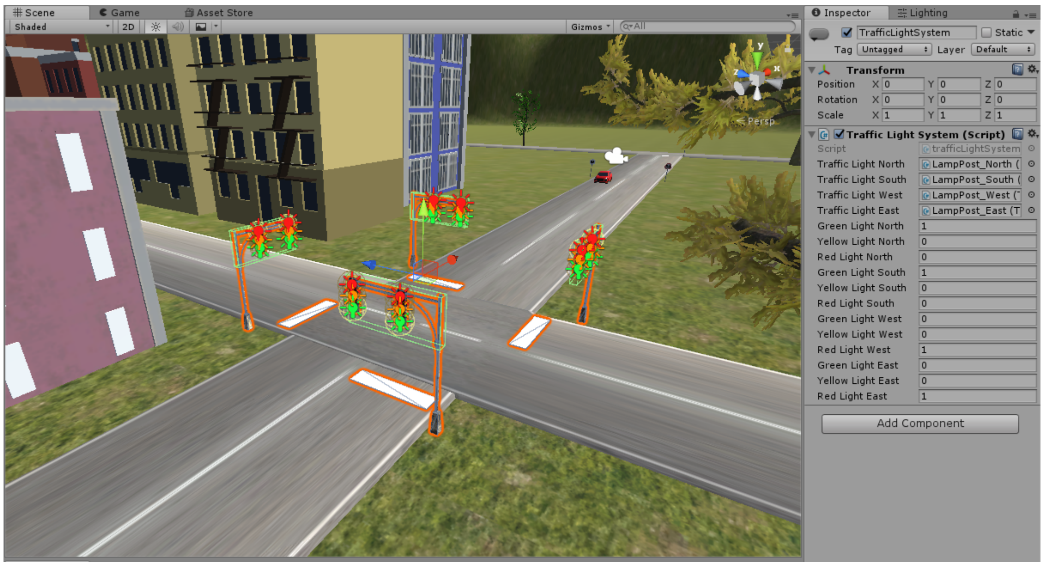
The crossing center has in addition the intersection connection, managing the segments that it has to connect. Each road center has a size, that allows to define the correspondent number of segments connected.

Traffic Light

Second step is the *Traffic Light System* implementation. It implements two different types of traffic light, in order to have the correct behaviour of the lights. Since the traffic light are 4, they were named with respect to the cardinal signs; the *North and South* have the *Traffic_Light_A* script attached while the *West and East* have the *Traffic_Light_B*. The light behaviour is resumed in Table 5.1, while Figure 5.5 shows a frame captured from a running scene.

The binary value for the light shows when the lights are active (1) or disabled (0). It is important to highlight the period for the green, yellow and red states:

- Green: on for 11 seconds and then off for 11 seconds.



TrafficLightSystem ☐ Static
Tag: Untagged Layer: Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

Traffic Light System (Script)

Script	trafficLightSystem
Traffic Light North	LampPost_North (
Traffic Light South	LampPost_South (
Traffic Light West	LampPost_West (T
Traffic Light East	LampPost_East (T
Green Light North	1
Yellow Light North	0
Red Light North	0
Green Light South	1
Yellow Light South	0
Red Light South	0
Green Light West	0
Yellow Light West	0
Red Light West	1
Green Light East	0
Yellow Light East	0
Red Light East	1

Figure 5.5: Traffic Light System

Traffic Light	Light	7s	4s	7s	4s
Traffic Light A	Green	1	1	0	0
	Yellow	0	1	0	0
	Red	0	0	1	1
Traffic Light B	Green	0	0	1	1
	Yellow	0	0	0	1
	Red	1	1	0	0

Table 5.1: One Traffic Light System period (22 seconds)

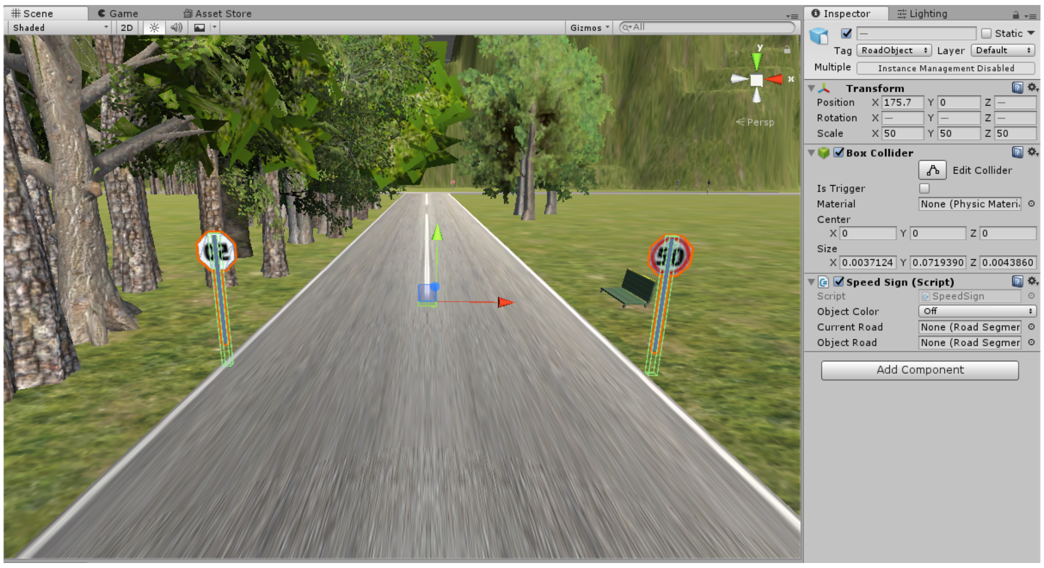
- Yellow: on for 4 seconds and then off for 18 seconds.

- Red: on for 11 seconds and then off for 11 seconds.

The traffic light system simply manages the rotation of this life-cycles.

Signs & Road Object Class

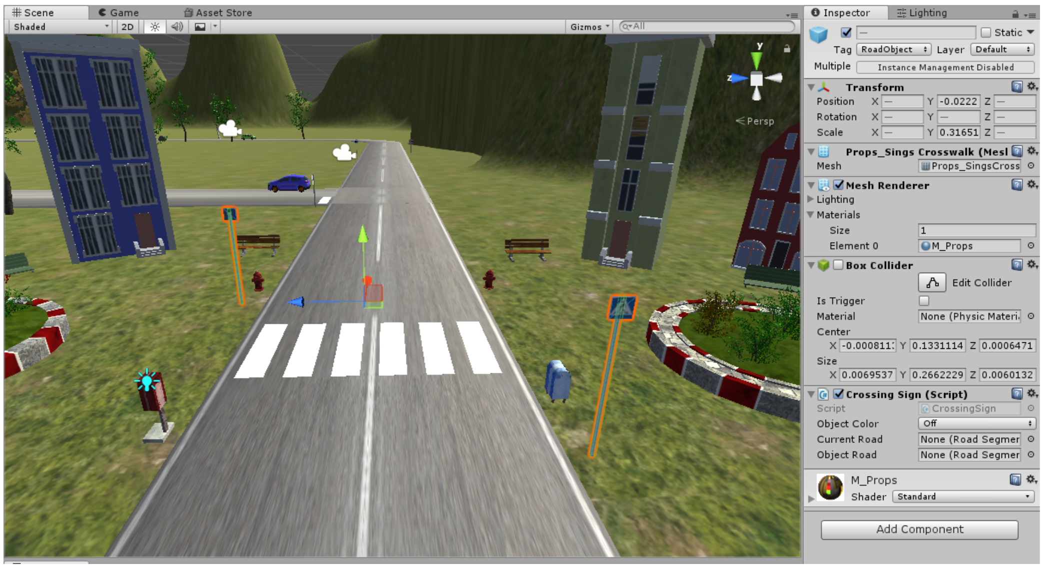
In this work 4 types of traffic sings were implemented: *Speed, Stop, Work In Progress and Pedestrian crossing.* Figure 5.6, 5.7, 5.8 and 5.9 show these different traffic signs. Each of these sign types has a script attached that allows them to behave as a *RoadObject* and a Tag "*RoadObject*". Being a *RoadObject* means that they will inherit all the functions present in this class. The Tag instead allows the intelligent vehicle to collect them knowing that they are *RoadObject* game objects.



```
Static
Tag RoadObject          Layer Default
Multiple  Instance Management Disabled
Transform
Position   X 175.7      Y 0          Z —
Rotation   X —          Y —          Z —
Scale      X 50         Y 50         Z 50
Box Collider
Is Trigger
Material       None (Physic Materi
Center
  X 0          Y 0          Z 0
Size
  X 0.0037124  Y 0.0719390  Z 0.0043860
Speed Sign (Script)
Script         SpeedSign
Object Color   Off
Current Road   None (Road Segmer
Object Road    None (Road Segmer
```

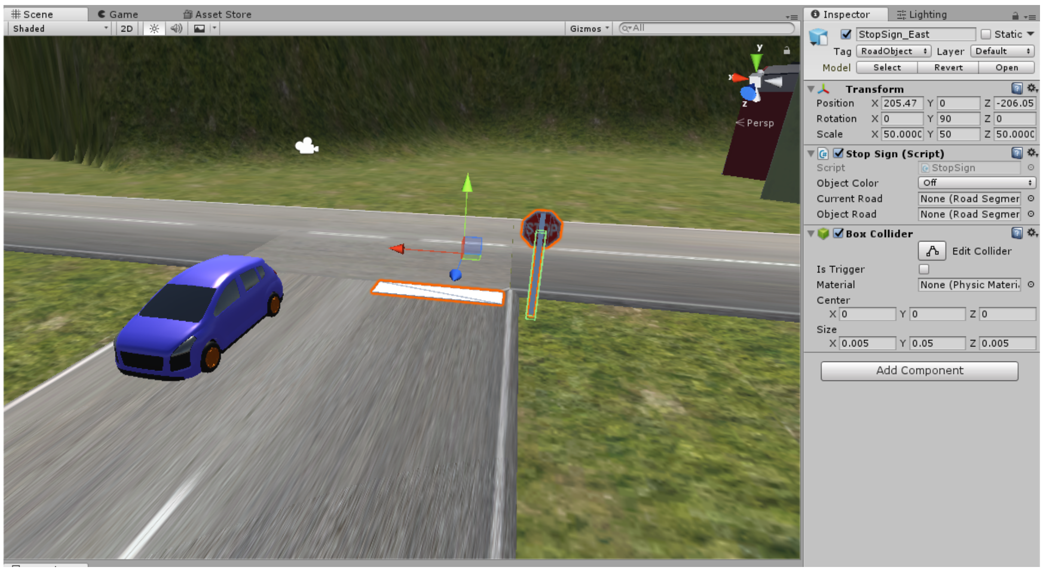
Figure 5.6: Speed signs

The RoadObject class has some important functions implemented in it:



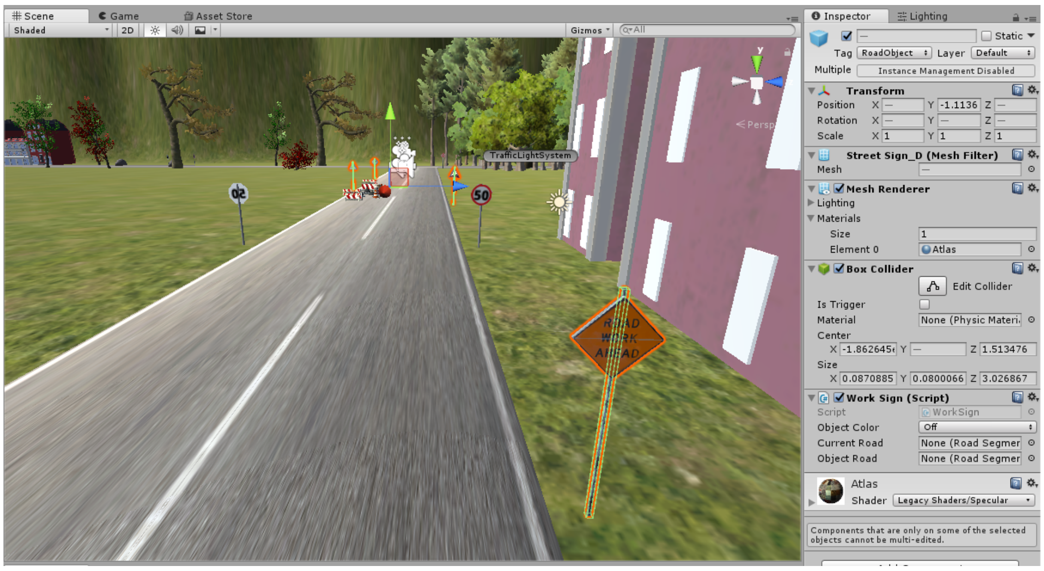
```
Tag RoadObject   Layer Default
Multiple  Instance Management Disabled
Transform
Position  X —        Y -0.0222   Z —
Rotation  X —        Y —         Z —
Scale     X —        Y 0.31651   Z —
Props_Sings Crosswalk (Mesl
Mesh                 Props_SingsCross
Mesh Renderer
Lighting
Materials
  Size               1
  Element 0          M_Props
Box Collider
                     Edit Collider
Is Trigger
Material             None (Physic Materi
Center
  X -0.000811:  Y 0.1331114  Z 0.0006471
Size
  X 0.0069537   Y 0.2662229  Z 0.0060132
Crossing Sign (Script)
Script               CrossingSign
Object Color         Off
Current Road         None (Road Segmer
Object Road          None (Road Segmer
M_Props
Shader Standard
```

Figure 5.7: Crossing signs



```
StopSign_East                        Static
Tag RoadObject   Layer Default
Model  Select  Revert  Open
Transform
Position  X 205.47    Y 0    Z -206.05
Rotation  X 0         Y 90   Z 0
Scale     X 50.0000   Y 50   Z 50.0000
Stop Sign (Script)
Script               StopSign
Object Color         Off
Current Road         None (Road Segmer
Object Road          None (Road Segmer
Box Collider
                     Edit Collider
Is Trigger
Material             None (Physic Materi
Center
  X 0       Y 0      Z 0
Size
  X 0.005   Y 0.05   Z 0.005
```

Figure 5.8: Stop signs



```
Static
Tag  RoadObject   Layer  Default
Multiple  Instance Management Disabled
Transform
Position   X —   Y -1.1136   Z —
Rotation   X —   Y —         Z —
Scale      X 1   Y 1         Z 1
Street Sign_D (Mesh Filter)
Mesh   —
Mesh Renderer
Lighting
Materials
  Size        1
  Element 0   Atlas
Box Collider
  Edit Collider
Is Trigger
Material   None (Physic Materi
Center
  X -1.862645   Y —           Z 1.513476
Size
  X 0.0870885   Y 0.0800066   Z 3.026867
Work Sign (Script)
Script         WorkSign
Object Color   Off
Current Road   None (Road Segmer
Object Road    None (Road Segmer
Atlas
Shader   Legacy Shaders/Specular
Components that are only on some of the selected
objects cannot be multi-edited.
```

Figure 5.9: Work signs

- *float getSize(), float getPosition(), float getSpeed()*: classic functions that return the correspondent value for the size, the position in the scenario and the current object speed.

- *void getLane(GameObject vehicle, RoadSegment objectRoad, string orientation)* : it is a void function that updates the value of the global variable *isOnLineRealtive*. This variable has 3 possible values:

  – 1: if the vehicle and the object are on the same road and lane.

  – -1: if the vehicle and the object are on the same road, but on different lane.

  – 0: if they are on different road or the game object is not on a road segment.

- *GameObject isHitting()* : return the GameObject that the object is hitting. For example the road signs are hitting the terrain since they are on it, while the moving cars or the work on the road are hitting different road segments.

- *string toJson(GameObject vehicle)*: this function returns a string containing all the informations standardized in the format required for the application. More informations about these formats can be found in appendix A and B.

5.2.2 Moving Objects

In this project there are two different moving objects: *Moving cars and Pedestrians*. They belong to the same *RoadObject* class, but as for the *Traffic Light System*, a specific behaviour is implemented.

