Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería electrónica industrial y automática

Joined Optimization of Traffic Signaling and Vehicle Speed Advisory in V2I-Enabled Traffic with Deep Reinforcement Learning

Autor:

Fernández Sánchez, Nerea

Eusebio de la Fuente

Technische Universität Dresden

Valladolid, septiembre y 2021.

TFG REALIZADO EN PROGRAMA DE INTERCAMBIO

TÍTULO:	Joined Optimization of Traffic Signaling and Vehicle Speed Advisory in V2I- Enabled Traffic with Deep Reinforcement Learning
ALUMNO:	Nerea Fernández Sánchez
FECHA:	23/09/2021
CENTRO:	Faculty of Electrical and Computer Engineering
UNIVERSIDAD:	Technische Universität Dresden
TUTOR:	Johaness Busch

RESUMEN

En nuestra sociedad, tenemos un problema muy grande que nos causa una gran cantidad de pérdidas tanto directa como indirectamente: la congestión del tráfico. Podemos observar en varios estudios datos que aportan evidencias sobre los efectos producidos.

Para intentar dar solución a este problema tenemos tecnologías emergentes como el 5G, el big data, la V2I y el DRL. Con la comunicación entre la infraestructura del tráfico y los vehículos podemos conseguir situaciones particularizadas en cada momento.

Previamente se había comprobado que el DRL da buenas soluciones para controlar la señalización de los semáforos. Buscamos conseguir integrar con DRL la señalización del tráfico y las recomendaciones de velocidad para los vehículos que se aproximen, en este caso al escenario que hemos elegido que es una intersección simple. Conseguir integrar ambas va a conseguir solucionar la gran mayoría de los problemas asociados, pero sobre este último punto no se han realizado estudios.

ABSTRACT

In our society, we have a very big problem that causes us a great deal of losses both directly and indirectly: traffic congestion. We can observe in several studies data that provide evidence of the effects produced.

To try to provide a solution to this problem we have emerging technologies such as 5G, big data, V2I and DRL. With communication between traffic infrastructure and vehicles we can achieve particularized situations at each moment.

DRL has previously been proven to provide good solutions for controlling traffic light signaling. We seek to integrate with DRL the traffic signaling and speed recommendations for approaching vehicles, in this case to the scenario we have chosen, which is a simple intersection. Integrating both will solve most of the associated problems, but no studies have been carried out on this last point.

PALABRAS CLAVE

Reinforcement learning, machine learning, Deep Reinforcement learning, Artificial Intelligence, V2I.

KEYWORDS

Reinforcement learning, machine learning, Deep Reinforcement learning, Artificial Intelligence, V2I.

TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Electrical and Computer Engineering Deutsche Telekom, Chair of Communication Networks

Student Project

Joined Optimization of Traffic Signaling and Vehicle Speed Advisory in V2I-Enabled Traffic with Deep Reinforcement Learning

Nerea Fernández Sánchez

Erasmus student
Home University: Universidad de Valladolid, Spain
Matriculation year: 2021

02.09.2021

Supervisor
Dipl. Ing. Johannes Busch
Supervising professor
Prof. Dr. -Ing.Dr.h.c Frank H. P. Fitzek

INDEX

1. ABSTRACT

In this project we manage to find a Deep Reinforcement Learning (DRL) system that jointly optimizes traffic signal control and speed advisory. A literature review of all the concepts involved in the final solution is going to be made. We introduce all the concepts involved in DRL in order to perfectly understand how it works and how we must implement it for arriving at our final solution. We will write a review about the current state of the art for Deep Reinforcement Learning in traffic control in order to understand where we start from. How this new project must be implemented in order to have our whole system is also going to be explained. For better understand how the simulations with different systems work, different scenarios are going to be explored and the results of this simulations are going to be explained and presented. We will also do an exploration between how everything works with Vehicle to Infrastructure (V2I) Communication and how it changes when we do not have this information. In this project, how the software and programs needed have been installed is also explained.

2. INDEX TERMS

Deep Reinforcement Learning, traffic control, Vehicle to Infrastructure (V2I), Flow, Sumo, RLlib.

3. INTRODUCTION

Today, in our modern society, we live trying to find the fastest way for everything [7]. In terms of mobility, being as quick as possible has become as important as a basic need. Despite all the options we have, particular cars are still the most popular. This leads us to one big problem; vehicles are increasing faster than the available traffic infrastructure [7]. The result: traffic congestion [9]. This big problem affects many aspects of our modern society such as “economic development,accidents, greenhouse emissions, time spent, and health issues”. [7]

In this project, we will be doing a review of the current state of the art of the DRP for intelligent traffic control. The system that we have developed is going to be explained, as well as the experiments that we have carried out and the results and conclusion that we have obtained. In order to make the project as clear as possible, we are going to be explaining all the programs that have been used and its installation process.

A long part of this project has involved all the code to make the training and the after-trained simulations. This code it is not included in the written part.

3.1 CURRENT GLOBAL SITUATION

According to a research carried out by Inrix (one of the biggest specialists in transport data) in 2017, “the US emerged as the most congested developed country in the world, with drivers averaging 41 hours a year in peak hour traffic, at a cost of 1445 dollars a driver. [Another and bigger study that was undertaken by the UK-based Centre for Economics and Business Research (CEBR), that analyzed the direct costs, wasted time, the fuel and the indirect effects]; congestion costs US, France, Germany and the UK an estimated 200 billion dollars in 2013. That’s 1,736 dollars for each US family and over 2,000 dollars a household in the UK, France and Germany. Without significant action, these costs are predicted to increase almost 50 per cent to 291 billion dollars by 2030. Traffic congestion costs EU member states approximately €100 billion annually, with forecasts suggesting this could rise to €150 billion by 2050“[6]. Due to the global situation

in 2019 caused by the COVID-19, another study carried by Inrix shows that collisions dropped considerably in all countries in the world, but the collisions that occurred become more deadly. Traffic volume dropped, but vehicle speed increased and this is one of the main causes for accidents becoming more deadly [12]. When we consider all this data, we perfectly understand that we need to start making changes. This is the starting point for this project. We need to develop more complex and effective systems. Only this will move us to a better society where all the problems considered before, derived from traffic congestion, could disappear or at least, be mitigated.

Now, we can understand why it is not only a problem in terms of traffic. Signal control is basic in order to avoid awful situations in traffic, but also it has been shown that is very important to have an adequate speed. As explained before, when speaking about speed in very important not only to think about congestion, stress and money, but also about the catastrophic results that we can have.

3.2 SOLUTIONS

With the new Vehicle-to-Infrastructure (V2I) communication technologies we are able to explore and develop intelligent traffic infrastructures that are able to adapt to the current state of the traffic [9].

These technologies allow the exchange of information between individual vehicles and the road infrastructure. For making this information exchange possible we have upcoming standards such as IEEE 802.11p and 5G. [7] The scheme presented in Figure 1, shows everything that has been discussed before.

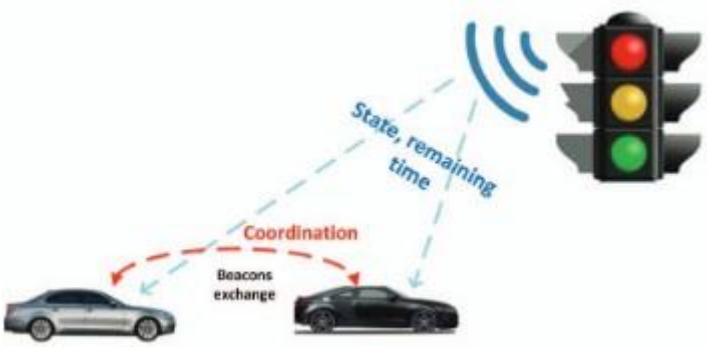

Figure 1: *V2I communication [9]*

All this considered, the optimization of large distribution systems with the possibility of knowing the situation of the current system is not that simple. One of the best solutions we can have is Deep Reinforcement Learning (DRL), a tool that has been proved to be very powerful for control and that “has demonstrated to have success in complex, data-rich problems”. [23]

DRL has proved to have great potential for large scale and dynamic systems. Generally speaking, in DRL, a deepneural network, (DQN), “is used to accelerate the learning process and also reduce the memory required to store the parameters of the model.This makes DRL perfect for mobile devices with limited resources. Deep Reinforcement Learning is promising to address mobile service and network management and control problems in complex, dense, and heterogeneous mobile 5G environments”. [24]

4. OBJECTIVE

The main objective of this project is to enable the joint optimization of traffic signaling control and speed advisory. Different experiments are going to be carried out in order to compare the results when we have V2I communication and no V2I communication and when is only the traffic signaling what we are going to control and we have both, the traffic signaling and the speed advisory part.