Moving Cars

The moving cars implement a pre-defined *AI*: this means that the car will accelerate and steer by itself, avoiding obstacles, but following a predefined path [40]. The pre-defined *AI* is mainly composed of 3 actions that can be performed:

- *Drive*: this function let the car accelerate till the speed limit imposed by the current road, steer in order to follow the path and break when there is an over speeding situation or an obstacle has to be avoided.

- *Avoid the obstacle*: the obstacle avoidance is possible thanks to the implementation of in game created sensors, positioned in the front part of the car. These sensors detect if an object is in front of the car, calling the *steer* function when needed. After an avoidance, the car will return to follow the path, returning to the correct lane.

- *Manage Traffic Light*: the 3 possible states of a traffic light has to be managed by a moving car that is approaching the road center. If the situations related to the state green and red are simple to manage, in the case of the yellow light the situation is more complicated. Basically there two cases:

  - The yellow light becomes on when the car is going to approach the center.
  - The yellow light become on when the car is crossing the center.

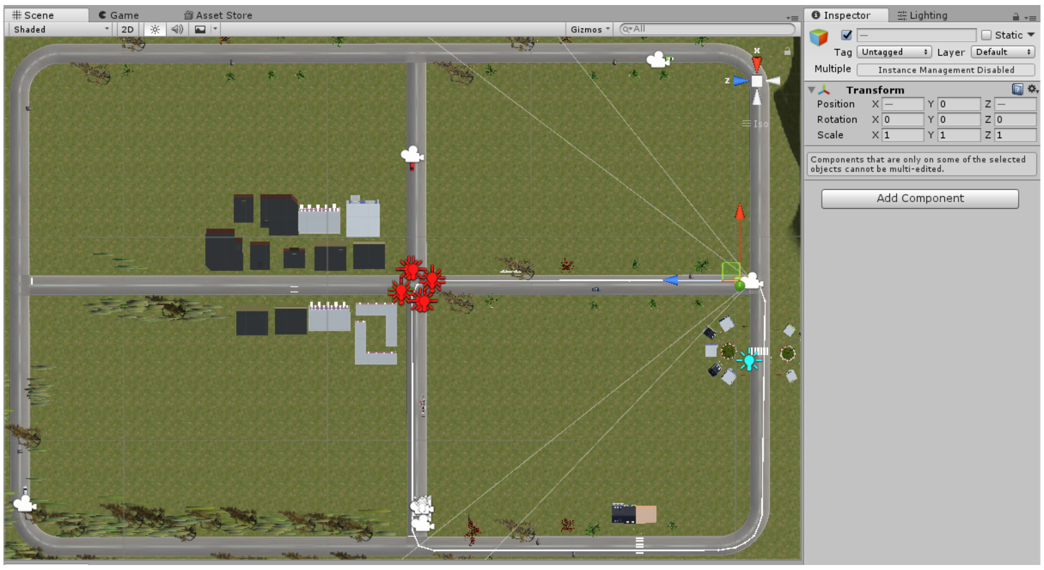
  The algorithm for the *void ManageTrafficLight()* function is presented in Algorithm 1. The variables *passing and passed* are boolean and both initialized as false.

  Basically the car will consider the light states only if the distance is less than 7 meters and the vehicle is not passing trough the center of the road. In this case, the car has to continue to follow the path, in order to avoid to stop in the middle of the road. The second part of the function is necessary in order to avoid the influence of the new closest traffic light, that has not to be taken into account until the next road center approach.

Figure 5.10, 5.11, 5.12 and 5.13 show the 4 different path followed by the moving cars. It is important to notice that every moving car will have to manage the traffic light system.

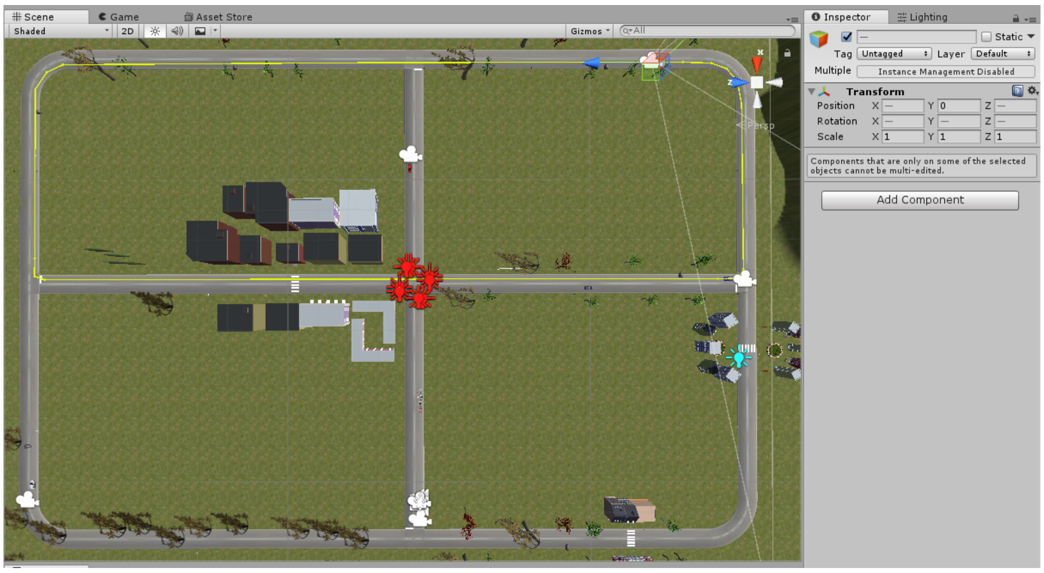
```
Data: The four Traffic lights (TL)
Result: Actions to take
for each Traffic Light (TL) do
    if TL is the closest OR car is passing the TL OR (passed the TL
     AND has not passed the other TLs) then
        Look at the lights;
        if distance > 15 meters AND not passed then
            update distance and minimum distance;
        else
            if minium Distance > distance from TL then
                if RED on OR (YELLOW on AND distance < 7
                 meters) then
                    break
                else
                    continue without breaking
                end
            end
            Passing = true;
            update distance and minimum distance;
        end
        else if TL is the closest OR car is passing the TL OR (passed
         the TL AND has not passed the other TLs) then
            Look at the lights;
            if distance from TL < 60 meters then
                Passed = true;
                update distance;
            else
                Passed = false;
                Update distance and minimum distance;
            end
        end
    end
end
```

Algorithm 1: The *ManageTrafficLight()* function



Tag Untagged Layer Default
Multiple Instance Management Disabled
Transform
Position X — Y 0 Z —
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Components that are only on some of the selected objects cannot be multi-edited.

Figure 5.10: Car SE Path



Tag Untagged Layer Default
Multiple Instance Management Disabled
Transform
Position X — Y 0 Z —
Rotation X — Y — Z —
Scale X 1 Y 1 Z 1
Components that are only on some of the selected objects cannot be multi-edited.

Figure 5.11: Car NE Path

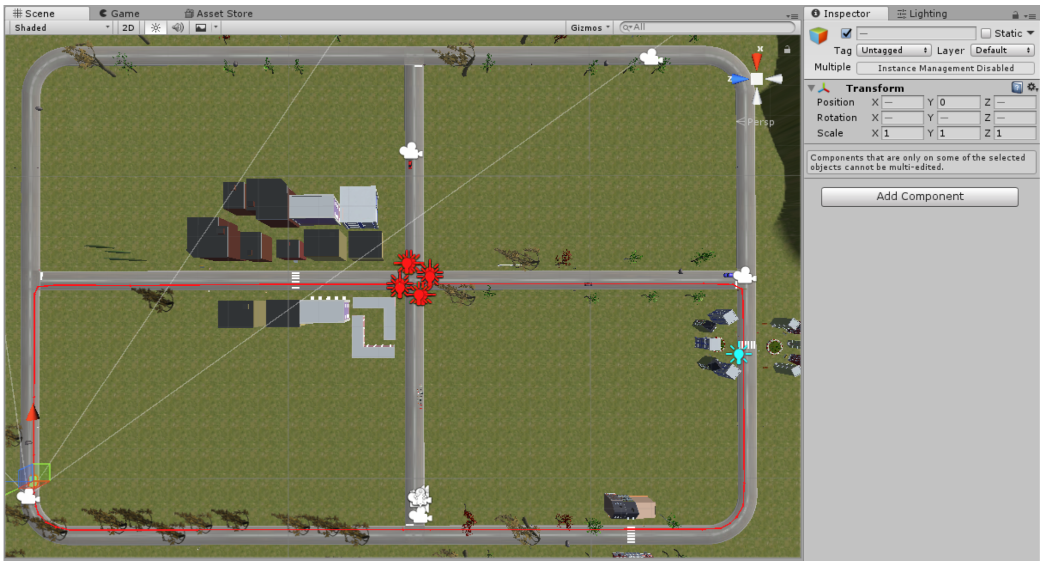


Figure 5.12: Car SO Path

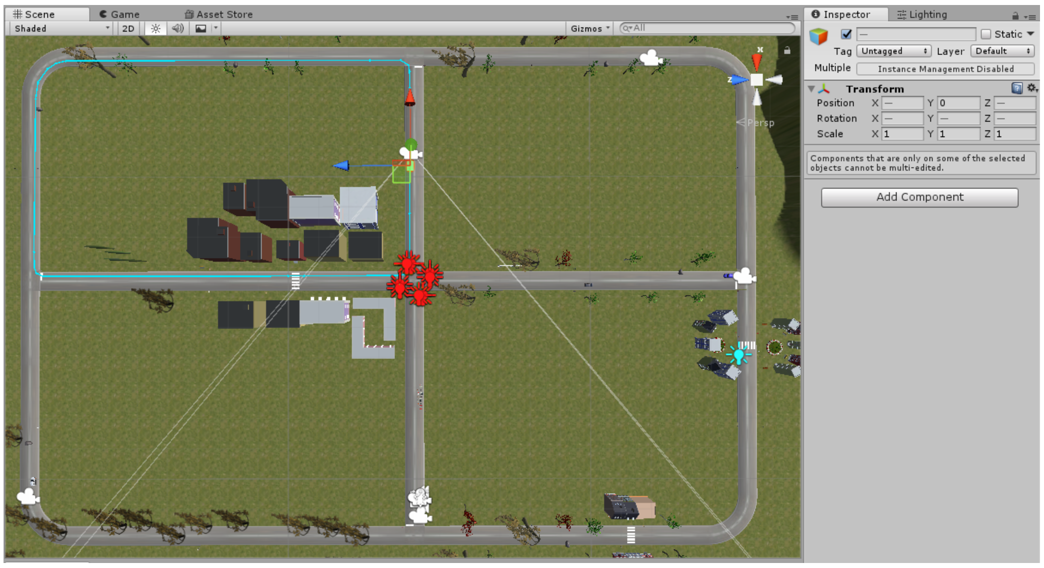
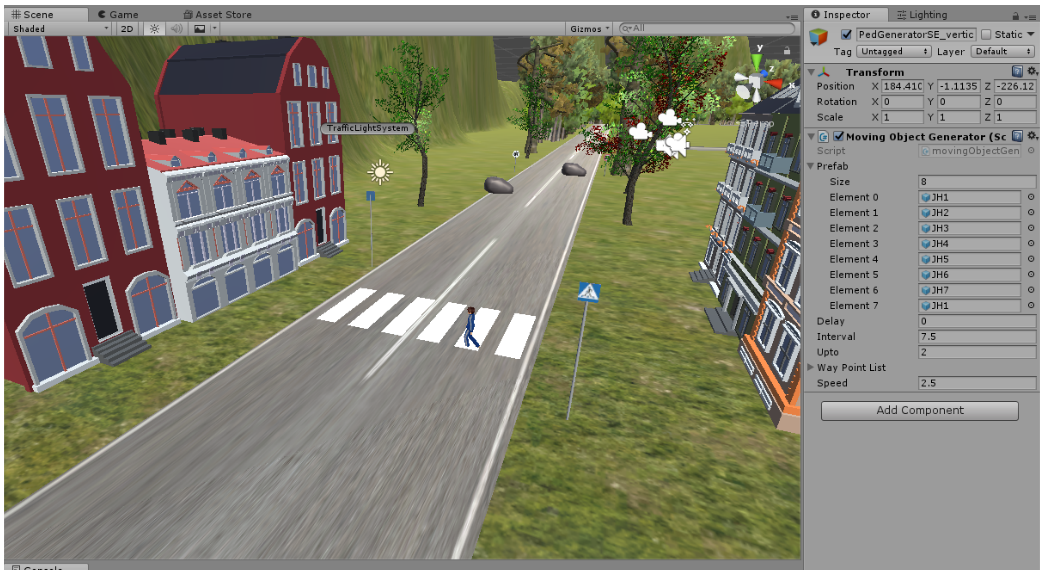


Figure 5.13: Car NO Path

Pedestrians

The pedestrian generation, movements and destruction are managed by the *Pedestrian_Generator* object. This script allows to create the *Pedestrian* game object and it has 4 important parameters that have to be set, as shown in Figure 5.14:



PedGeneratorSE_vertic
Static
Tag Untagged
Layer Default

Transform

Position	X	184.410	Y	-1.1135	Z	-226.12
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Moving Object Generator (Sc

Script	movingObjectGen
Prefab	
Size	8
Element 0	JH1
Element 1	JH2
Element 2	JH3
Element 3	JH4
Element 4	JH5
Element 5	JH6
Element 6	JH7
Element 7	JH1
Delay	0
Interval	7.5
Upto	2
Way Point List	
Speed	2.5

Figure 5.14: Pedestrian and generation parameters

- Delay: determine the delay in seconds for the activation of the generator.
- Size: fixes the number of pedestrians that will be generated.
- Interval: The time interval in seconds between each generation.
- Up to: How many pedestrians can exist at the same time.

The moving objects are considered as normal *RoadObject* by the vehicle, whose management would be explained in the next sections.

5.2.3 Static Objects

The *Static objects* are the second type of *Obstacles* that the vehicle can find on the road. As for the *Moving Objects*, they belong to the *RoadObject* class, inheriting all the functions described in the previous section. There are 4 different types of *Static objects* in this work:

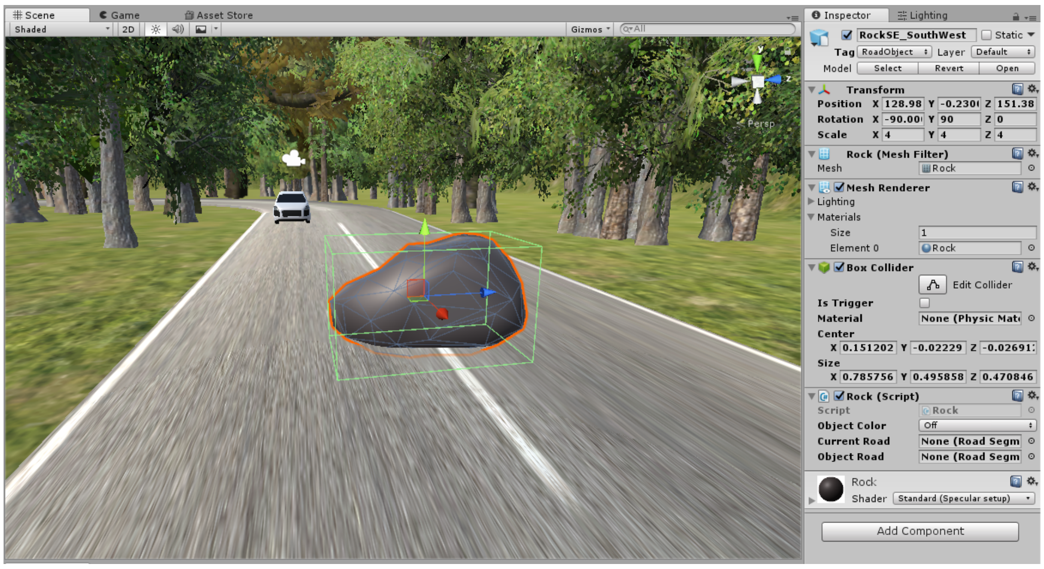
1. Rocks: 7 rocks are present in the scenario. They affect the moving cars and they are the most common and easiest obstacle to manage present in the scenario. They can be put in the middle of the lane, in the middle of the road or at the border of a lane.

2. Static Car: It is a game object identical to the other 4 moving cars, but it does not have attached the script *CarEngine* that allows the movements. It can be considered as a Rock with the shape of the moving car.