This is the goal of the project, but not only that is going to be set as our objective. We are starting to implement DRL from a basic base. With this project I set as an objective to have the ability, in the end, of understanding how everything is needed to be set up in order to develop a complete DRL system that we can optimize. Also, is very important that by the end of the project, I could have the knowledge and the ability of analyzing the results and interpreting them, in order to understand what is going on, and how it could be improved. I would also add to the global objective, having the ability, in the end, of coding everything that needs to be code for a whole experiment. In addition, understand and set up the requirements for each experiment.

5. STATE OF THE ART

5.1 REINFORCEMENT LEARNING

"Reinforcement Learning (RL) dates to the early days of cybernetics and work statistics, psychology, neuroscience, and computer science" [30]. Thanks to RL big steps in Artifial Intelligence has been made. [31]

When we talk about Reinforcement Learning, we are talking about an agent (that could be for example and autonomous car) that learns how to achieve a specific goal by dynamically interacting with its environment [11]. "RL is an AI subdomain allowing agent to fulfill a given goal while maximizing a numerical reward signal". [31]

"The agent learns how to map situations to actions in order to maximize that numerical reward signal". [11] Our chosen agent has a set of possible actions, and it discovers those that will lead him to the greatest reward at each setting by trying them out. We can say that RL is very useful in situations where data arrives in a continuous way, and we need our agent to be capable of adapt himself to the environment in real time [11].

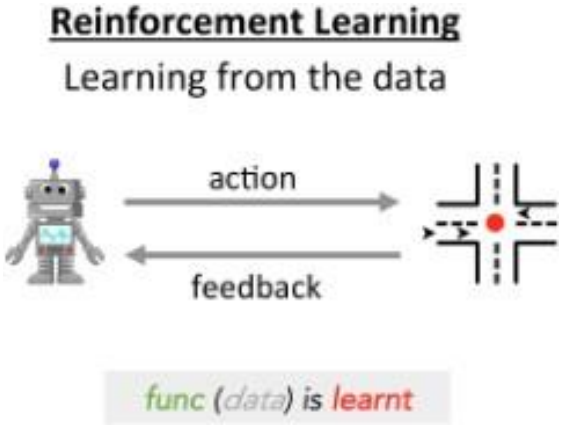

Figure 2: *RL Scheme [22]*

The problems in which "RL algorithms are applied can be viewed as Markov decision processes (MDPs)" [25]. "An MDP can be expressed as a four tuple *S, A, R, T* with the following meanings":[25]

- S: means the "possible state space. *s* is a specific state (*s ϵ S*)" [15].
- A: means the "possible action space. *a* is an action (*a ϵ A*)" [15].
- R: means the "reward space. $R_{s,a}$ means the reward in taking action *a* at state *s"* [15].
- T: means the "transmission function space among all states, which means the probability of the transmission from one state to another. In a deterministic model, T is usually omitted". [15]

In an environment, the agent receives "*st ϵ S* and chooses an action *at.* After taking the chosen action, the environment will generate *st+1 ϵ S* based on the transition function T" [25]. Once we have defined all the parameters of MDP we need to determine the optimal policy *π* for every state, "maximizing the expected cumulative discounted reward.If some actions lead to high reward" [26] the possibility of taking these actions will increase, and vice versa. The goal of RL algorithms is maximizing the cumulative discounted reward. The cumulative discounted reward Rt is defined:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))],$$

[26]

"where *π: S → A* is the policy that determines the action that the agent takes at the present state *st* , and *R(st,π(st))* is the corresponding reward to be obtained immediately by taking the action *at = π(st)".* [26]

"The following figure is an example of MDP. It consists of several states (*s0, s1, s2, s3*) and due to actions (*A0, A1, A2*) a state is transitioned to the next state. Also, a reward value (*R1, R2, R3*) is marked for each state". [13]

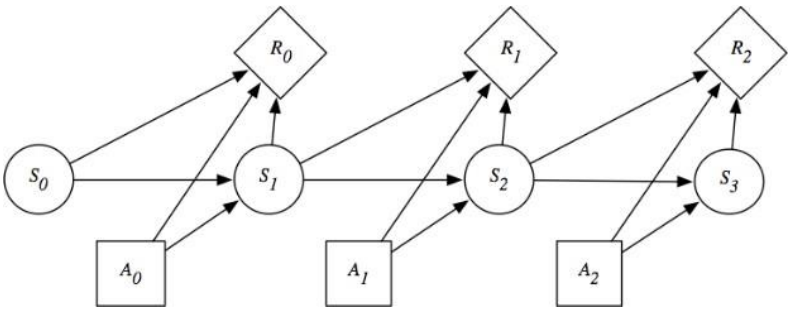

***Figure 3**: MDP scheme [13]*

Components of Reinforcement Learning

"Most of the components in a Reinforcement Learning model are also similar to MDP. But there are some small differences too.

- ***Agent*:** An actor or an object which it is placed inside the environment. This agent receives information from the environment and executes actions on the environment.

- ***Environment***: A place where the agent lives. Generate states and where actions are executed by the agent.

- ***State**:* A specific situation which returns from the environment.

- ***Policy**:* Is the behavior of an agent at a given time. A policy is a methodology which the agent uses to determine the next best action based on the state. It is a map of states and actions which can execute when in those states. The policy is the core component in areinforcement learning scenario. Policy can be simply a small function like a lookup table ora complicated structure like a deep neural network." [13]

"In general, the policy is a mapping from states to a probability distribution over actions":

$$\pi: \mathcal{S} \to p(\mathcal{A} = \mathbf{a} \mid \mathcal{S}).$$

[3]

- ***"Reward**:* Immediate feedback which is given by the environment for the executed action in the last step. The Agent's main goal is to maximize the total reward at a long run. Also, the reward informs the agent whether the action is good or bad. Values of the rewards dependon the scenario.

Depends on reward values, policy should identify whether the action is good or bad and it will try to avoid executing bad actions in the future.

- ***Value:*** Value is like reward. "But value is the long-term feedback. It is the total reward (accumulated) an agent can gain from that specific state to end. According to the experts, values are used to select the best action for each step, not the reward. But without a reward,there is no value". [13]

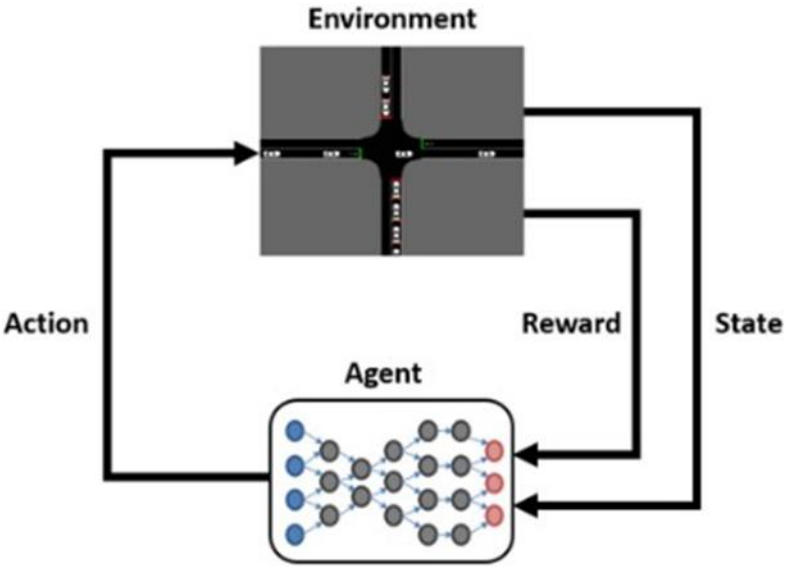

***Figure 4**: Relation between RL components [25]*

Challenges in RL:

This project does not focus on RL but as part of it we are going to do a brief aportation, listing the main challenges that we can find in Reinforcement Learning.
“

- The sample efficiency problem.
- The stability of training.
- The interference problem.
- The exploration problems.
- The meta-learning and representation learning for the generality of reinforcement learning methods across tasks.
- Multi-agent reinforcement learning with other agents as part of the environment.
- Sim-to-real transfer for bridging the gaps between simulated environments and the real world.
- Large-scale reinforcement learning with parallel training frameworks to shorten the wall-clock time for training. “[29]

5.2 DEEP REINFORCEMENT LEARNING

Although the integration of Deep Learning started in the 90s, it has been in the last recent past years when it has achieved its best. It has benefitted from “big data, powerful computation, new algorithmic techniques, mature software packages and architectures, and strong financial support. We have been witnessing the renaissance of reinforcement learning especially, the combination of deep neural networks and reinforcement learning, i.e., deep reinforcement learning (deep RL)”. [32]

“The first, kickstarting the revolution in DRL, was the development of an algorithm that could learn to play a range of Atari 2600 video games at a superhuman level, directly from image pixels” [43]. Providing solutions for the instability of function approximation techniques in RL. [33].

“One of the driving forces behind DRL is the vision of creating systems that are capable of learning how to adapt in the real world” [33]. DRL is here to increase the number of physical tasks that can be automated by learning. [33]

The DRL technique adds deep neural networks “to approximate, given a state, the different Q-values for each action. This allows the model to map between a state and the best possible action without needing to store all possible combinations”: [17]

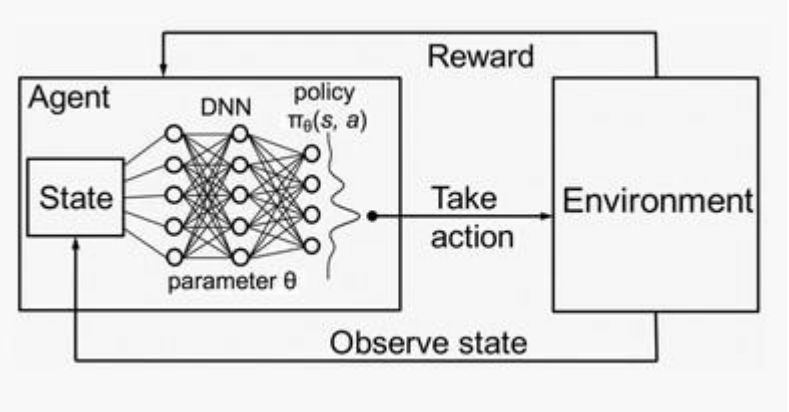

Figure 5: *DRL scheme [17]*

“Deep reinforcement learning is a category of machine learning and artificial intelligence where intelligent machines can learn from their actions like the way humans learn from experience. In this type of machine learning an agent is rewarded based on their actions. Actions that get them to the target outcome are rewarded (reinforced).Through a series of trial and error, a machine keeps learning, making this technology ideal for dynamic environments that keep changing.

The "deep" portion of reinforcement learning refers to multiple (deep) layers of artificial neural networks that replicate the structure of a human brain.” [4]

~~A deep neural network or multilayer perceptron (MLP) consists in mapping a set of input values to output values with a mathematical function formed by composing many simpler functions at each layer. A convolutional neural network (CNN) is a feedforward deep neural network, with convolutional layers, pooling layers and fully connected layers [41].~~

~~CNNs are designed to process data with multiple arrays, e.g., colour image, language, audio spectrogram, and video, benefit from the properties of such signals: local connections, shared weights, pooling and the use of many layers, and are inspired by simple cells and complex cells in visual neuroscience [41].~~

5.2.1 Algorithms

To find the optimal policy π* that achieves the maximum cumulative reward, RL algorithms involve estimating the following value functions. There are two main approaches to solving RL problems: methods based on value functions and methods based on policy search [3]. "There is also a hybrid actor-critic approach that employs both value functions and policy search". [3]

Value based methods

"The value-based algorithms aim to build a value function, which lets us define a policy". [10] Value function methods are based on estimating the value (expected return) of being in each state [34].

"The optimal action-value function with the environment ε could be expressed using Bellman's equation:" [34]

$$Q^*(s,a) = \mathbb{E}_{s' \sim \varepsilon}[r + \gamma \max_{a'} Q^*(s',a')|s,a]$$

[34]

The following list includes the value-based algorithms that we can find and use in DRL:

- Q-learning
- Fitted Q-learning
- Deep Q-networks
- Double DQN
- Dueling network architecture
- Distributional DQN
- Multi-step learning
- Combination of all DQN improvements and variants of DQN

"When we have DQN model, that have been used for solving a very large number of problems, a neural network is used as a non-linear approximator to estimate the action-value function. To train the neural network, TD error is considered as the loss. The loss function is expressed as": [34]

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

[34]

Policy gradient methods

“These methods optimize a performance objective (typically the expected cumulative reward) by finding a good policy (e.g., a neural network parameterized policy) thanks to variants of stochastic gradient ascent with respect to the policy parameters.

The basic idea behind these algorithms is to adjust the parameters θ of the policy in the direction of the performance gradient *∇θJ(πθ).* The fundamental result underlying these algorithms is the policy gradient theorem” [35]:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) \mathrm{d}a \mathrm{d}s \\ &= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\right]\end{aligned}$$

[35]

The following ones are the most used policy gradient methods [10].

- Natural-policy gradient
- Trust region optimization
- Combining policy gradient and Q-learning

Actor-Critic methods

“Actor-critic methods are the natural extension of the idea of reinforcement comparison methods to the full reinforcement learning problem. Typically, the critic is a state-value function. After each action selection, the critic evaluates the new state to determine whether things have gone better or worse than expected. That evaluation is the TD error:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t),$$

[36]

where V is the current value function implemented by the critic. This TD error can be used to evaluate the action just selected, the action a_t taken in state s_t. If the TD error is positive, it suggests that the tendency to select a_t should be strengthened for the future, whereas if the TD error is negative, it suggests the tendency should be weakened. Suppose actions are generated by the Gibbs softmax method” [36].

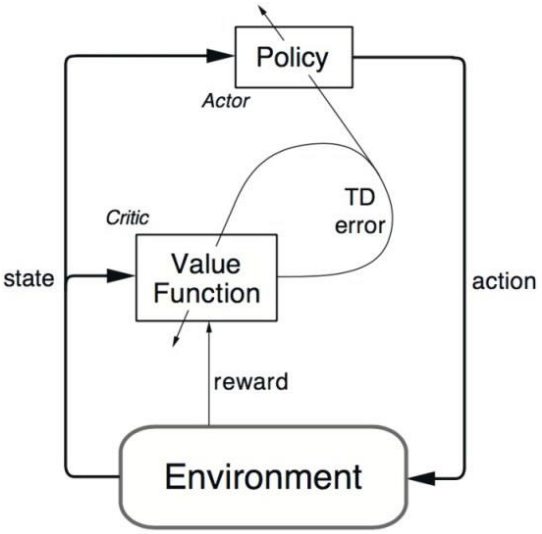

Figure 6: *Actor-Critic Scheme [36]*

5.3 DEEP REINFORCEMENT LEARNING AND TRAFFIC

In this section we are going to make a review of the different software and programs that are being currently used for training environments in traffic control situation. This section will only briefly describe the software that we need to implement, further in this project, they will be explained more in detail.

The general idea that we need to keep in mind is that, when we want to optimize a particular traffic situation, we need to have very clear, what our network is going to be (e.g., a single intersection), how our environment looks like (agent, action space…), which simulator for traffic are we going to use (e.g., SUMO), which framework is going to support our simulator (e.g., FLOW) and which interface are we going to use.

5.3.1 Traffic research

Recent results demonstrate that deep learning and deep reinforcement learning are a promising approach to traffic problems [23]. "A deep learning architecture was used to predict traffic flows, demonstrating success even during highly nonlinear special events; to learn features to represent states involving both space and time" [23]. Our work aims to further this by providing a framework for traffic experiments that uses reinforcement learning for control.

5.3.2 Traffic Simulators

"Traffic microsimulators include Quadstone Paramics, VISSIM, AIMSUN, MATSIM, and SUMO. The first three are closed-source commercial software, whereas the latter two are open-source software. Each of these tools are capable of large-scale traffic microsimulation and can handle variety of policies and control strategies. Each tool offers an Application Programming Interface (API) which permits overriding or extending the default models such as car following, lane changing, route choice, etc.

[We choose to integrate SUMO], an open-source, extensible, microscopic simulator that can simulate large road networks. SUMO discretizes time and progresses the simulation for a user-specified timestep; furthermore, because SUMO is microscopic, individual vehicles are controlled by car following models functions of the vehicle's headway" [23].[37]

"SUMO also includes a Python API called TraCI (TRAffic Control Interface), from which the user can retrieve information about the vehicles' current states and issue precise commands to set the vehicles' velocities, positions, and lanes. Using this interface, we can interface SUMO with RL libraries, read out state information, issue actions, define our own car following models, etc" [23].