3. Tree on the road: As for the static car, only one object belonging to this class is present in the scenario. It simulates a tree that felt on the road with the top part. It is affecting only one lane, without taking the entire road.

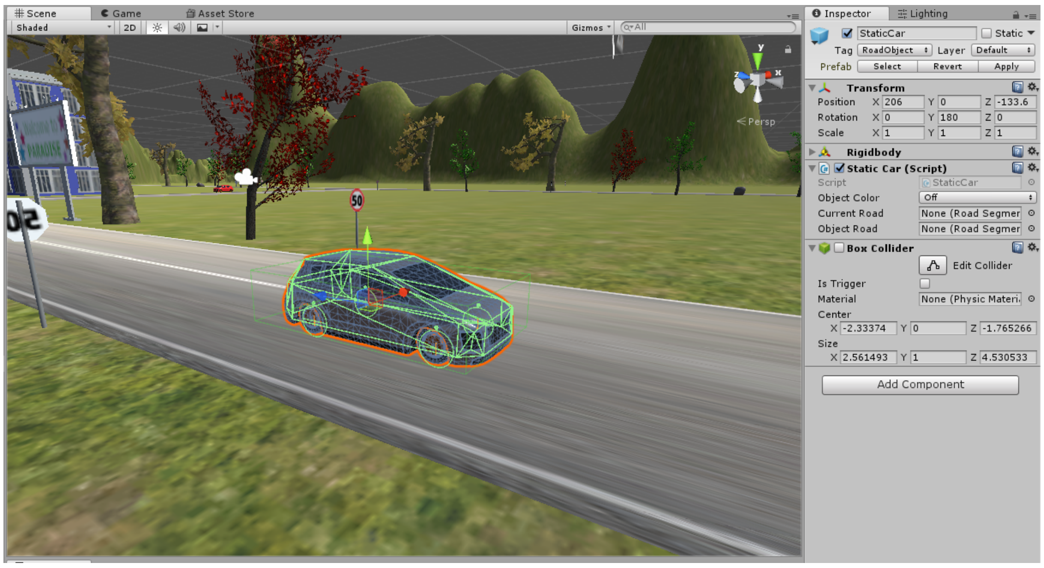
4. Work On the Road : It is a game object composed of several common work objects, such work cones and roadblocks. A single game collider is attached to the game object, allowing it to be identifiable as a single obstacle.

Figures 5.15, 5.16, 5.18, 5.17 show the objects with the attached script, while in the general scheme, shown in Figure 5.19, it is possible to see the actual position of all the objects present on the scene.



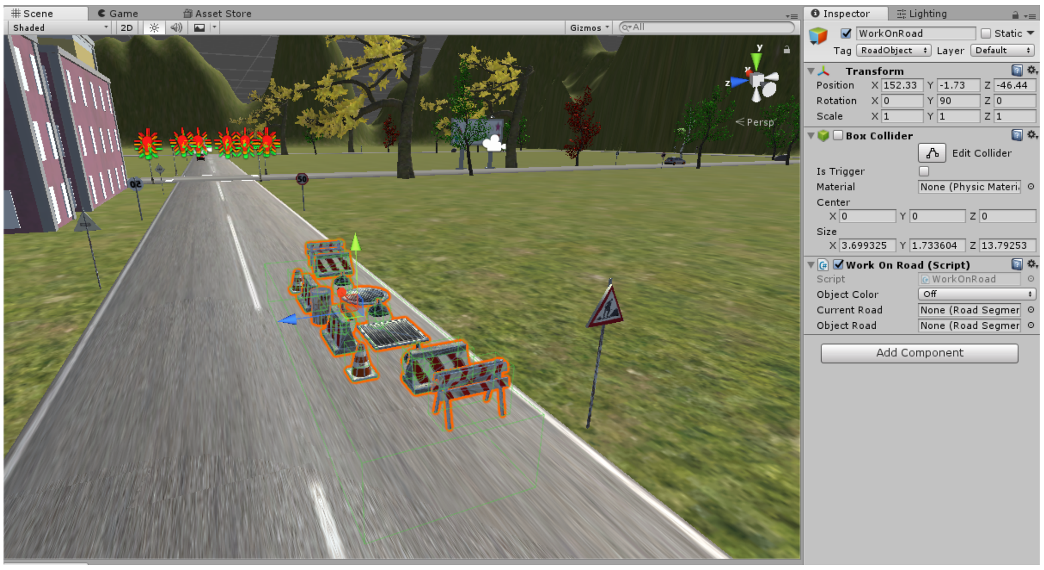
```
RockSE_SouthWest                         Static
Tag  RoadObject   Layer  Default
Model  Select  Revert  Open
Transform
Position   X 128.98   Y -0.230(   Z 151.38
Rotation   X -90.00(  Y 90        Z 0
Scale      X 4        Y 4         Z 4
Rock (Mesh Filter)
Mesh                 Rock
Mesh Renderer
Lighting
Materials
  Size               1
  Element 0          Rock
Box Collider
                     Edit Collider
Is Trigger
Material             None (Physic Mat(
Center
  X 0.151202  Y -0.02229  Z -0.02691:
Size
  X 0.785756  Y 0.495858  Z 0.470846
Rock (Script)
Script               Rock
Object Color         Off
Current Road         None (Road Segm
Object Road          None (Road Segm
Rock
Shader  Standard (Specular setup)
```

Figure 5.15: Generic Rock



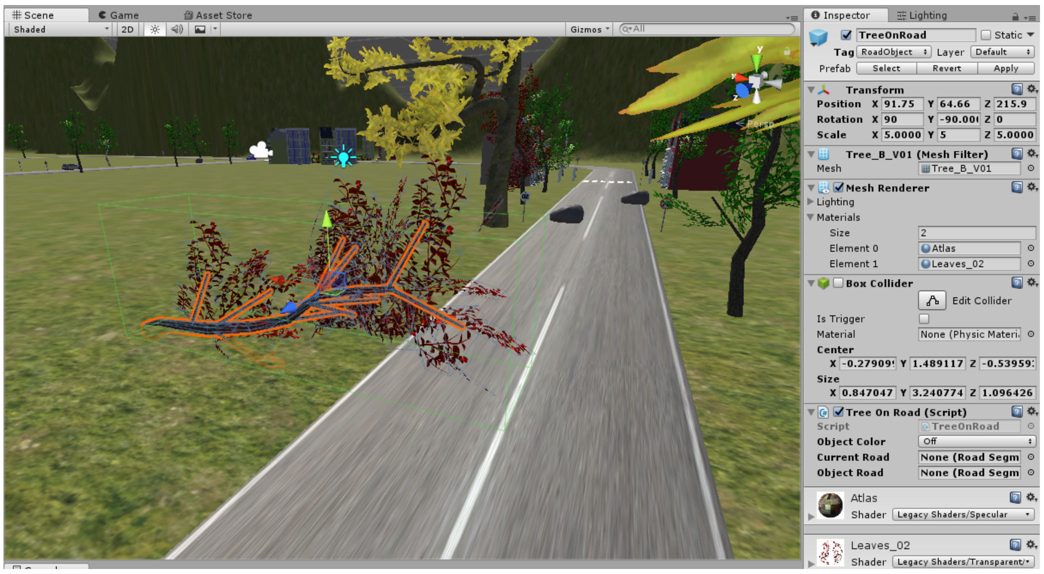
```
StaticCar                                Static
Tag  RoadObject   Layer  Default
Prefab  Select  Revert  Apply
Transform
Position   X 206   Y 0     Z -133.6
Rotation   X 0     Y 180   Z 0
Scale      X 1     Y 1     Z 1
Rigidbody
Static Car (Script)
Script               StaticCar
Object Color         Off
Current Road         None (Road Segmer
Object Road          None (Road Segmer
Box Collider
                     Edit Collider
Is Trigger
Material             None (Physic Materi.
Center
  X -2.33374  Y 0  Z -1.765266
Size
  X 2.561493  Y 1  Z 4.530533
```

Figure 5.16: Static Car



```
Inspector    Lighting
WorkOnRoad                    Static
Tag  RoadObject   Layer  Default
Transform
Position   X 152.33   Y -1.73   Z -46.44
Rotation   X 0        Y 90      Z 0
Scale      X 1        Y 1       Z 1
Box Collider
                 Edit Collider
Is Trigger
Material         None (Physic Materi.
Center
   X 0         Y 0         Z 0
Size
   X 3.699325  Y 1.733604  Z 13.79253
Work On Road (Script)
Script           WorkOnRoad
Object Color     Off
Current Road     None (Road Segmer
Object Road      None (Road Segmer
Add Component
```

Figure 5.17: Work On The Road



```
Inspector    Lighting
TreeOnRoad                    Static
Tag  RoadObject   Layer  Default
Prefab   Select   Revert   Apply
Transform
Position   X 91.75    Y 64.66    Z 215.9
Rotation   X 90       Y -90.00(  Z 0
Scale      X 5.0000   Y 5        Z 5.0000
Tree_B_V01 (Mesh Filter)
Mesh             Tree_B_V01
Mesh Renderer
Lighting
Materials
   Size          2
   Element 0     Atlas
   Element 1     Leaves_02
Box Collider
                 Edit Collider
Is Trigger
Material         None (Physic Materi.
Center
   X -0.27909!  Y 1.489117  Z -0.53959:
Size
   X 0.847047   Y 3.240774  Z 1.096426
Tree On Road (Script)
Script           TreeOnRoad
Object Color     Off
Current Road     None (Road Segm
Object Road      None (Road Segm
Atlas
Shader  Legacy Shaders/Specular
Leaves_02
Shader  Legacy Shaders/Transparent/
```

Figure 5.18: Tree On The Road

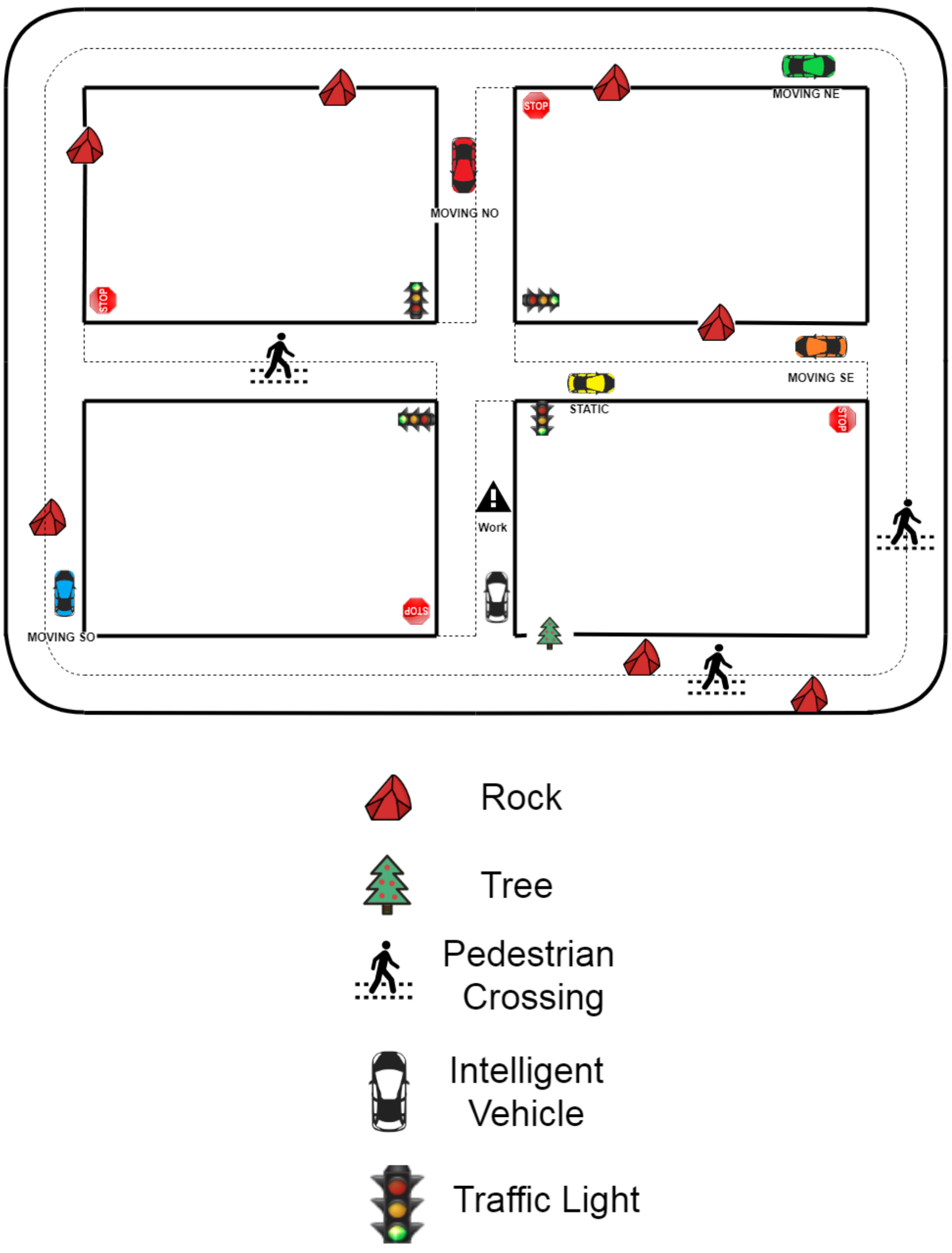


Figure 5.19: General Road Scheme

5.3 General Ontology

The general Ontology represents all the classes that could be present in the scenario. An individual would always be a member of one of those classes. Starting from this general ontology, the instantiated ones will be then created. Figure 5.20 shows the general object ontology.

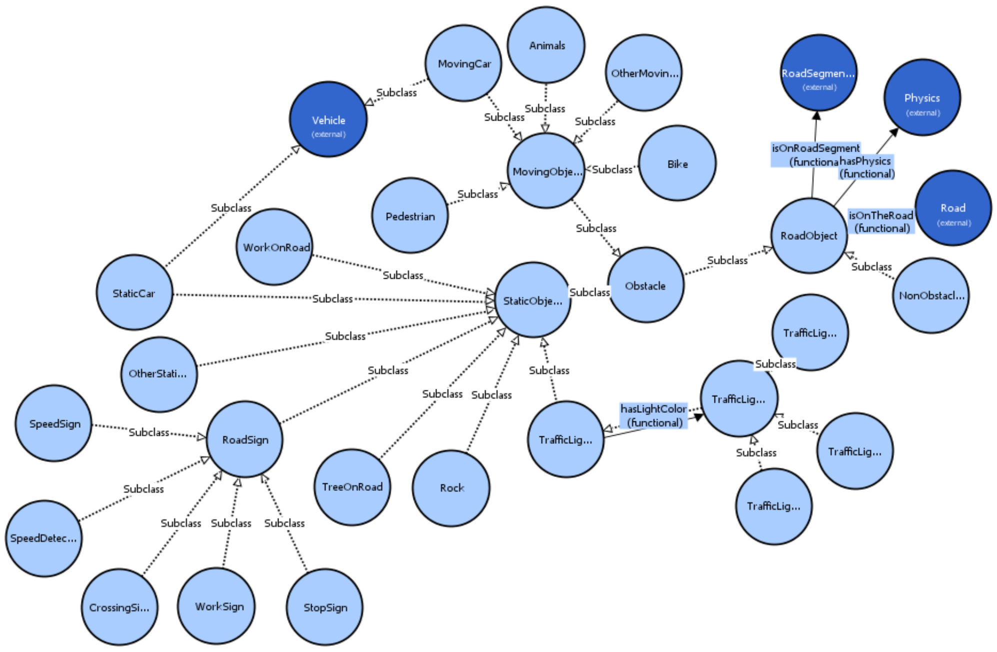


Figure 5.20: General Objects Ontology

It is important to notice that there are some classes that in this project are not present, like *Bike and animal.* Next step in this project could be another expansion of this general ontology. Another important point about the general ontology is that all the instantiated ones will be a smaller version of this one, because the classes with no individuals present in the scene will not exist. The general ontology was actually almost complete and only few changes were done. The most important one was a general subclasses reorganization, together with the addition of new classes, i.e. *Work On The Road, Tree On The Road, etc.*

5.4 Instantiated Ontology

This section presents the 3 instantiated ontologies created thanks to the JAVA application. These 3 ontologies represent the stages required for the obstacle collection, showing how ontology is a fast and simple way for representing the scene frame. Each ontology was created thanks to a corresponding *List* in Unity, collecting different data. These *Lists* were made following 3 simple algorithms, presented in the following subsections. This work is presented in *KEOD* [1] article, done at *ECE* with Prof. *Hina* and Dr. *Soukane*. Let R be the set of all the road objects present in the environment E, C the set of the close road objects and O the set of the obstacles. Mathematically:

- $E = \{e_1, e_2, \ldots, e_n\}$
- $R = \{r_1, r_2, \ldots, r_n\}, \quad R \subset E$
- $C = \{c_1, c_2, \ldots, c_n\}, \quad C \subseteq R$
- $O = \{o_1, o_2, \ldots, o_n\}, \quad O \subseteq C$

5.4.1 General Road Object

Every e related to the driving environment has a tag t, the notation used for identification purposes. For all e in E, if an element e has a tag of

RoadObject, then such e (denoted e_i) is a road object r_i. Mathematically: $\forall e \in E, \quad if \quad \exists e_i$ with $t = "RoadObject" \implies r_i = e_i \wedge R \neq \emptyset$

In figure 5.21 it is possible to see the environment E with all the $r \in R$ highlighted.