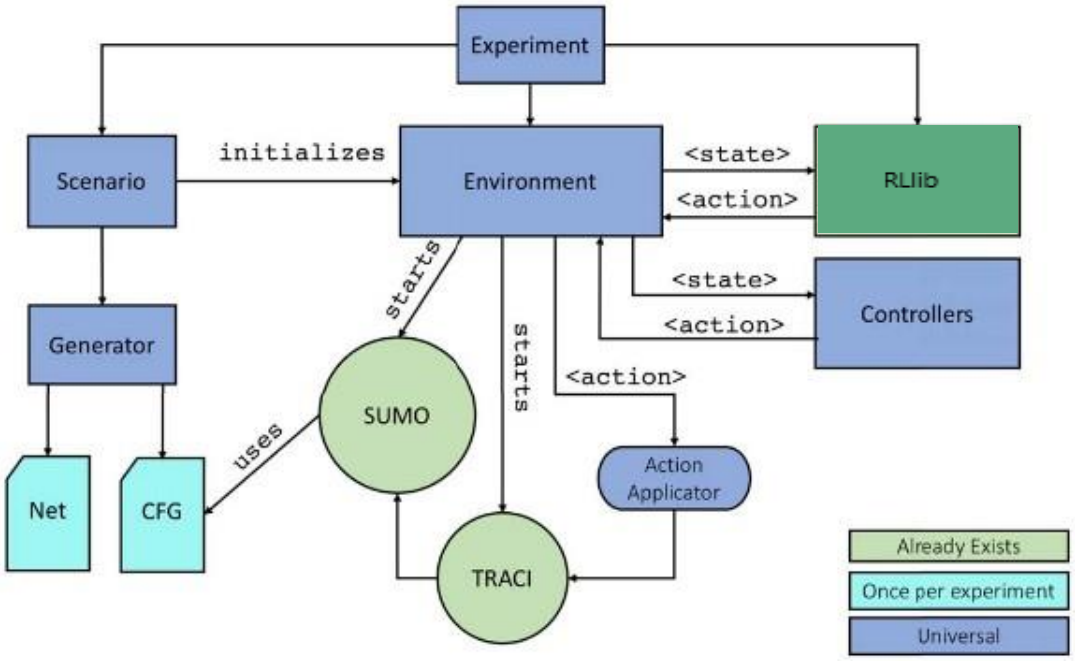

Figure 7: *Simulation Scheme using SUMO [23]*

5.3.3 Architecture of our experiments

The scenario

We need to specify the characteristics of our network. All the attributes of the network must be described in the network file. We will talk later about the scenario we have chosen for our experiments.

The environment

Encodes the MDP, including functions to step through the simulation, retrieve the state, sample and apply actions, compute the reward, and reset the simulation. “The environment is updated at each time step of the simulation and, importantly, stores each vehicle’s state (e.g.,position and velocity).” [23]

We will talk later about the shape of our environment and the characteristic it is going to have for the different situations that we are going to consider.

The algorithm chosen

We will discuss later about the different algorithms that we can use. Once we have defined our network and our environment it is time to train it with a suitable algorithm.

5.3.4 Gym Library

We have also developed our environments in terms of Gym, a toolkit for developing and comparing reinforcement learning algorithms. “It supports teaching agents everything from walking to playing games like pong”. [21]

The Gym API used in reinforcement learning defines a hard boundary between the agent and the environment [18]. “In particular, the agent only interacts with the environment by taking actions and receiving observations. The environment implements a function step that advances the state given an action by the agent; step defines the transition model of the environment” [18].

Our environment must be written in terms of Gym Open AI.

5.3.5 Ray

“Ray provides a simple, universal API for building distributed applications.

Ray accomplishes this mission by:
1. Providing simple primitives for building and running distributed applications.
2. Enabling end users to parallelize single machine code, with little to zero code changes.
3. Including a large ecosystem of applications, libraries, and tools on top of the core Ray to enable complex applications.

Ray Core provides the simple primitives for application building. On top of Ray Core are several libraries for solving problems in machine learning: RLlib” [20].

5.3.6 RLlib

“RLlib is an open-source library for reinforcement learning that offers both high scalability and a unified API for a variety of applications. RLlib natively supports TensorFlow, TensorFlow Eager, and PyTorch, but most of its internals are framework agnostic.

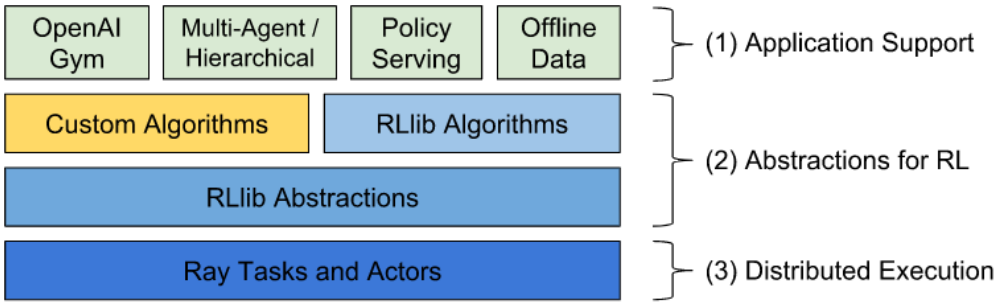

Figure 8: *RLlib: Scalable Reinforcement Learning [20]*

Policies are a core concept in RLlib. Policies are Python classes that define how an agent actsin an environment [20]. Rollout workers query the policy to determine agent actions. In a gym environment, there is a single agent and policy. In vector envs, policy inference is for multiple agents at once, and in mulit-agent, there may be multiple policies, each controlling one or more agents: Ray provides a simple, universal API for building distributed applications”. [20]

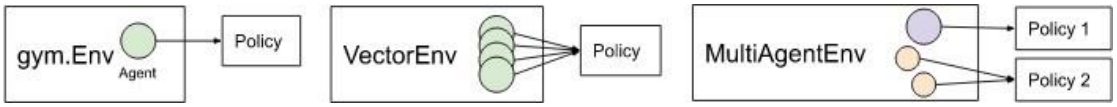

Figure 9*: Policy concept in Ray [20]*

“At a high level, RLlib provides a trainer class which holds a policy for environment inter- action. Through the trainer interface, the policy can be trained, checkpointed, or an action computed” [20].

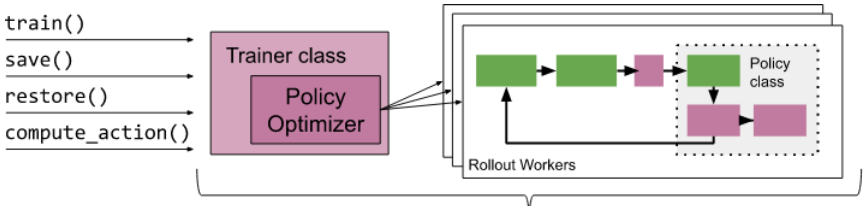

Figure 10: *Rllib Trainer Class [20]*

In the following table we present a list of the available algorithms that we can find on RLlib. If we visit the Ray page, we can see all the parameters configuration that each of them has. The ones used for this project will be described in detail in this paper later.

Algorithm	**Discrete actions**	**Continuous actions**	**Multi-Agent**
A2C, A3C	Yes	Yes	Yes
ARS	Yes	Yes	No
BC	Yes	Yes	Yes
CQL	No	Yes	No
ES	Yes	Yes	No
DPDG, TD3	No	Yes	Yes
APEX-DPDG	No	Yes	Yes
DREAMER	No	Yes	No
DQN, Rainbow	Yes	No	Yes
APEX-DQN	Yes	No	Yes
IMPALA	Yes	Yes	Yes
MAML	No	Yes	No
Marwil	Yes	Yes	Yes
MBMPO	No	Yes	No
PG	Yes	Yes	Yes
PPO, APPO	Yes	No	Yes
R2D2	Yes	Yes	Yes
SAC	Yes	No	Yes
SlateQ	Yes	No	No
LinUCB,LinTS	Yes	No	Yes
AlphaZero	Yes	No	No

Table 1: *RLlib algorithms*

5.3.7 Flow

In order to complete our set up for our simulator, we need a traffic control benchmarking framework that supports SUMO. In this case, we have use FLOW, "it provides a suite of traffic control scenarios (benchmarks), tools for designing custom traffic scenarios, and integration with deep reinforcement learning and traffic micro simulation libraries". [19].