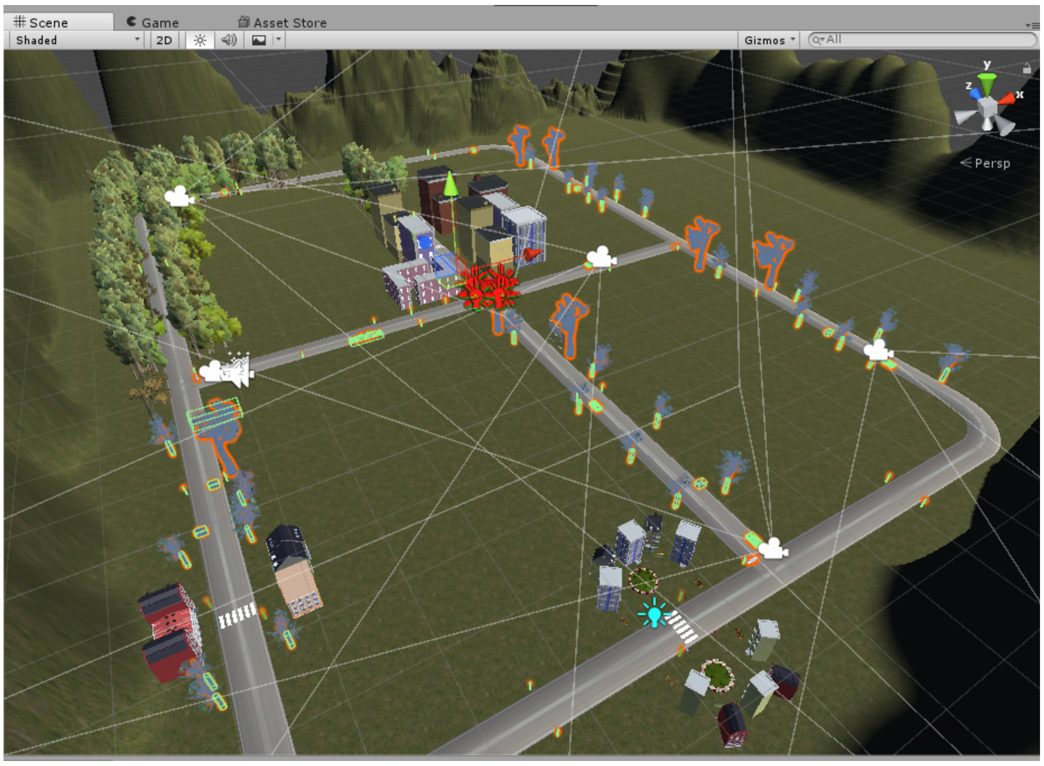


Figure 5.21: Environment E and the entire road objects set R

In figure 5.22, it is possible to see the R ontological representation. The subclasses describe the different r and every sub classes of *Thing* can have one or more individuals. The arrows *hasSubClass* and *hasIndividual* show how this is done in Protégé.

The List is created thanks to the Unity function *FindGameObjectsWithTag()* that allows to find the corresponding tag in the scene (environment E).

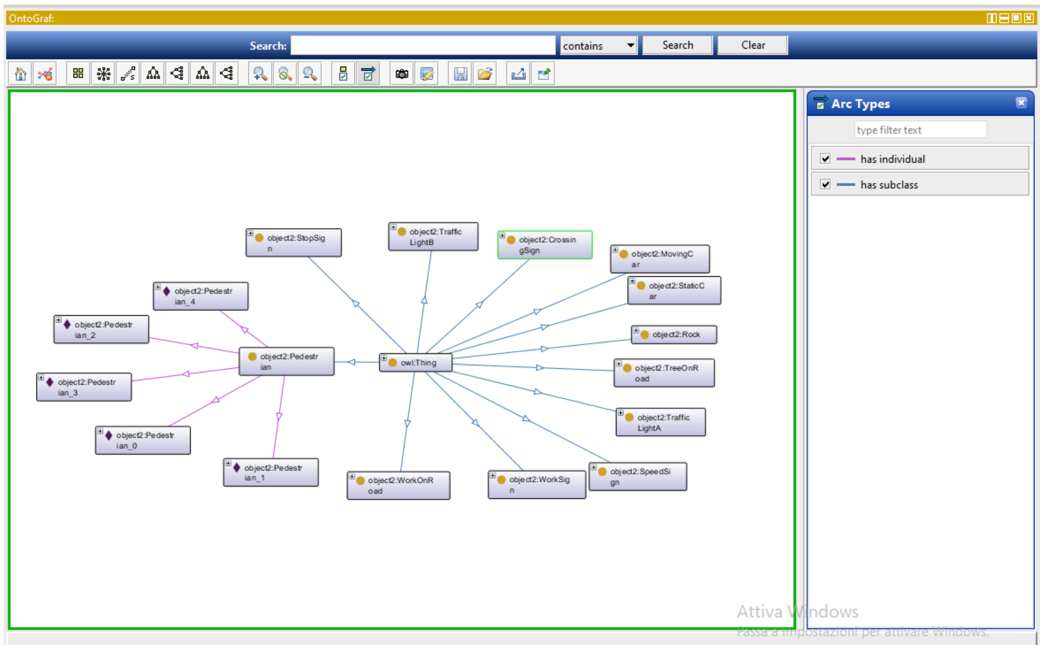


Figure 5.22: The ontological representation of road object collection R

5.4.2 Close Road Object

The nearby road objects are those elements that are within the vicinity of the referenced vehicle. They are parts of the road objects collection. An element has to be considered "near" if it is located within the referenced radius (example: 50 meters) of the referenced vehicle. Let m be the radius of the referenced sphere (as it is the case in Unity 3D), and d the distance between the vehicle and a road object. If the distance d of the road object is less than m then such road object r is considered a nearby road object c. Mathematically, $\forall r \in R, \quad if \quad \exists r_i$ with $d < m \implies c_i = r_i \wedge C \neq \emptyset$.

Figure 5.23 shows the sphere collecting a specific part of E, with all elements $c \in C$ highlighted. It has to be noted that this one is just one of the possible c that can be considered. Nearby road objects may be in front, at the back, on the left or on the right side of the referenced vehicles. Some of the nearby road objects may be obstacles while some may be not. Ontology creation is therefore important because it allows to have a good vision of the closest road objects that may be considered as road obstacles.

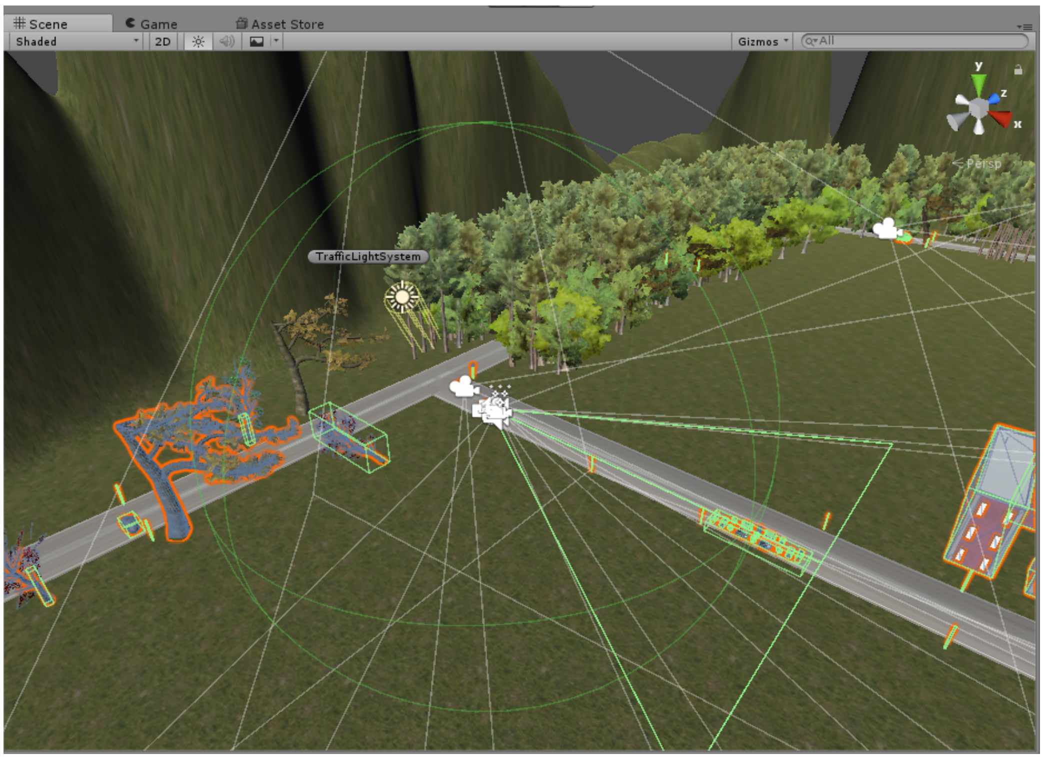


Figure 5.23: Unity 3D Close Road Object scenario

Figure 5.24 shows all elements $c \in C$. As shown, the ontology is much smaller than the previous one, given that we only consider objects that are present in the specified radius of a sphere with the vehicle as the point of reference, with $radius = m$.

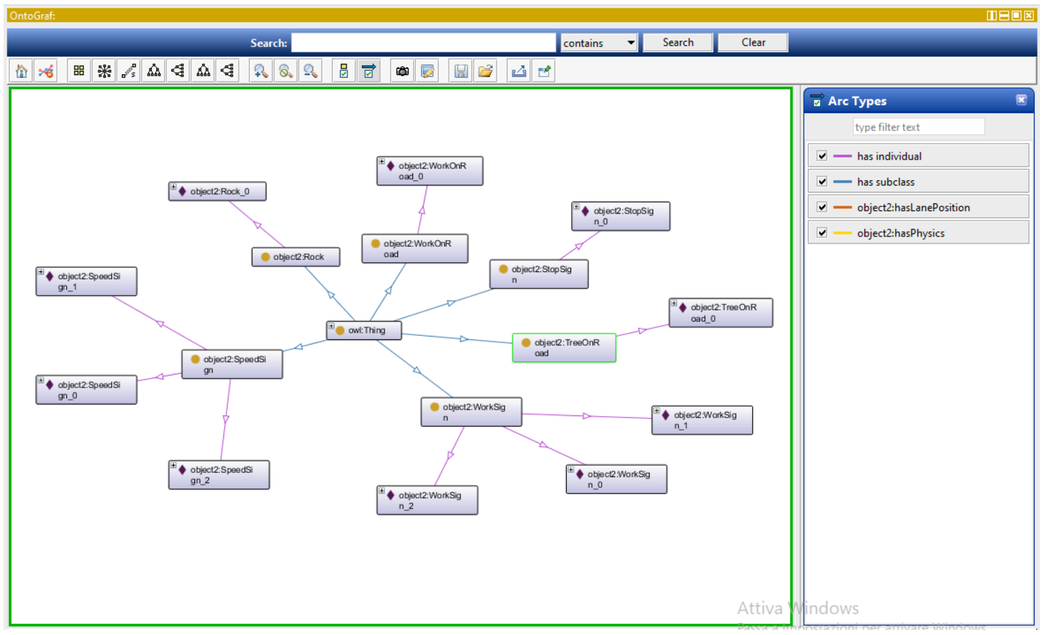


Figure 5.24: Ontological representation of C

5.4.3 Obstacle

The nearby road objects are obstacles if they are in front of the vehicle (located in the same lane), the distance between it and the referenced vehicle is "small", and the time to collision is "near". A formal definition for this scenario is given below.

Let v be the variable describing the current speed of the vehicle, d the one describing the distance between the vehicle and the road object r, t the time to collision limit, m the radius of the sensor sphere, l the one describing if the vehicle and the object are on the same lane and p the cardinal point direction of the vehicle. From a mathematical point of view, O is build as:

$\forall c \in C$, if $\exists c_i$ with $d < m \wedge v/d < t \wedge l = 1 \wedge d > 0 \implies o_i = c_i \wedge O \neq \emptyset$

p is used in the l computation, since the system is dealing with 3D coordinate system, the orientation is necessary in order to define if s and c are on the same lane. Let dcr be the distance from the center of the road.

If $p = North \vee p = South \implies dcr$ is taken on the x plane, else $\implies dcr$ is taken on the z plane. Let dcr_s be the distance system-center of the road, $dcr_c =$ be the distance close object-center of the road:

- $l = 1$, if $dcr_s > 0 \wedge dcr_c > 0 \vee dcr_s < 0 \wedge dcr_c < 0$

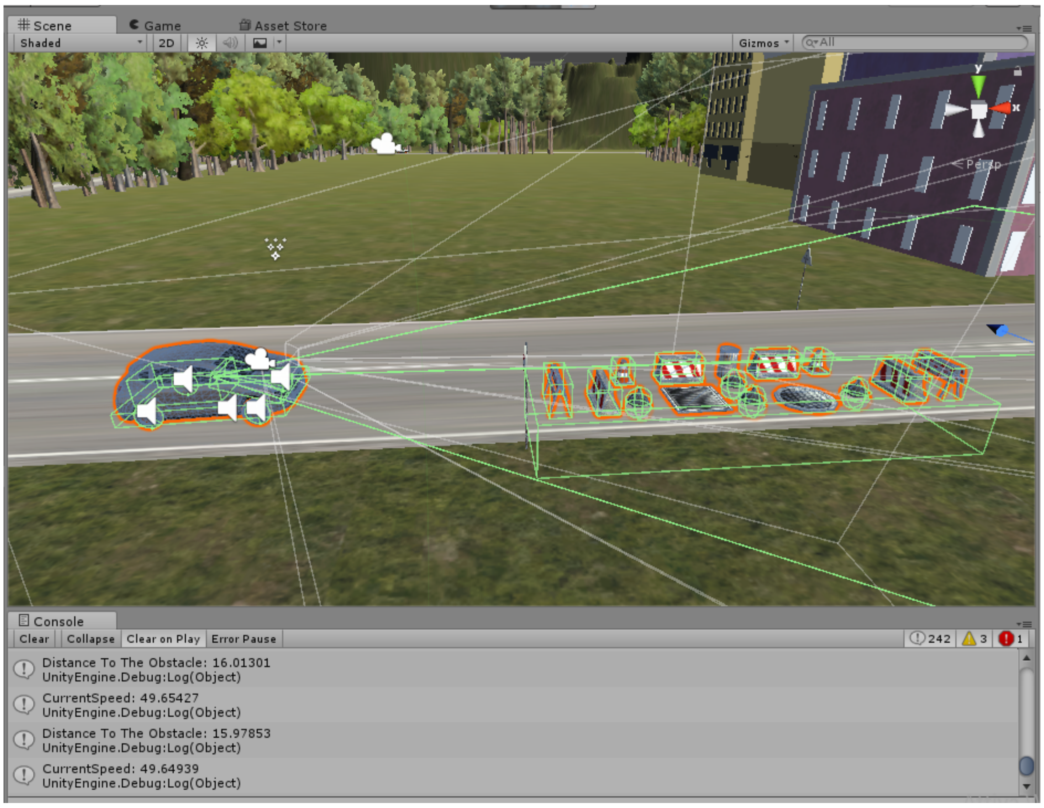
- $l = -1$ if f $dcr_s > 0 \wedge dcr_c < 0 \vee dcr_s > 0 \wedge dcr_c < 0$

In figure 5.25 it is possible to see the o detection scenario. As for the close road object collection phase, it is important to highlight the fact that this is only one of the possible O. Here the ontology representation details are very important, since it is possible to see that all the conditions are verified.