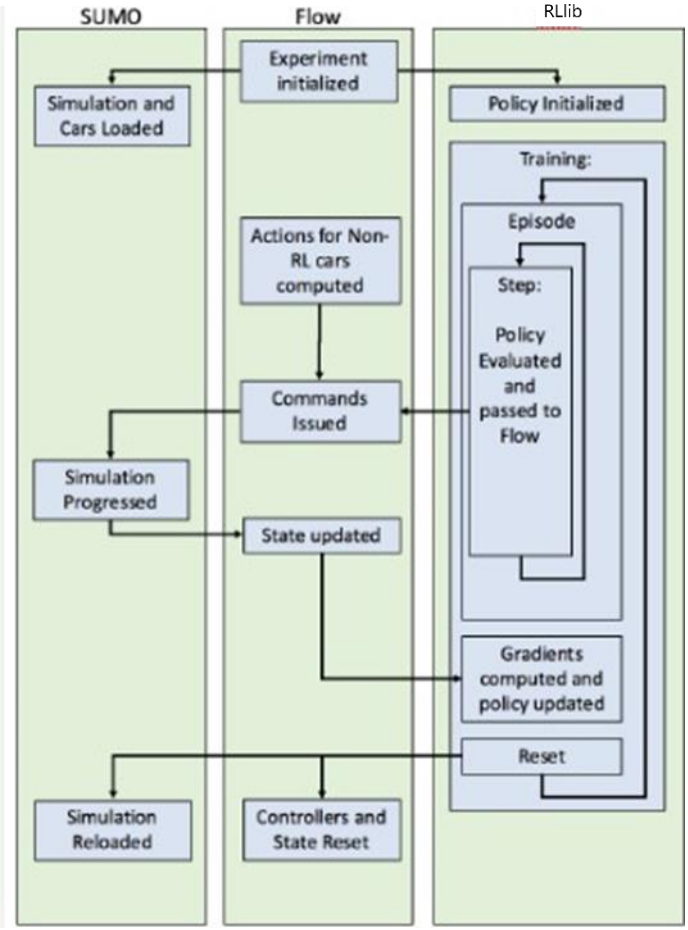

***Figure 11**: Flow + Sumo + RLlib Scheme [19]*

5.3.8 Baselines

There is another framework that can be used with reinforcement learning.

“OpenAI Baselines is a set of high-quality implementations of reinforcement learning algorithms.

These algorithms will make it easier for the research community to replicate, refine, and identify new ideas, and will create good baselines to build research on top. Our DQN implementation and its variants are roughly on par with the scores in published papers. We expect they will be used as a base around which new ideas can be added, and as a tool for comparinga new approach against existing ones.” [8]

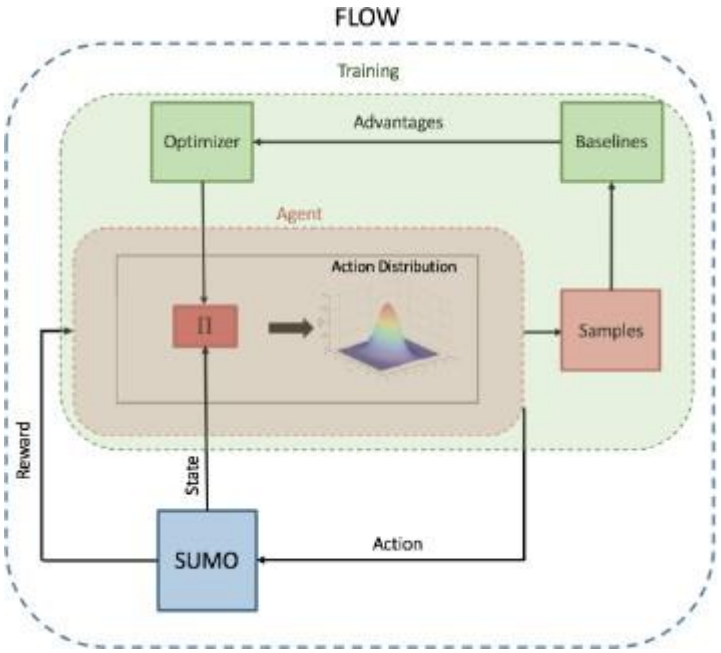

Figure 12: *Flow + SUMO + Baselines Scheme [19]*

5.3.9 Python

In order to develop our system in terms of coding, we are going to use python. Python is going to allow us to integrate everything in a very efficient way. Over the recent years it has demonstrated to be perfect suitable for reinforcement learning applications [44]. With python we are going to beable to write many complex things in very simple ways. Many base documents that we are going to use as at a starting point are included in python.

5.3.10 Anaconda

“Anaconda is created by Continuum Analytics, and it is a Python distribution that comes preinstalled with lots of useful python libraries for data science.

Anaconda is popular because it brings many of the tools used in data science and machine learning with just one install, so it’s great for having short and simple setup.

Like Virtualenv, Anaconda also uses the concept of creating environments so as to isolate different libraries and versions. Anaconda also introduces its own package manager, called conda, from where you can install libraries.

Additionally, Anaconda still has the useful interaction with pip that allows you to install any additional libraries which are not available in the Anaconda package manager.” [45] [37]

6. SET UP FOR OUR EXPERIMENTS

6.1 INTRODUCTION

Now that we have presented the objective of this article and that we have done a research into the current situation of reinforcement learning, deep reinforcement learning and the possibilities that we have in order to train our experiments, we are going to explain how we have developed our simulator and the set up that has been done before the obtaining our results.

In this project we have carried out different experiments that have been compared between them after.

In this article we are going to work with a small traffic system. We are going to simulate a single intersection.

We are going to introduce during this section the state space, action space and reward for all the cases that we are going to consider. We will also explain the algorithm chosen for the control of the traffic environment.

As it has been discussed before, we are going to present our network, our environment and the framework used to train them.

6.2 NETWORK

For completing this project, we have chosen to train our environment using a single intersection as our network.

Traffic signal control in road networks, is an important practical problem due to substantially increasing delay and fuel cost caused by traffic congestion.

We consider a signalized intersection with a set of entry and exit approaches, in which each approach has a fixed length and a set of lanes.

The following figure shows the popular phase schemes, all of them are compatible streams.

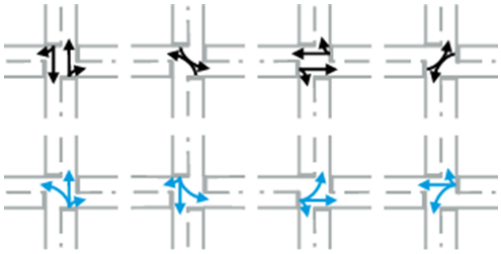

Figure 13: *Phase Scheme (1) [15]*

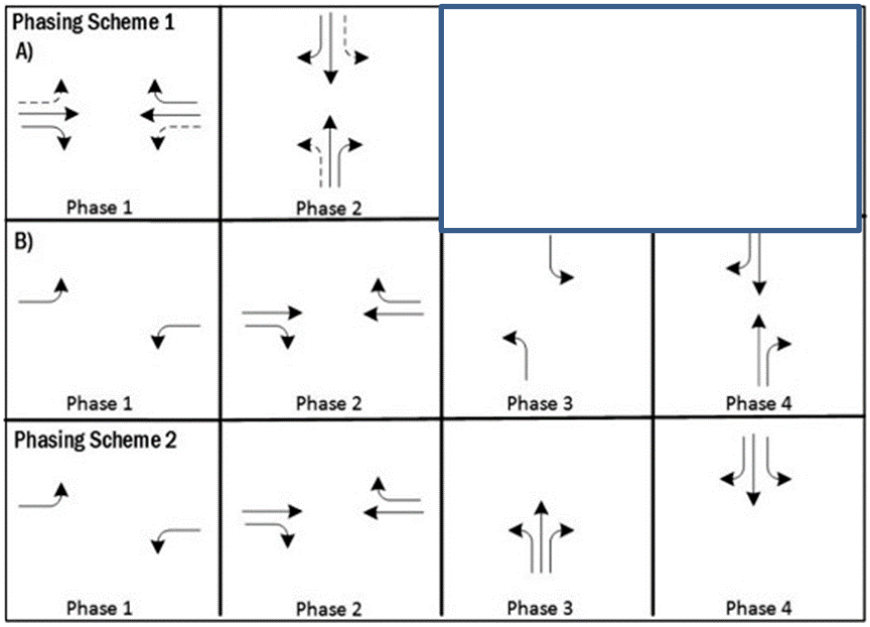

Figure 14: *Phase Scheme (2) [2]*

In our experiments, our network as it has been said before, has three roads for each lane.

Each approaching part of the intersection is managed by a traffic light, and each traffic light has three phases (green, ambar and red). We have coded our network in terms of flow, using the core params of it for the initial configuration and for the params of the traffic lights. So, once we have installed flow, we can make use of this params just by importing them from the appropriate folder.

It needs to be in terms of flow because as we have said before, it is the framework that supports SUMO, what we are using. If we start from the already defined class in flow *TrafficLightGridNetwork*, we can then create our network with the specific attributes that we want to use. All the connections that encode how the network is made are also already defined. The important thing about the code used to build the network is the python class that encodes properly the structure of the network, and the specific parameters that we want to use. In our case we have decided to work with inflows, so this is also described with all its attributes. The value of the parameters used in the network are described in Table 2.