In figure 5.26, it is possible to see every $o \in O$. Here the ontology is presented in details because there is only $o_1 \in O$. The legend on the right allows to show the obstacle characteristics such as speed or size.

In algorithm 2 it is possible to see the 3 situations described before. Targets, maxDistance and timeToCollision are the 3 parameters passed as argument to the function *getObstacles()*. The Targets variable is the *List* storing all the *RoadObject* $r \in R$.

The lane determination is presented in the next section, where the *getLane()* algorithm will be explained since it is a feature used in the Supervised Learning application.



```
Distance To The Obstacle: 16.01301
UnityEngine.Debug:Log(Object)
CurrentSpeed: 49.65427
UnityEngine.Debug:Log(Object)
Distance To The Obstacle: 15.97853
UnityEngine.Debug:Log(Object)
CurrentSpeed: 49.64939
UnityEngine.Debug:Log(Object)
```

Figure 5.25: Unity 3D Obstacle scenario

```
Data: Targets, maxDistance, timeToCollision
Result: Obstacle List
Store initial state: position, acceleration and velocity. for each Target
 do
    Calculate distance;
    if Distance < maxDistance then
        if Speed / acceleration < timeToCollision then
            Determine which object the target is hitting;
            if ObjectTag = Road then
                store the road in the currentRoad variable;
            else
                currentRoad = NULL;
            end
            if currentRoad != NULL then
                Determine the lane;
                if same lane then
                    Target is an obstacle;
                end
            end
        end
    end
end
```

Algorithm 2: The *getObstacles()* function

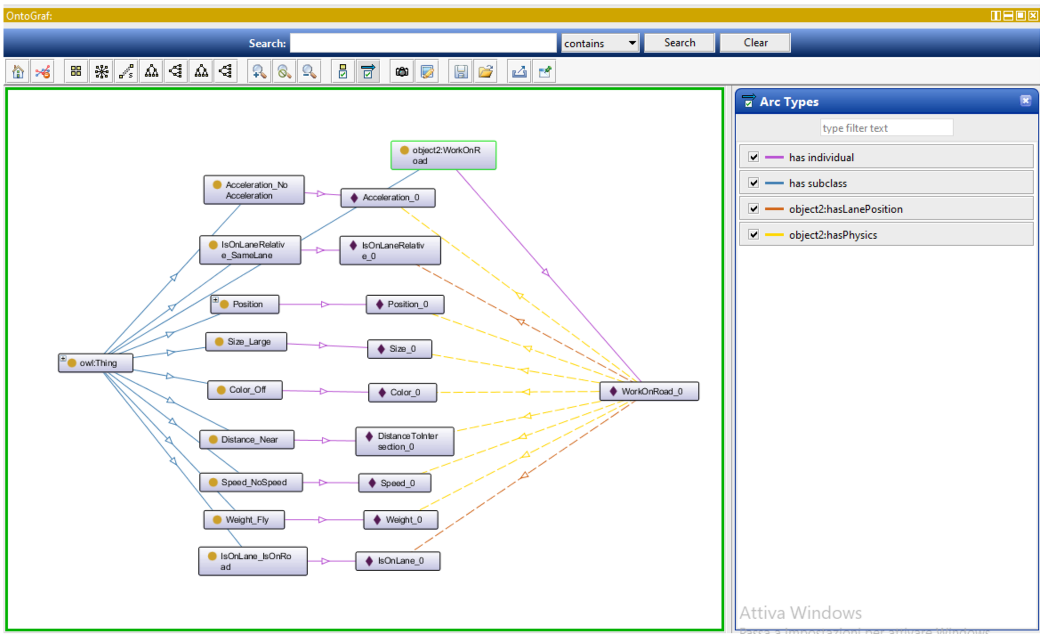


Figure 5.26: Ontological representation of O

5.5 Supervised Learning

This section presents the Supervised Learning application for the *RoadObjects* classification. Appendix B shows in details the data collection phase, highlighting the data format and processing.

There are 12 different objects present in the scene and each of them has 7 features. The different *RoadObjects* are the one described in previous sections, while the features are:

1. *Speed*: calculated directly in Unity, given in real-time.

2. *Acceleration*: calculated looking at the speeds in a Δt.

3. *Distance From vehicle*: calculated looking at the 3 different planes.

4. *Is On The Same Lane*: function shown in algorithm 3, used also in the

obstacle classification.

5. *Size*: each object has a game *Collider* component, whose bounds are used to determine its size.

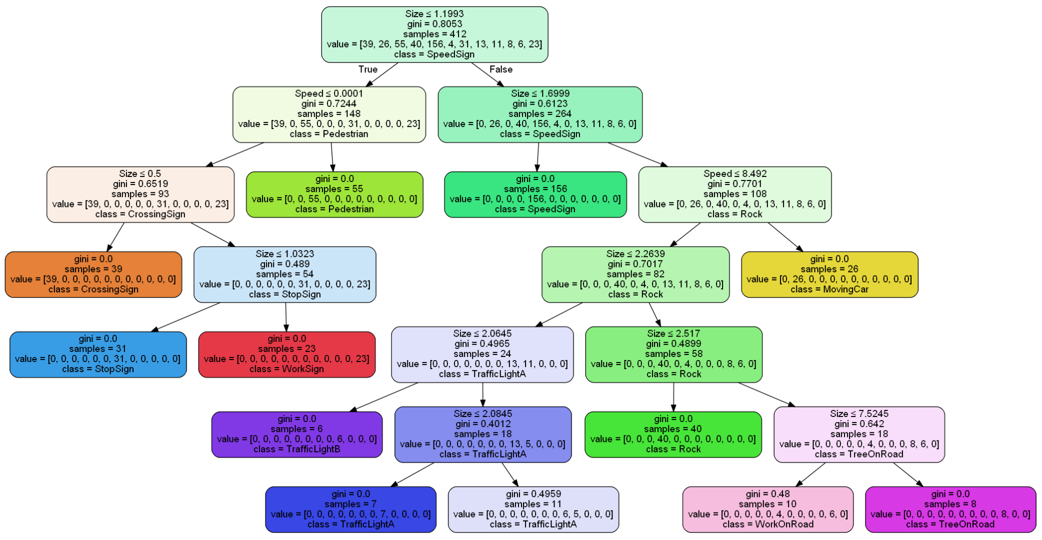
6. *Colour*: this feature is usually set as *"off"*, expect for the traffic light object, whose value changes during time.

```
Data: Vehicle, RoadObject, orientation
Result: Is on Lane
if orientation == NORTH OR SOUTH then
    distance on the x plane;
else
    distance on the z plane;
end
if Vehicle and Object are hitting the same road then
    if (Object distance from road center < 0 AND Vehicle distance
      from road center < 0) OR (Object distance from road center > 0
      AND Vehicle distance from road center > 0) then
        Same Road and Lane;
        Is On Lane = 1;
    else
        Same road, but different Lane;
        Is On lane = -1;
    end
else
    Different Road;
    Is On lane = 0;
end
```

Algorithm 3: The *getLane()* function

5.5.1 Decision Tree

Decision tree learning algorithm uses a decision tree as a predictive model. A decision tree is a flowchart-like structure in which each internal node represents a test on an attribute, each branch representing the outcome of the test while each leaf representing a class label for classification tree. A tree can be created by splitting the training set into subsets based on an attribute value test and repeating the process until each leaf contains a single class label or the desired maximum depth is reached. There are multiple criterion that can be used to divide a node into two branches, such as the information gain, which consist of finding the split that would give the biggest information gain, based on the entropy from the information theory. Figure 5.27 shows the decision tree created for the object classification. *Gini* impurity parameter is a measure of how often a randomly chosen element from the set would be incorrectly labelled if it were randomly labelled according to the distribution of labels in the subset. The value signifies various obstacles considered and in this case: $value$ = [crossingSign, movingCar, pedestrian, rock, speedSign, workOnroad, stopSign, traffickLightA, traffickLightB, treeOnRoad, staticCar, workSign].



Size ≤ 1.1993 gini = 0.8053 samples = 412 value = [39, 26, 55, 40, 156, 4, 31, 13, 11, 8, 6, 23] class = SpeedSign
True
False
Speed ≤ 0.0001 gini = 0.7244 samples = 148 value = [39, 0, 55, 0, 0, 0, 31, 0, 0, 0, 0, 23] class = Pedestrian
Size ≤ 1.6999 gini = 0.6123 samples = 264 value = [0, 26, 0, 40, 156, 4, 0, 13, 11, 8, 6, 0] class = SpeedSign
Size ≤ 0.5 gini = 0.6519 samples = 93 value = [39, 0, 0, 0, 0, 0, 31, 0, 0, 0, 0, 23] class = CrossingSign
gini = 0.0 samples = 55 value = [0, 0, 55, 0, 0, 0, 0, 0, 0, 0, 0, 0] class = Pedestrian
gini = 0.0 samples = 156 value = [0, 0, 0, 0, 156, 0, 0, 0, 0, 0, 0, 0] class = SpeedSign
Speed ≤ 8.492 gini = 0.7701 samples = 108 value = [0, 26, 0, 40, 0, 4, 0, 13, 11, 8, 6, 0] class = Rock
gini = 0.0 samples = 39 value = [39, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] class = CrossingSign
Size ≤ 1.0323 gini = 0.489 samples = 54 value = [0, 0, 0, 0, 0, 0, 31, 0, 0, 0, 0, 23] class = StopSign
Size ≤ 2.2639 gini = 0.7017 samples = 82 value = [0, 0, 0, 40, 0, 4, 0, 13, 11, 8, 6, 0] class = Rock
gini = 0.0 samples = 26 value = [0, 26, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] class = MovingCar
gini = 0.0 samples = 31 value = [0, 0, 0, 0, 0, 0, 31, 0, 0, 0, 0, 0] class = StopSign
gini = 0.0 samples = 23 value = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 23] class = WorkSign
Size ≤ 2.0645 gini = 0.4965 samples = 24 value = [0, 0, 0, 0, 0, 0, 0, 13, 11, 0, 0, 0] class = TrafficLightA
Size ≤ 2.517 gini = 0.4899 samples = 58 value = [0, 0, 0, 40, 0, 4, 0, 0, 0, 8, 6, 0] class = Rock
gini = 0.0 samples = 6 value = [0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0] class = TrafficLightB
Size ≤ 2.0845 gini = 0.4012 samples = 18 value = [0, 0, 0, 0, 0, 0, 0, 13, 5, 0, 0, 0] class = TrafficLightA
gini = 0.0 samples = 40 value = [0, 0, 0, 40, 0, 0, 0, 0, 0, 0, 0, 0] class = Rock
Size ≤ 7.5245 gini = 0.642 samples = 18 value = [0, 0, 0, 0, 0, 4, 0, 0, 0, 8, 6, 0] class = TreeOnRoad
gini = 0.0 samples = 7 value = [0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0] class = TrafficLightA
gini = 0.4959 samples = 11 value = [0, 0, 0, 0, 0, 0, 0, 6, 5, 0, 0, 0] class = TrafficLightA
gini = 0.48 samples = 10 value = [0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 6, 0] class = WorkOnRoad
gini = 0.0 samples = 8 value = [0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0] class = TreeOnRoad

Figure 5.27: Obstacle classification using decision tree.

Figure 5.28 shows the feature importance of the decision tree. As shown, the features that are the most important for the decision-tree classification algorithm, as per simulation result, is the obstacles size and speed; all others are not even considered, due to the Decision Tree simplicity.

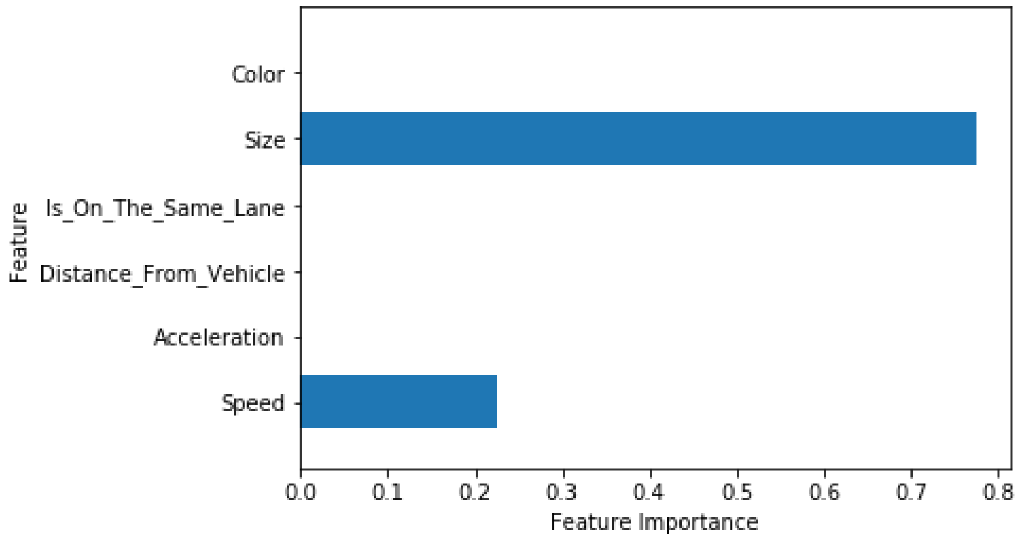
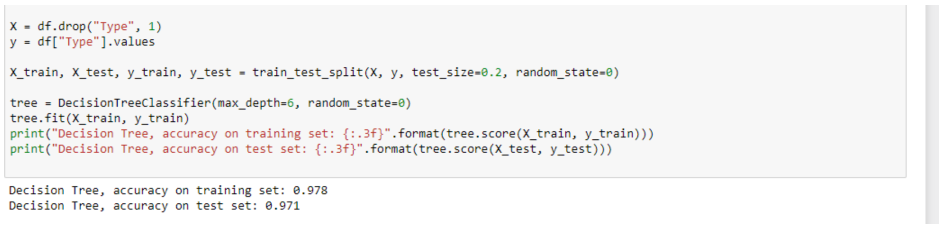


Figure 5.28: Obstacle classification using decision tree.

Figure 5.29 shows the decision tree score in the identification of an obstacle. It uses the data as 80% for training, 20% for testing. Accordingly, it obtains 97.8% accuracy in identifying obstacles in the training set and 97.1% in identifying the obstacles within the test set, showing a good results for this application.



```
X = df.drop("Type", 1)
y = df["Type"].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

tree = DecisionTreeClassifier(max_depth=6, random_state=0)
tree.fit(X_train, y_train)
print("Decision Tree, accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Decision Tree, accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Decision Tree, accuracy on training set: 0.978
Decision Tree, accuracy on test set: 0.971
```

Figure 5.29: Decision Tree Score.

Decision tree is normally used for its implementation simplicity and readability. It suits for simple application, like in this case, but if in future works new features or road objects will be added this could not be the best algorithm. Decision tree is often used as first algorithm to evaluate more complicated ones.

5.5.2 K-Nearest Neighbours (KNN)

kNN categorize data based on the class of the nearest object already present in the dataset: kNN is based on the assumption that near object belongs to the same class and acts in the same way. k value is defined as the number of neighbors that have to be considered for classify a new object. kNN weaknesses start to appear when a new data is good for one or more classes or can be part of another class if we consider more neighbors. The distance can be computed in different ways, such as the Euclidian distance for continuous variables like ours. The importance of each neighbor can be weighted; often the weight used is inversely proportional to the distance to give more importance to closer neighbor. Figure 5.30 shows the scores for different k values, with $k = [0, 10], k \in \mathbb{Z}$.