Basically, what we are going to define inside the *TrafficLightGridNetwork* class is the initialization of the network, defining the routing table, the inflows at the outer edges of the grid, the vehicles and its params, the specify routes and connections. Also, we are going to code everything about the inflows, the veh_type, the depart_lane, the depart_speed, the step_lenght.

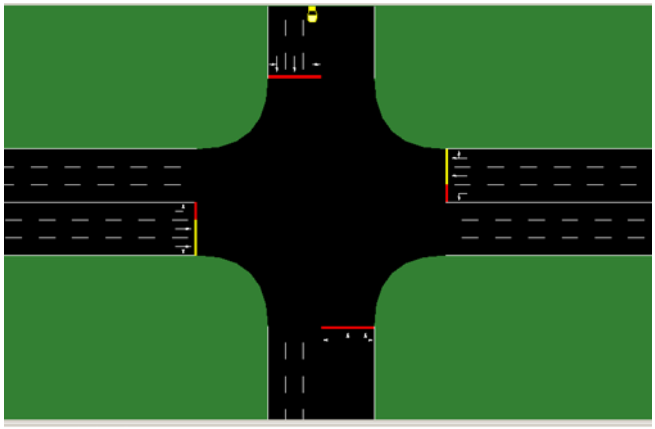

Figure 15: *Single intersection in Sumo*

6.3 ENVIRONMENT

In this section we are going to describe how the environment has been coded. This part is very important because here we need to define the state space, the action space and the reward. It is very important to have in mind, that we have to code our environment in terms of flow and gym so it can be simulated with our traffic micro simulator SUMO and it can be trained with a RLlib algorithm.

First of all, and as it is described in the task proposed for this project, we need to compare the obtained results of only traffic signal control with and without V2I communication and also these results need to be compared whit the proper system developed that jointly optimizes traffic signal control and speed advisory.
For this reason, we have to set up four different classes as our environments that in general terms are going to be equal but when we will talk about the action and space state, they are going to be different.

To sum up, this is how the project is going to be structured:

Only traffic signal control:
-With V2I
-Without V2I

Joined optimization of traffic signaling control and speed advisory:
-With V2I
-Without V2I

It is important to add, that talking about the code developed other python libraries (e.g numpy) have been used in order to complete correctly everything.

In the end, as at it has been explained with the objective of the project, we want to improve the traffic state situation providing the traffic infrastructure with information of the vehicles. The other goal is to allow the traffic infrastructure to provide the drivers information aboutthe speed they should have in order to decrease the traffic congestion.

In terms of coding, as we have said, we are going to code two different classes: *SolitaryTrafficLightGridEnv* and *CommuicativeTrafficLightGridEnv.* These two classes, part from *BaseTrafficLightGridEnv* that parts from the general Env class of flow. We need to import this before start coding the new ones. The first one, is going to be the one for no V2I and the other one is going to be the one with V2I. In the last experiments the action_space and the rl method are going to be modified. This changes are common to both classes so they will be applied to the common part of the code, set in the class *BaseTrafficLightGridEnv.*

Before starting to define the classes and their variants, we are going to define the parts that are common in terms of coding. We need to import first of all, everything that we are going to use from gym.spaces. After, we need to import everything we need from flow. We are also going to import sumolib.

Once everything is imported, we are going to define the *TrafficPhaseOptions*, the additional env params, the additional communicative env params and the solitary env params. Then, we can start with our classes.

Then, we define the *BaseTrafficLightGridEnv*, our classes are going to be based on this one, here we defined general items. We need to initialize everything, define the action space here

depending which situation we are considering, we also need to define the apply_rl_actions, again, depending of which situation we are considering. Another point to define here is the compute_reward, the step and the restart_simulation. Before the particular classes, we defined some aspects that are copied from *GreenWaveEnv*. We define how to record the velocities and edges at each time step, how to know the smallest distance from the current vehicle´s position to any of the intersections, how to know the distance from the vehicle´s current position to the position of the node it is heading toward, how to know the ids of the vehicles in the network by their distance to the intersection, how to convert the string edge to a number and how to know the veh_id of the k closest vehicles to an intersection for each edge.

6.3.1 State Space

The state space of an RL agent defines the knowledge on which it can base its decisions [5]. For the thesis we have chosen two different state spaces. One considering the V2I communication and another one without V2I communication.

No V2I communication

In this case, our agent has no information about all the vehicles in the road. States are limited to internal information of the traffic lights. In order to create a state space when no V2I is existing that allows us to simulate both only traffic signaling control and the joined optimization of traffic signal control and speed advisory. “The agent features the time since the last phase change and a trace for every phase, which increases while the respective phase is activated and slowly decays while it is not”. [5]

When being in the situation of not having any information related to the vehicles, the drivers and the state of the network, the observation space used for these situations is quite smaller. The information that we have here is limited and only regarding the traffic lights situation.

We are going to have the following information available:

- Time passed since phase was last change.
- Traces of all phases
- Current phase
- Current period

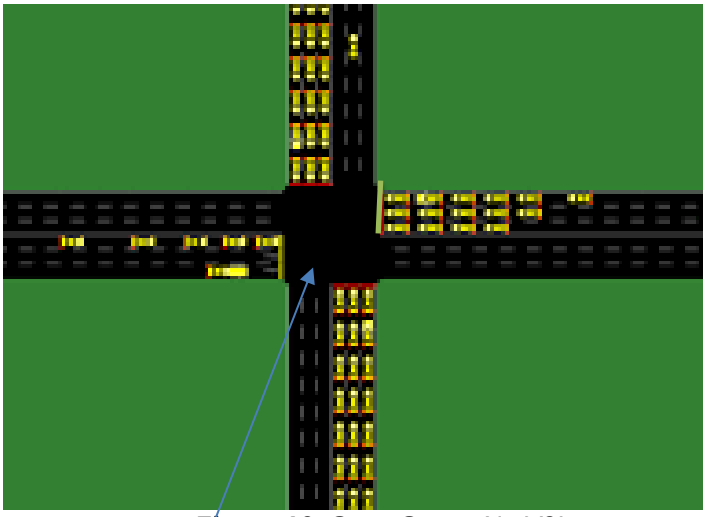

Figure 16: *State Space No V2I*

In this case the agent is only allowed to know what is the current phase and the time that has passed since the last phase. We do not know anything about the vehicles involved.

We cannot forget the compatible streams that we have in the road in order to have non-conflicting situations. The defined class for this case is called *SolitaryTrafficLightGridEnv* .

V2I communication

The agent receives detailed information about the state of the traffic network. This results in our agent having more detailed information about everything happening around him.

This leads us to a more complex system but a better one. In these systems where much information about the vehicle can be known, for example, we can note the distance to the intersection of the vehicles approaching the intersection. This leads as, as I have said, to better and more complex systems that are going to be able to adapt the signal control in order to have better situations for all the drivers. This is how the congestion problem is going to be solved and how everything that is generated from this congestion is going to get better. Again, the state space considering V2I communication is going to be the same for the only traffic control signal case and for the develop of the joined optimization of traffic signal control and speed advisory.

In this case, the amount of information that we have is considerably bigger, so our agent is going to be in an environment where he is going to have many information about what surrounds him.

In this case, the information that the agent is going to have is the following:

- Time passed since phase was last change.
- Traces of all phases
- Current phase
- Current period
- Velocities of observed phases
- Distance to intersection of observed vehicles
- Edges of observed vehicles
- Lanes of observed vehicles
- Vehicle density for all edges

All this considered we can say that the more information the agent has, the better results we are going to obtained in relation to the reward that we have chosen but also the more resources are going to be needed.

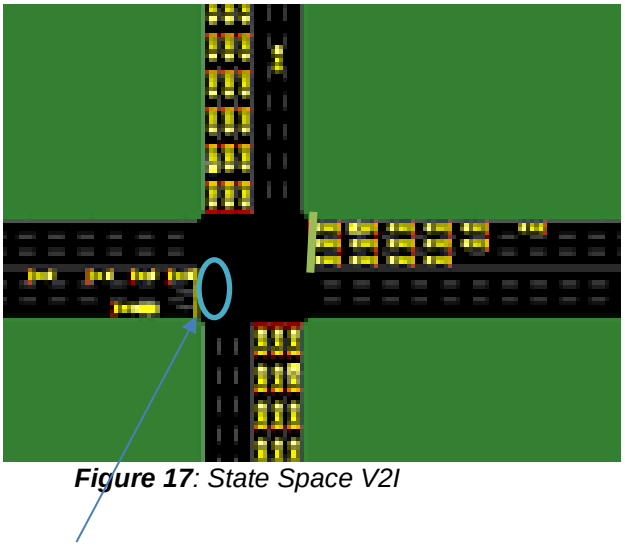

***Figure 17**: State Space V2I*

In this case, for example, the agent could know the velocity of the cars approaching the lane, the distance and of course the current phase and the time passed since the last phase.