In this case it is possible to see how the kNN is not very suitable for this application, since many road objects have similar characteristics, hence the neighbours are sometimes very close and not really impacting.

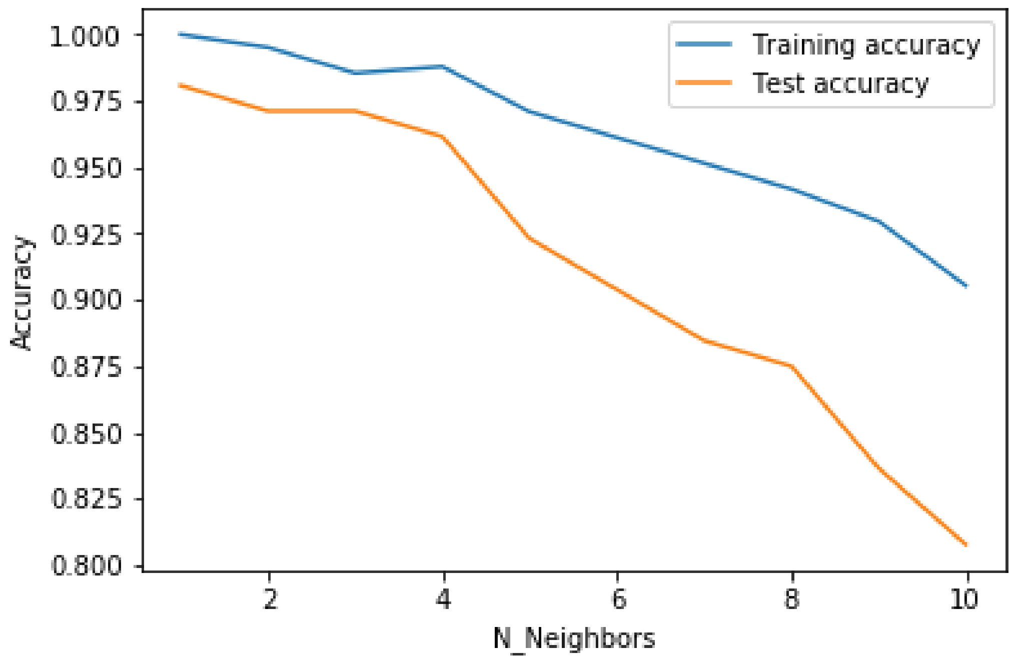
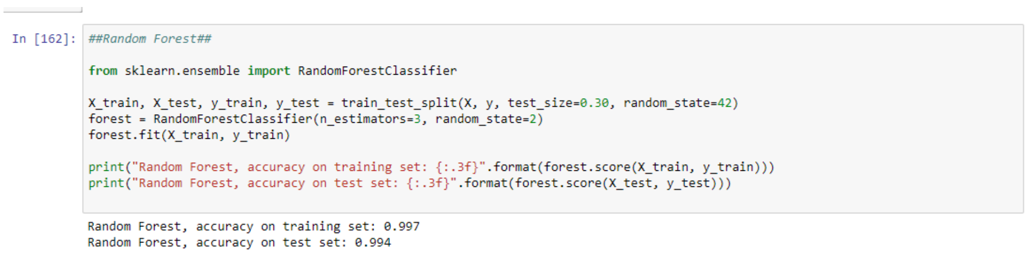


Figure 5.30: KNN predictions accuracy result

5.5.3 Random Forest

One of the main drawbacks of Decision Trees is the trend of overfitting the data. Random Forest algorithm is usually a solution to this problem. It is essentially a collection of decision trees, but each tree composing the forest is slightly different from the other ones. Basic idea is to use the decision tree overfitting property to create many trees that overfit in different ways different part of the data. The average of all the results will reduce the complex overfitting. Since this work is not really complex, the *n_estimators* parameter, that allows to define the number of tree created, is set to 3. Figure 5.31 shows the random forest score in the identification of an obstacle. It uses 70% of the data for training, and 30% for testing. Accordingly, it obtains 99.7% accuracy in identifying obstacles in the training set and 99.4% in identifying the obstacles within the test set. The results are better than the ones obtained using decision-tree learning algorithm. Figure 5.32 shows the feature importance for the Random Forest, highlighting that other features can concur for the final classification. Colour and acceleration are not very important since are used to distinguish traffic lights and moving objects, while the size is always the most important feature.



```
In [162]: ##Random Forest##

          from sklearn.ensemble import RandomForestClassifier

          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
          forest = RandomForestClassifier(n_estimators=3, random_state=2)
          forest.fit(X_train, y_train)

          print("Random Forest, accuracy on training set: {:.3f}".format(forest.score(X_train, y_train)))
          print("Random Forest, accuracy on test set: {:.3f}".format(forest.score(X_test, y_test)))

          Random Forest, accuracy on training set: 0.997
          Random Forest, accuracy on test set: 0.994
```

Figure 5.31: Random Forest Score

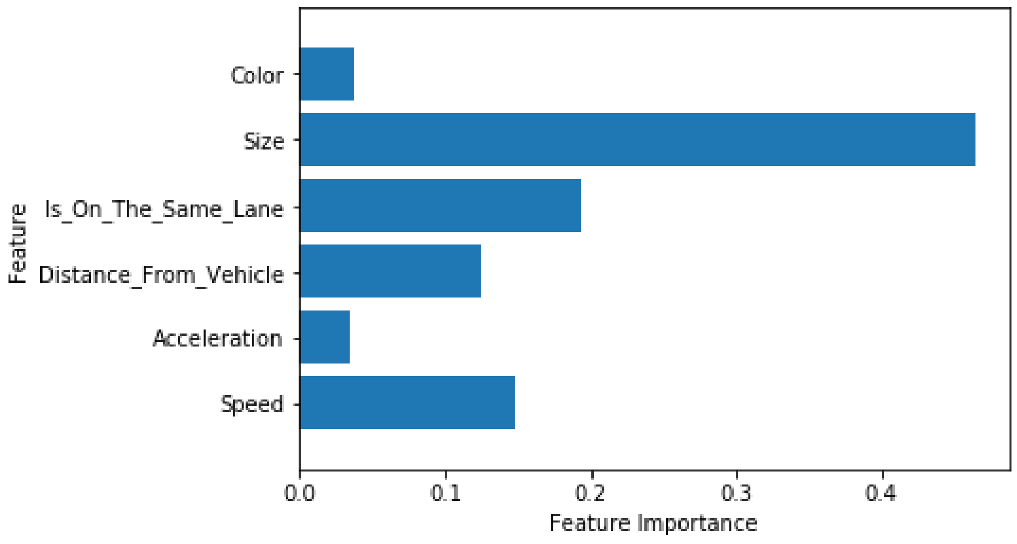


Figure 5.32: Random Forest Feature Importance

Random Forest obtained results show that it is a suitable algorithm also for simpler problems, giving good results with respect to a simple implementation.

5.5.4 Multilayer Perceptron (MLP)

A multilayer perceptron (MLP) is a deep, artificial neural network [41] composed of more than one perceptron. A perceptron is a binary classifier that, taking an input vector x, is able to return a $f(x)$ output value. The most common implementation of a perceptron is given in equation 5.1, where w is the weight vector, assuming real values, $\langle \cdot, \cdot \rangle$ operator corresponds to the dot product, b corresponds to the common *bias* and $\chi(y)$ is the output function, whose value is usually $\chi(y) = sign(y)$, giving only -1 and 1 as output.

$$f(x) = \chi(\langle w, x \rangle + b) \tag{5.1}$$

MLP are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input and, in between those two, an arbitrary number of hidden layers that are the true MLP computational engine. Training involves adjusting the parameters, or the weights and biases, of the model in order to minimize error. Back-propagation is used to make those weigh and bias adjustments relative to the error, and the error itself can be measured in a variety of ways, including by root mean squared error.

Figure 5.33 shows the weights that were learned connecting the input to the first hidden layer. The 6 features represent the rows, while the columns correspond to the 100 hidden units. On the right, the heat map legend shows that light colours correspond to positive values while dark colours to negative values. Features with very small weights are the less important ones, while increasing the weight of a feature means increasing its importance. For example the *Is_On_The_Same_Lane* feature is not really important in the obstacle classification, while speed and size cover a key role in it. Figure 5.34 Multilayer perceptron score (70% training, 30% test).

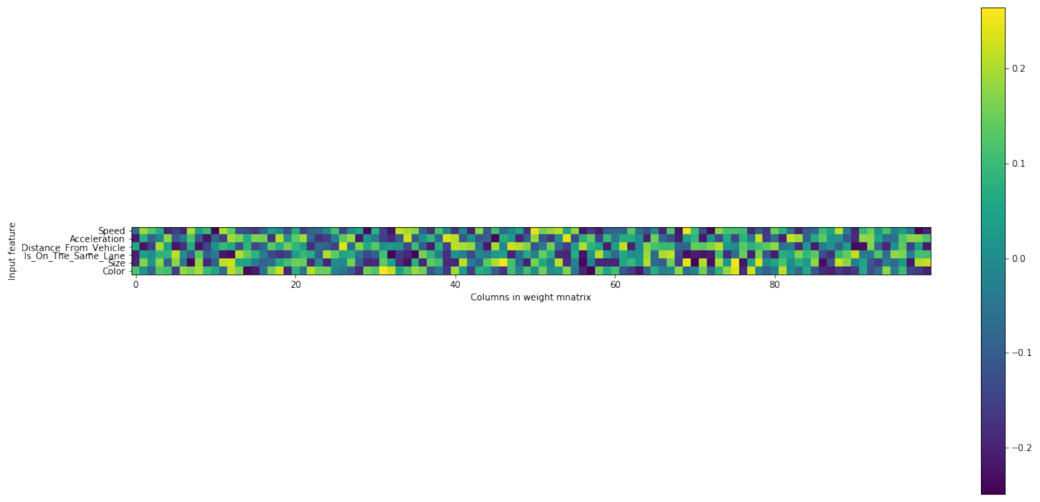
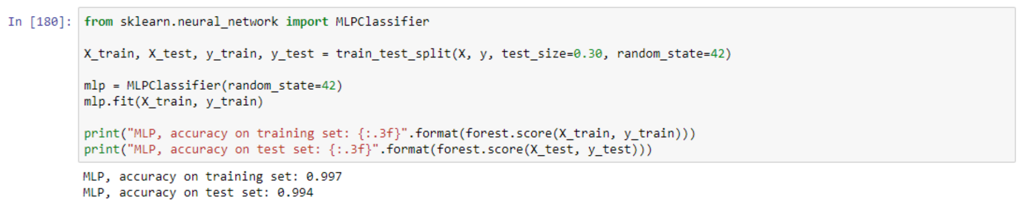


Figure 5.33: MLP Classifier



```
In [180]: from sklearn.neural_network import MLPClassifier

          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

          mlp = MLPClassifier(random_state=42)
          mlp.fit(X_train, y_train)

          print("MLP, accuracy on training set: {:.3f}".format(forest.score(X_train, y_train)))
          print("MLP, accuracy on test set: {:.3f}".format(forest.score(X_test, y_test)))

          MLP, accuracy on training set: 0.997
          MLP, accuracy on test set: 0.994
```

Figure 5.34: MLP Score

MLP is a good alternative to Random Forest, but it is more difficult to interpret and has a bigger computational cost.

5.6 Reinforcement Learning

Once the vehicle has detected an obstacle, the intelligent system has to manage the situation, taking actions in order to act in the right way. As illustrated in chapter 2, Reinforcement Learning main application is in the decision-taking domain. In this work, Reinforcement Learning will be used to teach the car to avoid an obstacle, returning to the right lane once the obstacle is overcome. This problem will be formulated as a MDP, where:

- The vehicle is the *Agent a.*
- The environment E is the simulated unity 3D scene.
- The states set S is composed of the vehicle positions at each frame.
- The actions set A is composed of the *steer, break and accelerate* actions.
- The policy π will be the overcome action.
- The reward function r based on the vehicle lane and collision with obstacles.

The reward function r would be based on the position of the vehicle with respect to the lane and on the collision with obstacles:

- Positive reward if the vehicle stays on the road and no collision occurs.
- Negative reward if the vehicle is no longer on the road and a collision is detected.

5.6.1 Issue On the Training Scene

The first action was to implement the ML-Agents components in the complete scene, where the ontology creation and Supervised Learning application where made. Since the ML-Agents plug-in is still an experimental tool, it was not possible to use it in the scene. The problem was related to the fact that a complex scene like the one created, with lot of imported assets, raised a socket problem communication between Unity and Python. This was also because of the vehicle implementation: the brain is not yet capable of take actions for the car wheels, leading to a non-moving situation. The problem was exposed to the Unity community and developers, opening a discussion on the issue that will be maybe solved in future updates. The problem raised was the following:

```
Exception
ArgumentException: An item with the same key has already been
added.
System.ThrowHelper.ThrowArgumentException (System.ExceptionRe
source resource) (at :0)
System.Collections.Generic.Dictionary2[TKey,TValue].Insert (T
Key key, TValue value, System.Boolean add) (at <a90417619fac4
9d5924050304d0280bb>:0) System.Collections.Generic.Dictionary2
[TKey,TValue].Add (TKeykey, TValue value) (at :0)
Brain.SendState (Agent agent, AgentInfo info) (at <4282bb72582
e4d569cd9952b328765a8>:0)
Agent.SendInfoToBrain () (at <4282bb72582e4d569cd9952b328765a8
>:0)
Agent.SendInfo () (at <4282bb72582e4d569cd9952b328765a8>:0)
Academy.EnvironmentStep () (at <4282bb72582e4d569cd9952b328765
a8>:0)
Academy.FixedUpdate () (at <4282bb72582e4d569cd9952b328765a8>:
0)
```

It is important to notice that with a *Player* brain the scene was perfectly working, with a satisfying reward function response and a working *Vector Observation*. The problem raised in the training phase, when the brain was set to *External* so it was impossible to train the model and come to a result for the original unity scene.

5.6.2 The Reinforcement Learning scene

In order to face this issue, a new scene was created. This scene was built following the same scale for the starting road of the original one, creating an obstacle with the dimension of a rock. The car was substituted by a Sphere game object, able to move thanks to a force applied to its *RigidBody*. Figure 5.35 shows the new unity scene, in which it is possible to see the 3 main actors:

- The light blue Sphere *Agent*.
- The red obstacle.
- The green target.

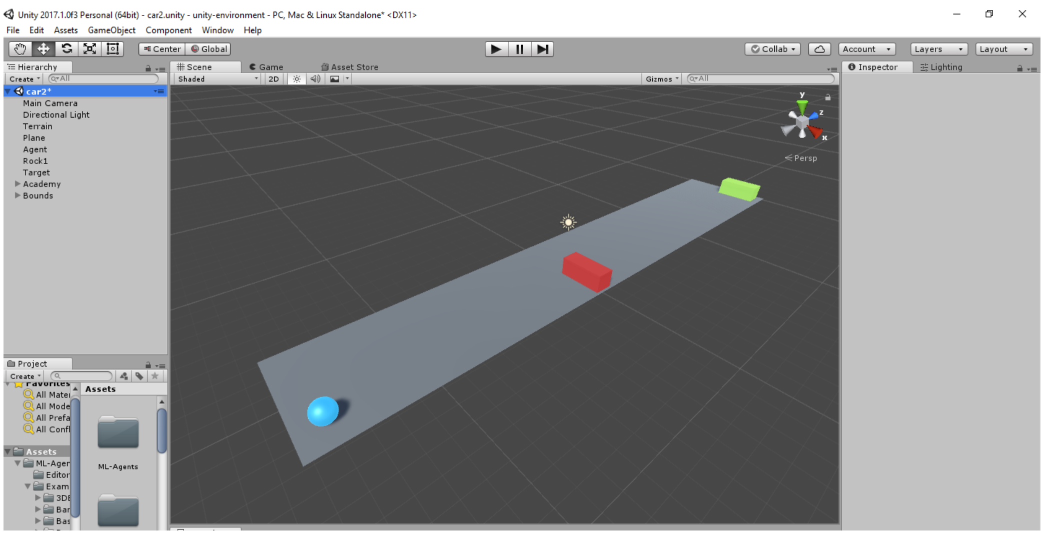

Unity 2017.1.0f3 Personal (64bit) - car2.unity - unity-environment - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Window Help
Hierarchy
car2*
Main Camera
Directional Light
Terrain
Plane
Agent
Rock1
Target
Academy
Bounds
Scene
Game
Asset Store
Inspector
Lighting
Project
Assets
ML-Agents
Persp

Figure 5.35: Reinforcement Learning scene

In this scenario, the sphere has to pass the obstacle staying on the road and collide with the green target returning to the correct lane.

Reward Function

The reward function was built considering this application, whose goal is to reach the object *target*. Equations 5.2 and 5.3 show the reward functions structure, where *collision* is a variable whose value is 1 if the sphere collides with the target and 0 if collides with the obstacle, *isOnLane* is a variable whose value is 1 as long as the sphere hits the road, otherwise its value is 0. *distanceToTarget* and *previousDistance* are variables updated every frame, representing the current and the previous distance between the target and the vehicle.

$$r = \begin{cases} \text{+1 if } collision = 1 \\ \text{+0.1 if } distanceToTarget < previousDistance \\ \text{-0.05 if } distanceToTarget > previousDistance \\ \text{-0.05 if } t_i < t_{i+1} \\ \text{-1 if } collision = 0 \mid\mid isOnLane = 0 \end{cases} \tag{5.2}$$

$$r = \begin{cases} \text{+1 if } collision = 1 \\ \text{+0.05 if } distanceToTarget < previousDistance \\ \text{-0.05 if } distanceToTarget > previousDistance \\ \text{-0.01 if } t_i < t_{i+1} \\ \text{-1 if } collision = 0 \mid\mid isOnLane = 0 \end{cases} \tag{5.3}$$

A key point in the reward function is the *time penalty* factor: -0.05 for function 5.2 and -0.01 for function 5.3. This small reward is added at every frame, expressed with mathematical condition $t_i < t_{i+1}$. The *time penalty* factor is very used in the RL reward function implementation, encouraging the agent to move and reach the target. The best performances were achieved with function 5.2, while function 5.3 showed high values of the reward function, but worst performances.

5.6.3 Results

This subsection presents 3 different training results, showing graphs concerning the most important statistics for the training phase:

1. *Cumulative Reward*: The mean cumulative episode reward over all agents (in this case, only one). It should increase during a successful training session.

2. *Entropy*: How random the decisions of the model are. It should slowly decrease during a successful training process.

All these 3 training were made using almost the same training parameters, explained in detail in Appendix C, with only two changes: the reward function used and the *max_step* parameter, that fixes the maximum number of step for the training phase. Next subsubsections will highlight problems and strength point in the training phase.

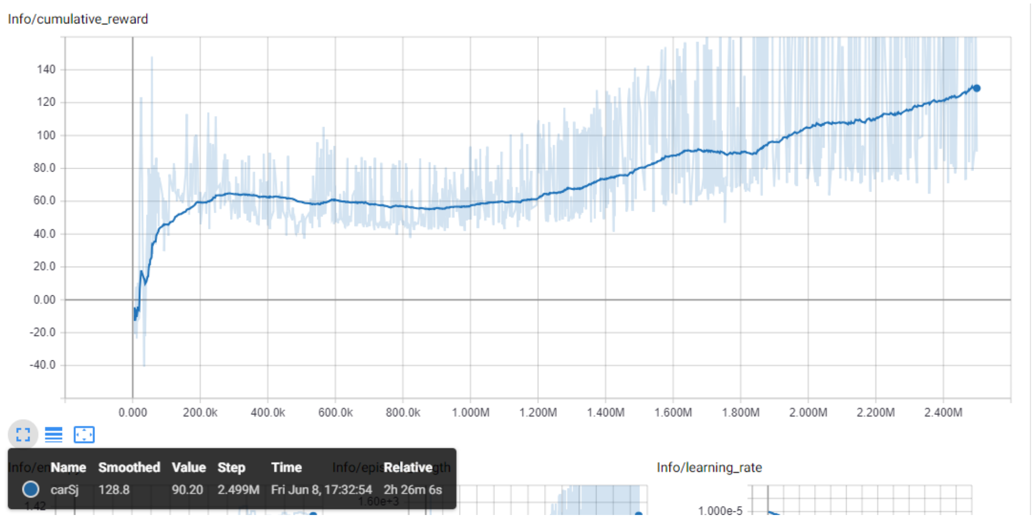
CarSimpleJ

The first good performances were accomplished using the following parameter, using the 5.3 reward function:

```
default:
    trainer: ppo
    batch_size: 4096
    beta: 1.0e-4
    buffer_size: 40960
    epsilon: 0.1
    gamma: 0.99
    hidden_units: 256
    lambd: 0.95
    learning_rate: 1.0e-5
```

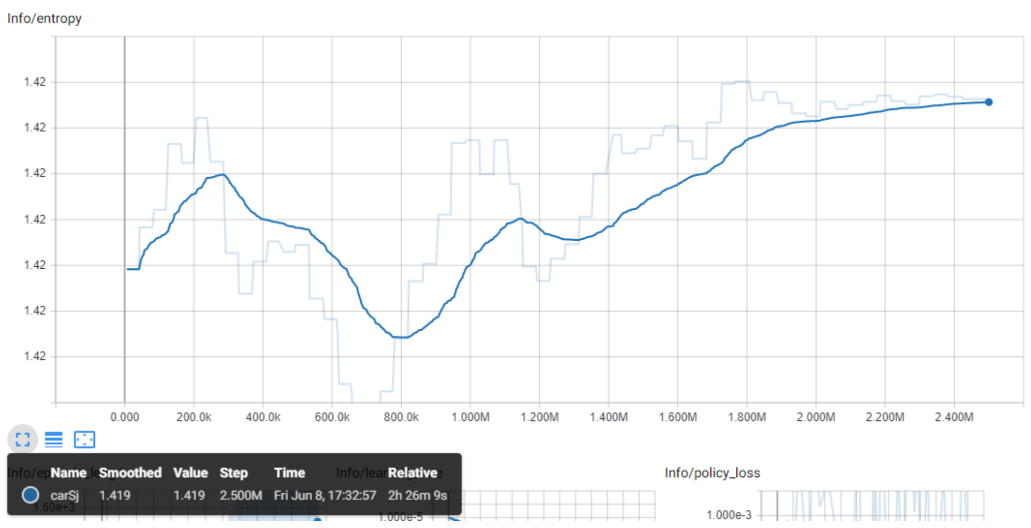
```
max_steps: 2.5e6

memory_size: 256

normalize: false

num_epoch: 3

num_layers: 2

time_horizon: 64

sequence_length: 64

summary_freq: 1000

use_recurrent: false
```

Figure 5.36 and 5.37 show the 2 graphs for the parameters *Cumulative Reward* and *Entropy*. Key point is that the reward is overall increasing, but has lot of peaks, both high and low. This is linked to the *Entropy* parameter that is not decreasing in the right way. This leads to a behaviour that sometimes gives a good results, sometimes not, with the agent that collides with the road or with the obstacle.



Name	Smoothed	Value	Step	Time	Relative
carSj	128.8	90.20	2.499M	Fri Jun 8, 17:32:54	2h 26m 6s

Figure 5.36: Cumulative Reward, carSj scene



Name	Smoothed	Value	Step	Time	Relative
carSj	1.419	1.419	2.500M	Fri Jun 8, 17:32:57	2h 26m 9s

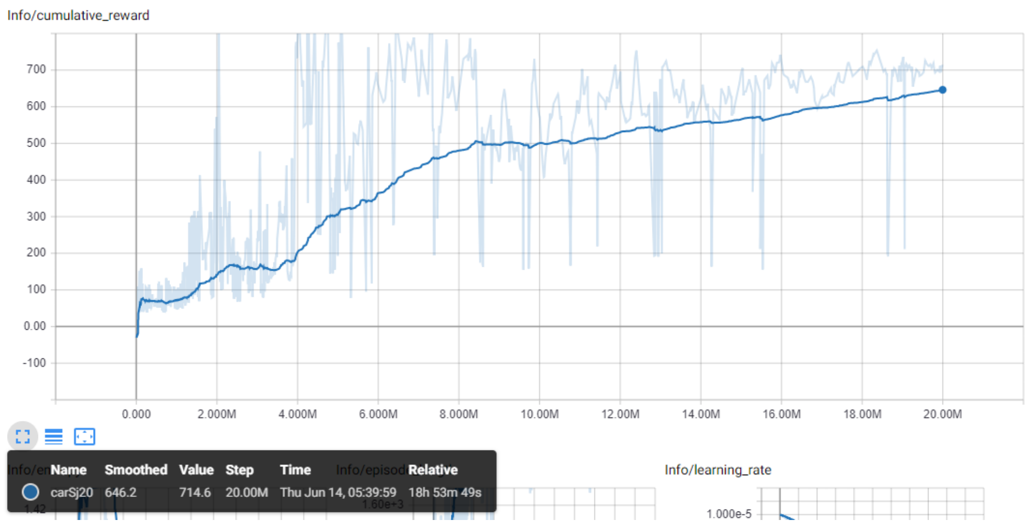
Figure 5.37: Entropy, carSj scene

With respect to previous simulations, this one was the first one giving a reward that was overall increasing, accomplishing the task. However, that was because of the small number of iterations, 2.5 millions. This result was the starting point for other simulations, since the parameters for the learning phase were suitable for the application.

carSimpleJ20

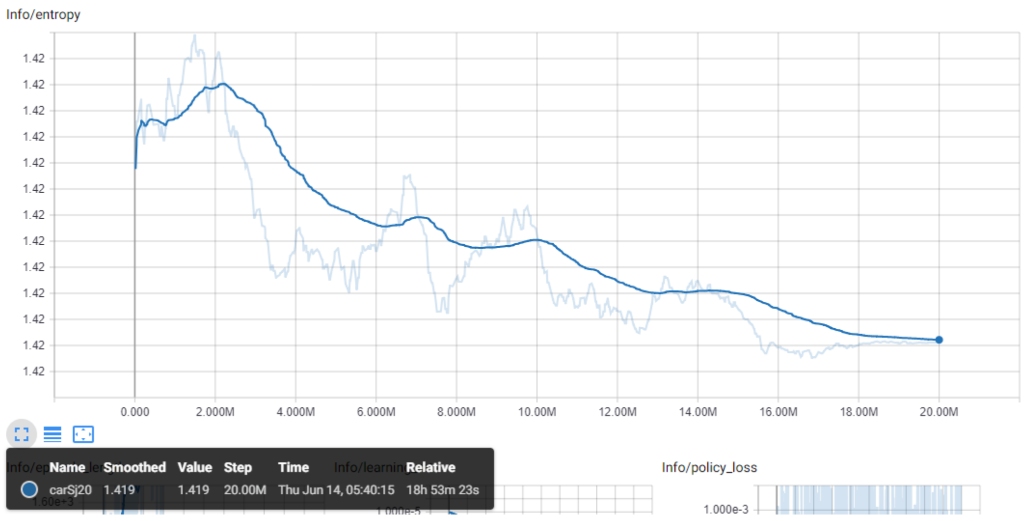
The only change that was made on the *max_steps* parameter, that was fixed to 20 millions, in order to see if the reward and entropy would have followed the correct behaviour. Figure 5.38 and 5.39 show the behaviour of the parameters, highlighting the correct trend, even with some low peaks for the reward.

From a numerical point of view, the results were very satisfying, but the problem was linked to the agent behaviour. It was going too slow, taking positive reward thanks to the fact it was getting closer to the target, but



Name	Smoothed	Value	Step	Time	Relative
carSj20	646.2	714.6	20.00M	Thu Jun 14, 05:39:59	18h 53m 49s

Figure 5.38: Cumulative Reward, carSj20 scene



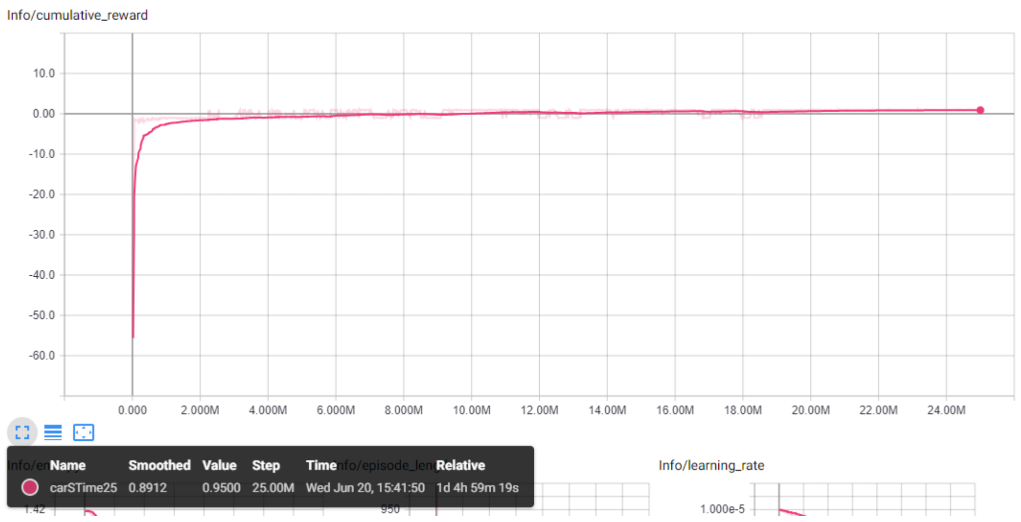
Name	Smoothed	Value	Step	Time	Relative
carSj20	1.419	1.419	20.00M	Thu Jun 14, 05:40:15	18h 53m 23s

Figure 5.39: Entropy, carSj20 scene

after have avoided the obstacle it was not able to return on the correct lane. This *overfitting* behaviour was caused due to the fact that the reward given for the target approaching was too high respect to the one given for the final goal accomplishment.

carSimpleTime25

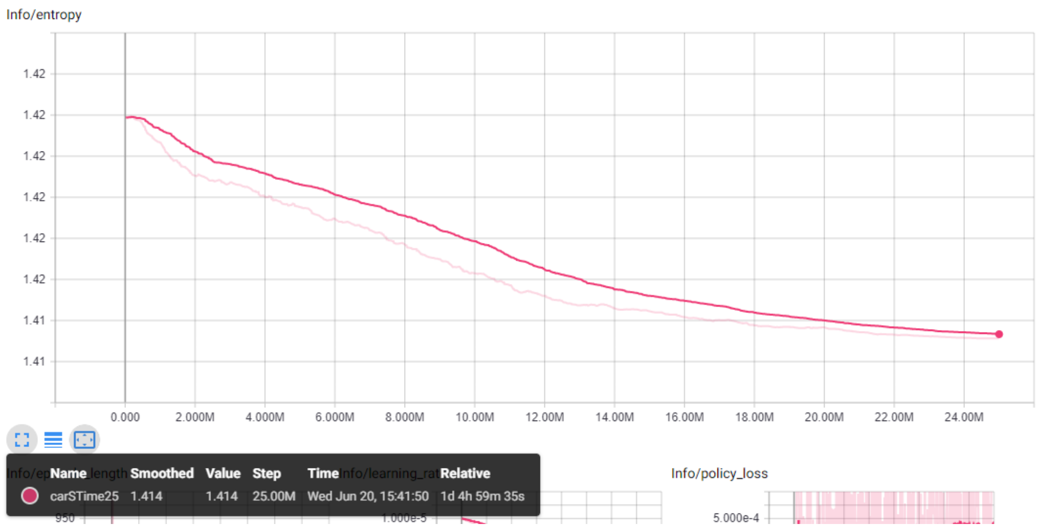
The behaviour obtained in the previous simulation suggested a change in the reward function. Equation 5.2 was adopted, changing the value assigned for the target approaching and time penalty. In Figure 5.40 and 5.41 it is possible to notice the immediate stabilization of the reward function, while the entropy is decreasing in the correct way.



Name	Smoothed	Value	Step	Time	Relative
carSTime25	0.8912	0.9500	25.00M	Wed Jun 20, 15:41:50	1d 4h 59m 19s

Figure 5.40: Cumulative Reward, carSTime25 scene

The behaviour is almost perfect, with the agent that is basically always able to avoid the obstacle and return in the correct lane, hitting the target. In this simulation the *max_steps* parameter was set to 25 millions.