With all this information the agent can make decisions in function of many parameters which is going to be very helpful. As we will see in the results, this capacity of having much information is going to give us better results.

6.3.2 Action Space

According to what we have considered theoretically, once the agent has observed the state of the environment, it must choose one action from the set of all available actions [16]. For the experiments that we have done, we need to distinguish again between two different action spaces: the one that only considers the traffic light control and the other one that jointlyoptimizes the traffic signal control and gives advice about what speed vehicles should have.

A traffic light needs to choose an appropriate action to well guide vehicles at the intersection based on the current traffic state. And also, would have to advise of the adequate speed when talking about the joined system.

Only traffic signaling control

"The agent decides every time step which phase to show and the display duration. If the current phase has already been shown longer than a chosen duration (at any decision time step while a given phase is displayed), then the newly chosen (different) phase will be displayed next" [5].

In this case the green phase duration is limited to a fixed time. All available phase options consist of only compatible streams, making the agent's actions inherently safe. As it is set, afterthe green phase has been shown, the traffic light needs to go through an amber phase beforegoing to red. "All available phase options consist of only compatible streams, making the agent's actions inherently safe" [5].

This is the first approach of the thesis; we have read many articles in relation to this part but we use as starting point one which results proved good behavior not only in single intersection but in more complex systems. From our starting point we can continue developing our new system.

Joined optimization of traffic light signaling and speed advisory

Here, we are going a little bit further. The agent in this case, is going to choose what phase to show and the duration of the phase, always accomplish with what is set for the minimum phase duration and the maximum phase duration. In addition, now, the agent is also going to show to all the vehicles approaching the road what speed to have. The phase and conditions of the traffic lights are going to be the same as in the other case considered.

Even though we could have pretty good options here all of them should be carefully evaluated as long as it does not exist as many papers talking about speed advisory as talking about traffic control.

The simplest way of implementing speed advisory would be let the agent decide a max velocity for every approaching lane, then it would collect all the vehicles of that lane and then it would set each of their max-speed to that velocity.

Other options where considered. For example, we could have an agent that could suggest every vehicle to accelerate, decelerate or maintain it speed. Of course, for this case we would have to have information about the vehicles approaching. For implementing this, our action space needs to have a fix number of vehicles and this is not our case. Again, if we had a fixed number of cars space, we could implement an agent that could suggest a particular speed for each vehicle approaching the lane.

As we have said before, the action space is the same no matter if we have V2I or not V2I for both defined classes.

6.3.3 Reward

We have previously defined what the reward means inside a reinforcement learning context. In terms of traffic control, we have a wide range of objectives that we want to manage, including safety, efficiency, environmental sustainability, comfort, and fairness. Rewards can be based in one only parameter or in a multiple-parameters batch. After have been reading many papers, we have come to the conclusion that in general terms, velocity of the cars is the more common reward for evaluating the efficiency of a network. In no V2I communication cases, the agent has no information about the velocity of thevehicles, so it has to be provided by doing previous simulations or by an external simulator.

Finally, we are going to explain how all that has been explained before in this section, has been code. It is very important that in order to have everything prepare for training, everythingneeds to be written in terms of flow and gym. As it has been also explained, we are coding inpython and we are using anaconda to set up the env.

As we have different situations to train different codes are needed. This is out of the scope of the thesis, but it is going to be briefly described. For each kind of environment, for example, no V2I only traffic control and V2I traffic control, two different classes in python have been created. Nevertheless, both use as a base a pre-made python file for traffic grid environments. We also have some parameters like the traffic light phases that are defined outside of all classes because they are the same independently of what situation we are in.

6.4 PPO

For training the environments and obtaining the results, we have decided to use one of the RLlib algorithms. In this section, we are going to explain why the reasons for choosing the algorithm, the parameters of the algorithm, and so modifications that can be made in order to speed up the stabilization of the system.

This algorithm is included inside the policy gradient methods that directly search for an optimal policy. "Policy gradient is the most used approach of policy based methods, which computes an estimator of the agent's policy gradient by a stochastic gradient ascent algorithm" [27].

"Proximal Policy Optimization (PPO), which perform comparably or better than state-of-the-art approaches while being much simpler to implement and tune. PPO has become the default reinforcement learning algorithm at OpenAI because of its ease of use and good performance.

With supervised learning, we can easily implement the cost function, run gradient descent on it, and be very confident that we'll get excellent results with relatively little hyperparameter tuning.

PPO strikes a balance between ease of implementation, sample complexity, and ease of tuning, trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small." [40]

"PPO's clipped objective supports multiple SGD passes over the same batch of experiences. RLlib's multi-GPU optimizer pins that data in GPU memory to avoid unnecessary transfers from host memory, substantially improving performance over a naive implementation. PPO scales out using multiple workers for experience collection, and also to multiple GPUs for SGD". [20]

Figure 18: *PPO architecture [20]*

We also present here the PPO-specific configurations:

“

```
# Adds the following updates to the (base) `Trainer` config in
# rllib/agents/trainer.py (`COMMON_CONFIG` dict).
DEFAULT_CONFIG = with_common_config({
    # Should use a critic as a baseline (otherwise don't use value baseline;
    # required for using GAE).
    "use_critic": True,
    # If true, use the Generalized Advantage Estimator (GAE)
    # with a value function, see https://arxiv.org/pdf/1506.02438.pdf.
    "use_gae": True,
    # The GAE (lambda) parameter.
    "lambda": 1.0,
    # Initial coefficient for KL divergence.
    "kl_coeff": 0.2,
    # Size of batches collected from each worker.
    "rollout_fragment_length": 200,
    # Number of timesteps collected for each SGD round. This defines the size
    # of each SGD epoch.
    "train_batch_size": 4000,
    # Total SGD batch size across all devices for SGD. This defines the
    # minibatch size within each epoch.
    "sgd_minibatch_size": 128,
    # Whether to shuffle sequences in the batch when training (recommended).
    "shuffle_sequences": True,
    # Number of SGD iterations in each outer loop (i.e., number of epochs to
    # execute per train batch).
    "num_sgd_iter": 30,
    # Stepsize of SGD.
    "lr": 5e-5,
    # Learning rate schedule.
    "lr_schedule": None,
    # Coefficient of the value function loss. IMPORTANT: you must tune this if
    # you set vf_share_layers=True inside your model's config.
    "vf_loss_coeff": 1.0,
    "model": {
        # Share layers for value function. If you set this to True, it's
        # important to tune vf_loss_coeff.
        "vf_share_layers": False,
    },
    # Coefficient of the entropy regularizer.
```

```
    "entropy_coeff": 0.0,
    # Decay schedule for the entropy regularizer.
    "entropy_coeff_schedule": None,
    # PPO clip parameter.
    "clip_param": 0.3,
    # Clip param for the value function. Note that this is sensitive to the
    # scale of the rewards. If your expected V is large, increase this.
    "vf_clip_param": 10.0,
    # If specified, clip the global norm of gradients by this amount.
    "grad_clip": None,
    # Target value for KL divergence.
    "kl_target": 0.01,
    # Whether to rollout "complete_episodes" or "truncate_episodes".
    "batch_mode": "truncate_episodes",
    # Which observation filter to apply to the observation.
    "observation_filter": "NoFilter",

    # Deprecated keys:
    # Share layers for value function. If you set this to True, it's important
    # to tune vf_loss_coeff.
    # Use config.model.vf_share_layers instead.
    "vf_share_layers": DEPRECATED_VALUE,
})" [20]
```

The important point of this section is to notice that with one algorithm we can simulate all the scenarios that we are considering.

7. IMPROVEMENTS

Once that we have reached this point, I would like to add to the project some improvements that could have been considered. In case of the jointly optimization of traffic signal control and speed advisory, a multi-agent system could have been developed. This system would have one agent as the traffic lights for traffic control and the other agent would be the own cars incorporating information from the current traffic state in order to tell the driver which speed would be better for each case. However multi-agent presents some complex problems.

"The first challenge is the representation problem. Specifically, the challenge is in defining the problem in such a way that an arbitrary number of agents can be represented without changing the architecture of the deep Q-Network. To solve this problem, we make a number of simplifying assumptions: (i) two-dimensional representation of the environment, (ii) discrete time and space, and (iii) two types of agents. Because we limit ourselves to two agent types, we will refer to the two agent types as allies and opponents (assuming competing agents). These assumptions allow us to represent the global system state as an image-like tensor, with eachchannel of the image containing agent and environment specific information" [1].