Name	Smoothed	Value	Step	Time	Relative
carSTime25	1.414	1.414	25.00M	Wed Jun 20, 15:41:50	1d 4h 59m 35s

Figure 5.41: Entropy, carSTime25 scene

Chapter 6

Conclusions

6.0.1 Results

The biggest challenge in this project was to create a useful and re-usable application for Unity 3D, Ontology in Protégé and both Supervised Learning and Reinforcement Learning. This work contributed in the global system giving an improved road environment for the simulations, a new ontological building phase regarding the *Road Context* and two Machine Learning applications. As seen in previous sections, the final work presents satisfying results, both for the Supervised and Reinforcement Learning applications: this results are resumed in Table 6.1. In addition to this, the improved scene showed a working traffic light system, that is correctly handled by the 4 different moving cars. The Ontological representation phase also presented a correct behaviour, creating the right individuals representing the Unity3D objects.

Among the 4 algorithms used for the Supervised Learning technique, De-

Machine Learning Type	Algorithm/Scene	Result
Supervised Learning	Decision Tree	97.8% for the training set and 97.1% for the test set.
	kNN	Ratio decreasing with the k increase: not suitable.
	Random Forest	99.7% for training set and 99.4 % for the test set.
	MLP	99.7 % for the training set and 99.4 % for the test set.
Reinforcement Learning	carSj	Good reward trend, not the entropy one. Behaviour correctness borderline.
	carSj20	Perfect reward and entropy trend. Training lead to unusable model.
	carSTIme25	Perfect reward and entropy trend. Perfect model for simulated obstacle avoidance.

Table 6.1: Summary of ML results

cision Tree and Random Forest are the most suitable ones. At the current state of the work, Decision Tree is the best one, because of simplicity, readability and computational time, even if MLP and Random Forest are giving better results. However, MLP and Random Forest are not really suitable because of the scenario simplicity, leading to an high overfitting probability.

6.0.2 Future Works

The complexity and the dimension of the global system allow several future works to be done. The most important future work will be the real implementation of the system, passing from the simulated data collection to real ones. Related to this, the ontology creation for the last *context*, with a deep analysis of the driver status, will complete this part of the system. One of the problem in the current state is the amount of data and road object variety. Future works should be focused on other real road object implementation, in order to create a more realistic environment; *Animals* and *Bikes* implementations would be one of the possible addition that could be made in the environment. Future works related to the Supervised Machine Learning would be focused on the individuation of the best algorithm with respect to the system grow, focusing also on the validation part for the algorithms. Reinforcement Learning is the part the most improvable: future works should be focused not only on the real implementation, but also to create a working training scenario for the complete simulated scene. Next big step in the ML application would be the implementation of a Deep Learning image recognition, with the usage of cameras in order to collect more and different data. This implementation would mean a more realistic and complete system, giving more useful data to the SL and RL applications.

Appendix A

Building the Ontology: Data & Methodology

The data used for the ontology creation has to follow a specific format. In the csv file each screenshot representing the environment is saved in a unique string, that is then processed by the *JAVA* program. After the precessing phase, the *JAVA* program create and instantiate a real ontology for screenshot in the csv file. An example of a string showing only the *objects* saved in a csv file is shown below:

"0":"type":"WorkSign","speed":0,"acceleration":0,"distance":44.11644, "position":"(3.0, -79.9, -49.0)","rotation":"(270.0, 180.0, 0.0)","rotationRelative":"175.1878","directionRelative":
"2.732076E-05","isOnLane":0,"isOnLaneRelative":0,"roadAngle":0,
"weight":0,"size":0.08708858,"color":off,

"1":"type":"SpeedSign","speed":0,"acceleration":0,"distance":74.77336,

"position":"(33.7, -78.8, -49.4)","rotation":"(0.0, 0.0, 0.0)","rotationRelative":"177.023","directionRelative":
"-89.99989","isOnLane":0,"isOnLaneRelative":0,"roadAngle":0,
"weight":0,"size":0.1856232,"color":off,

"2":"type":"WorkOnRoad","speed":0,"acceleration":0,"distance":51.33749,
"position":"(10.3, -80.5, -46.4)","rotation":"(0.0, 90.0, 0.0)","rotationRelative":"133.4264","directionRelative":
"-0.337466","isOnLane":1,"isOnLaneRelative":1,"roadAngle":0,
"weight":0,"size":13.79253,"color":off,

"3":"type":"Rock","speed":0,"acceleration":0,"distance":59.54245,
"position":"(-58.2, -79.0, -103.3)","rotation":"(270.0, 0.0, 0.0)","rotationRelative":"73.23051","directionRelative":
"2.732076E-05","isOnLane":1,"isOnLaneRelative":0,"roadAngle":0
,"weight":0,"size":3.143028,"color":off,

"4":"type":"TreeOnRoad","speed":0,"acceleration":0,"distance":23.036,
"position":"(-44.3, -80.2, -69.1)","rotation":"(90.0, 270.0, 0.0)","rotationRelative":"81.84576","directionRelative":
"-1.366038E-05","isOnLane":1,"isOnLaneRelative":0,"roadAngle":0,
"weight":0,"size":16.20387,"color":off

Here the data are collected only in the *CarController* function, adding in the string the totality of the objects present in the list. As it is possible to see, there are 14 different variables collected (*type, speed, acceleration, distance, position, rotation, rotationRelative, directionRelative, isOnLane, isOnLaneRelative, roadAngle, weight, size and color*) plus the *id* variable, used just for a checking purpose.

Appendix B

Supervised Data: format & collections

For the Supervised application the data were saved in a predefined format, that allowed to use them in python through the *panda* package. Every game object has 7 variables characterizing it: Type, Speed, Acceleration, Distance_From_Vehicle, Is_On_The_Same_Lane, Size, Color. Here is how the data look like in a csv file example:

WorkOnRoad,0,0,51.33749,0,6.408485,0

Rock,0,0,59.54245,1,2.336617,0

TreeOnRoad,0,0,23.036,0,8.640416,0

SpeedSign,0,0,60.76823,0,1.337128,0

WorkSign,0,0,91.54299,0,1.064657,0

WorkSign,0,0,24.93469,0,1.064657,0

StopSign,0,0,14.46818,1,1,0

Rock,0,0,78.07341,-1,2.336614,0

```
WorkSign,0,0,58.30757,0,1.064657,0
WorkSign,0,0,44.11644,0,1.064656,0
SpeedSign,0,0,74.77336,0,1.333961,0
```

The data collection is composed of 2 parts, both made in C# through Unity scripts: *RoadObject* and the *CarController.* The former one writes a string for every game object belonging to this class, already in the correct format, while the latter is in charge of write the csv file. The 3 lists created in the *CarController* script (*RoadObject, CloseRoadObject and Obstacle*) write the string for every road object with the correct characteristics.

Appendix C

Agent Observations & Training Parameters

This appendix presents the Agent Observations made by the agent and explain in details the training parameters.

C.1 Observations

- AddVectorObs(isOnLane): allow to observe if the agent is still on the Road.
- AddVectorObs(collision): observe the collision variable. 1 if obstacle, 2 if the target.
- AddVectorObs(rigidbody.velocity); AddVectorObs(rigidbody.angularVelocity): current velocities.
- AddVectorObs(Target.position.x / maxDistanceBoundX): Target posi-

tion (also on the z plane)

- AddVectorObs((Target.position.x - this.transform.position.x) / maxDistanceBoundX): agent position with respect to the target (also on the z plane).
- For each obstacle: distance, position and collider bounds.
- AddVectorObs((this.transform.position.x - bound_x1.position.x) / maxDistanceBoundX): agent distance from the Road bounds; the same obersvation is made for the other 3 bounds.

C.2 Training Parameters

- Gamma : corresponds to the discount factor for future rewards. This can be thought of as how far into the future the agent should care about possible rewards. **0.99**
- Lambda : corresponds to the lambda parameter used when calculating the Generalized Advantage Estimate (GAE). This can be thought of as how much the agent relies on its current value estimate when calculating an updated value estimate. **0.95**
- Buffer Size : corresponds to how many experiences (agent observations, actions and rewards obtained) should be collected before we do any learning or updating of the model. Should be a multiple of batch_size. **40960**

- Batch Size : is the number of experiences used for one iteration of a gradient descent update. This should always be a fraction of the buffer_size. **4096**

- Number of Epochs : is the number of passes through the experience buffer during gradient descent. **3**

- Learning Rate : corresponds to the strength of each gradient descent update step. **1.0e-5**

- Time Horizon : corresponds to how many steps of experience to collect per-agent before adding it to the experience buffer. When this limit is reached before the end of an episode, a value estimate is used to predict the overall expected reward from the agent's current state. **64**

- Max Steps : corresponds to how many steps of the simulation (multiplied by frame-skip) are run during the training process. **2.5e6 - 2.5e7**

- Beta : corresponds to the strength of the entropy regularization, which makes the policy "more random." This ensures that agents properly explore the action space during training. Increasing this will ensure more random actions are taken. **1.0e-4**

- Epsilon : corresponds to the acceptable threshold of divergence between the old and new policies during gradient descent updating. **0.1**

- Number of Layers : corresponds to how many hidden layers are present after the observation input, or after the CNN encoding of the visual observation. For simple problems, fewer layers are likely to train faster and more efficiently. More layers may be necessary for more complex control problems. **2**

- Hidden Units : correspond to how many units are in each fully connected layer of the neural network. **256**

Bibliography

[1] A. Ortalda, M. D. Hina, A. Soukane, A. Moujahid, and A. Ramdane-Cherif, “Safe Driving Mechanism: Detection, Recognition and Avoidance of Road Obstacles,” *KEOD*, 2018, submitted: 2018-06-11.

[2] M. William, B.-B. Cristopher, and B. Lawrence, *Reinventing the automobile: Personal urban mobility for the 21st century.* MIT PRESS, 2010.

[3] “Research and Innovation,” 2018, URL: http://www.acea.be/industry-topics/tag/category/research-and-innovation [accessed: 2018-03-01].

[4] “ AAPC 2016 Research and Development Report,” 2016, URL: http://www.americanautocouncil.org/research-development-0 [accessed: 2018-03-01].

[5] Mathworks, *Machine Learning with MATLAB.* Mathworks, 2018, ch. 1–4. [Online]. Available: https://mathworks.com/campaigns/products/offer/machine-learning-with-matlab.html

[6] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, "Traffic flow prediction with big data: A deep learning approach," *IEEE Transactions On Intelligent Transportation Systems*, vol. 16, no. 2, pp. 865–873, April 2015.

[7] S. Kaplan, M. A. Guvensan, A. G. Yavuz, and Y. Karalurt, "Driver Behavior Analysis for Safe Driving: A Survey," *IEEE Transactions On Intelligent Transportation Systems*, vol. 16, no. 6, pp. 3017 – 3032, 2015.

[8] Tesla. [Online]. Available: https://www.tesla.com/autopilot

[9] Waymo. Accessed 22/01/2018. [Online]. Available: https://waymo.com/

[10] INTEL. Advanced driver assistant system: Threats, requirements, security solutions. Accessed 19/01/2018. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/advanced-driver-assistant-system-paper.pdf

[11] "Safety," 2018, URL: http://www.acea.be/industry-topics/tag/category/safety [retrieved: june, 2018].

[12] A. Moujahid, A. Ortalda, M. D. Hina, A. Soukane, and A. Ramdane-Cherif, "Machine Learning Solutions for ADAS," *ICACCE*, 2018, submitted: 2018-03-20.

[13] New-York-Telegraph. Driving investment returns through active safety. Accessed 19/01/2018. [Online]. Avail-

able: http://www.telegraph.co.uk/sponsored/finance/investments/habitat/car-road-safety-technology.html

[14] Road crash statistics. Accessed 19/01/2018. [Online]. Available: http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics

[15] BOSCH. Chassis systems control — bosch study on driver assistance systems 2012. Accessed 19/01/2018. [Online]. Available: http://www.bosch-presse.de/pressportal/de/media/migrated-download/de/7966ks-e-Detailed-information-driver-survey.pdf

[16] I.-H. Kim, J.-H. Bong, J. Park, and S. Park, "Prediction of drivers intention of lane change by augmenting sensor information using machine learning techniques," *SAICSIT*, pp. 47–55, 2013.

[17] Subaru-Forester. Pre-collision braking system. Https://carmanuals2.com/get/subaru-forester-2014-pre-collision-braking-system-32039.

[18] Mitsubishi-Motors. (1998, December) Mitsubishi motors develops "new driver support system". Accessed 19/01/2018. [Online]. Available: http://www.mitsubishi-motors.com/en/corporate/pressrelease/corporate/detail429.html

[19] FCA. (2018). [Online]. Available: https://www.fiat.it/fiat-500x

[20] E. Snyder. (2015). [Online]. Available: https://www.edgarsnyder.com/car-accident/who-was-injured/teen/teen-driving-statistics.html

[21] Mercedes-Benz. (2017, September) What is mercedes-benz attention assist? Accessed 19/01/2018. [Online]. Available: http://mercedesbenz.starmotorcars.com/blog/what-is-mercedes-benz-attention-assist/

[22] V. Jha, "Study of machine learning methods in intelligent transportation systems," December 2015.

[23] Y. Hou, P. Edara, and C. Sun, "Modeling mandatory lane changing using bayes classifier and decision trees," *IEEE Transactions On Intelligent Transportation Systems*, vol. 15, no. 2, pp. 647–655, April 2014.

[24] M. T. Jones and IBM. (2017, December) Unsupervised learning for data classification. [Online]. Available: https://www.ibm.com/developerworks/library/cc-unsupervised-learning-data-classification/index.html

[25] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Second Edition).* MIT Press, 2017.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[27] M. A. Nielsen, *Neural Networks and Deep Learning.* Determination Press, 2015, ch. 1.

[28] P. Tchankue, J. Wesson, and D. Vogts, "Using machine learning to predict the driving context whilst driving," *MDIP*, sensors 2017, 17, 1350; doi:10.3390/s17061350. [Online]. Available: http://www.mdpi.com/journal/sensors

[29] C. DAgostino, A. Saidi, G. Scouarnec, and L. Chen, "Learning-based driving events classification," in *Proceedings of the 16th International IEEE Annual Conference on Intelligent Transportation Systems (ITSC 2013), The Hague, The Netherlands*, October 6-9 2013.

[30] A. Jahangiri and H. A. Rakha, "Applying machine learning techniques to transportation mode recognition using mobile phone sensor data," *IEEE Transactions On Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2406–2417, October 2015.

[31] C. Miyajima, Y. Nishiwaki, K. Ozawa, T. Wakita, K. Itou, K. Takeda, and F. Itakura, "Driver modeling based on driving behavior and its evaluation in driver identification," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 427–437, February 2007.

[32] A. Ortalda, M. D. Hina, A. Soukane, and A. Ramdane-Cherif, "Analysis on Distraction Factors of Young Drivers," *HUSO special track*, 2018, submitted: 2018-06-11.

[33] UnityTechnologies. Accessed 04/06/2018. [Online]. Available: https://unity3d.com/

[34] G. Brain. Accessed 06/06/2018. [Online]. Available: https://www.tensorflow.org/

[35] GoogleBrain. Tensorflow. Accessed 11/06/2018. [Online]. Available: https://www.tensorflow.org/

[36] R. J. Hoekstra, “Ontology representation : design patterns and ontologies that make sense,” 2009, iOS press.

[37] StanfordUniversity. Protg. Accessed 04/06/2018. [Online]. Available: https://protege.stanford.edu/

[38] Oracle. Accessed 04/06/2018. [Online]. Available: https://developer.oracle.com/java

[39] C. Thierry, M. D. Hina, A. Soukane, and A. Ramdane-Cherif, “Ontological and machine learning approaches for managing driving context in intelligent transportation,” *KEOD*, 2017.

[40] EYEmaginary. Aicar. Accessed 14/06/2018. [Online]. Available: http://www.eyemaginary.com/

[41] DL4J. Guide to multilayer perceptron. Accessed 14/06/2018. [Online]. Available: http://www.eyemaginary.com/