"The second challenge is multi-agent training. When multiple agents are interacting in an environment, their actions may directly impact the actions of other agents. To that end, agents must be able to reason about one another in order to act intelligently. In order to incorporate multi-agent training, we train one agent at a time, and keep the policies of all the other agentsfixed during this period". [1]

We could have also chosen an algorithm directly design by us for speed up the converging of the system. Also, designing an algorithm would have let us use more specific parameters that could adjust better to our work.

8. INSTALLATION

In order to get to the results and being able to start training our environments, all the software requirements that have been described along this project have needed to be installed. In this section, we need to make clear that this year has been very complicated, due to a global pandemic and going to the university has not been possible. So, with this considered, I have worked from my own machine using a remote server from the University via VPN connection.

All these problems considered, a brief description of the installation process is going to be made.

Once we are connected to VPN of the university, we can connect to the server via *ssh* specifying the IP direction of where we want to connect.

After this has been made, we can start installing everything in the remote server. First of all we need to install *conda* in order to be able after, to set up the flow environment that we need for our simulations. We also need to install python as long as this is going to be the programming path we have chosen to code our network, our environment and the training algorithm. It is very importat not just intalling pyhton but all the libraries that we are going to need for codingour project (Tensorflow, gym...).

Next step it is to start with the installation of Flow and SUMO. How to install and create the flow environment via the terminal of Linux is explained in the official Flow page [19]. In order to being able to watch the simulations in the SUMO simulator, we need to have installed sumo-gui, the graphical user interface. This cannot be done via remote server, due to the root privileges but there are other ways of installing gui apps. This is important and very visual but as long as we are talking in terms of comparing results, we can work without the graphical interface.

Last but not less, we need to start Ray that is quite simple to install but not than simple to understand. Training an environment with Rllib is quite easy but first of all we need to register the environment as a gym one in order to be able to use the algorithm we want with our environment.

The most important thing about Ray that we need to consider is that, when we call the ray.init() we are initializing the Ray Client, an API that connects a python script to a Ray cluster.

9. SIMULATIONS

At this point, we have made a theorical revision of DRL for traffic signal control and we have also explained, how we have set up everything for our simulations. We need to remember that we are extending an already existing system, so we also have used the code of that project as a starting point. [5]

Another interesting point that is to going to be added here is that, for developing all the simulations and understand all the results, all the tutorials from Flow and SUMO have been followed and read.

Also, the tutorial from Ray and Rllib were needed to be read in order to understand how to install everything and how it works.

Once everything that has previously been done is explained, we start to explain properly how the simulations work and the results that we have obtained.

All the python folders that we are going to need as well as the file where the proper code for the training is written have been added to a jupyter notebook. We have decided with jupyter because of my previous experience with it.

For starting our simulations, we need to open a jupyter file. The extension of this file is. *ypynb*. Of course we need to start the notebook in the flow environment in order to be able to use the flow parameters. Our training code is going to be in a folder called *lib.* In this folder we are going to have our network and environment code and the controllers code.

Let's take a moment to discuss about controllers. This file of python code defines the process of movement of human-driven vehicles within a network. We need to use them because we need our vehicles to know how to move inside the network. As long as in this study we are not going to control the vehicles, but the traffic lights, we use the controllers defined for the vehicles in Flow.

Another point that we need to include in this code is that the traffic phase options, they need to be imported from where they are written. Also, the class defining the particular case of our environment that we want to simulate needs to be imported.

The net params, the Flow params and the sim params need also to be imported from the core folder of Flow and we will talk about them later.

Once everything that needs to be imported has been brought, we start writing.
We need to define the aditional env_params here. This includes some attributes such as switch time, the min phase duration, the max phase duration…In the following table we include the params that have been used and its value.

Params	Value
Switch time amber	5.0 (s)
Min phase time	5.0 (s)
Switch time red	7.0 (s)
Reward function	Velocity
Num observed	10
Discrete	True
Tl_type	Controlled
Phases	Traphic_Phase_Options

Table 3: *Additional_env_params*

About this params we can add that, in the environment code different possibilities in terms of reward were written so we need to choose and, in this case, we are chosen the average velocity of all vehicles.

We also need to define here again some of the attributes that define the structure of the network. They are shown in the following table.

Params	Value
Short length	300
Inner length	300
Long length	300
Row num	1
Col num	1
Cars top	0
Cars bot	0
Cars left	0
Cars right	0

Table 4: *Grid Array Params*

In the table shown above, we can see the length of the lanes that we have, remember that we have three lanes in each road and two roads in each arterial approaching the lane. We also define the number of vehicles that we want when starting the simulation in the top, in the bottom, in the left and in the right.

As we have also explained before, we are going to work with inflows, this is optional but as long as in the code we are starting from they are being used we will also include them here. This params are used to specify the characteristics of the vehicles that are going to enter randomly in the system. We define the specific speed of departure, the type of vehicle, that in this case is going to be a normal vehicle driven by a person and the lane from where they are going to start, in this case, random.

Params	Value
Vehicle type	Human
Vehicles per hour	800
Depard Speed	5
Row Col Ratio	1
No turns Ratio	1
Depart Lane	Random
Step lenght	None

Table 5: *Inflows params*

Last list of parameters that we need to particularly define are the additional net params. This are attributes of the network that we add to the fix ones in the own network code.

Params	Value
Grid array	Grid array
Inflows	Inflows
Horizontal lanes	3
Vertical lanes	3
Speed limit	20

Table 6: *Additonal net params*

We define here the attributes defined before for the network, and other important params such as the number of lanes and the speed limit.

At this point we have already in the code all the network and environment parts of code that we need for the simulation, and we need to start coding the ray part.

First of all, as we have said our environment needs to be set up in terms of gym and once, we have accomplished that, we need to register our environment. When we install ray, we can find a folder called tune where all the registry environments are defined. We can see there the classic ones used in DRL (e.g cartpole). We will import the part of the file where the environments are registered in order to after select the one that we need. Important to remember, we want to do different simulations, so we need to register each code for each simulation separately.

Now that we have everything let's put it together.

We need to call the *def_make()* because we need to add the env_params, the sumo_params and the sim_params that we have defined to the environment that we have registered. Remember that we have called the standard and the base of the env_params, the sumo_params and the sim_params before so now we only have to add the extra ones that we have defined before.

Now all the conditions are ready and we can call ray.

We need to import ray and the algorithm that we want to use from the rllib agent's folder. Right after we can call the *ray.init().* We need to call this in order to connect to a free ray cluster. A Ray cluster consists of a head node and a set of worker nodes. The head node needs to be started first, and the worker nodes are given the address of the head node to form the cluster:

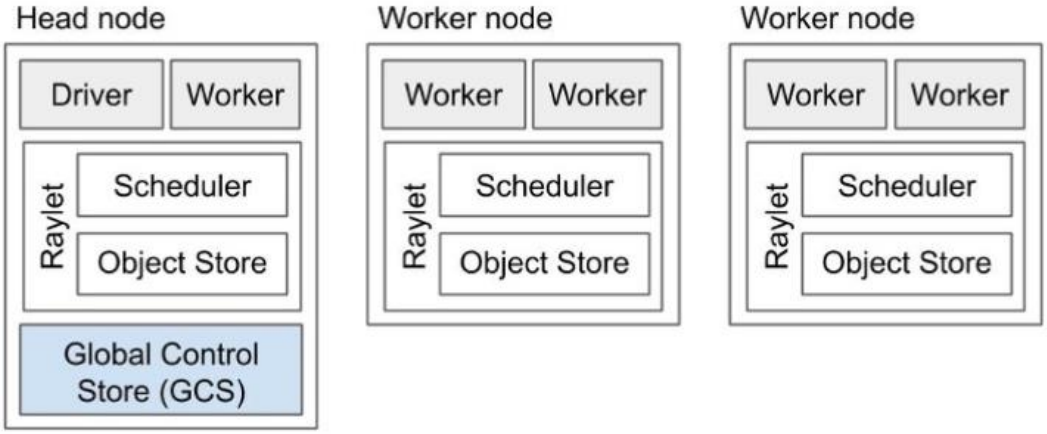

Figure 19: *Ray Cluster*

So, we are almost ready with the simulation.
Now we need to set the configuration of the algorithms. In all the cases, we are going to work with the Default Configuration, presaved by rllib. With the modification of the params we could adjust to have a quicker system or for it to get stabled sooner.

Once the configuration is set, we define the agent with the Trainer of the algorithm that we are going to use, the set configuration and the environment that we have registered with all the params included.

Now we are ready to start training our environment.

10. RESULTS

First of all, and before showing the results, we are going to see which files are going to be generated and how we can work with them in order to compare the results obtained in the different simulations.

As it is explained in the flow tutorials, when we run a simulation involving flow, we are going to obtain three types of files. Each of this group will allow us to see a piece of information about the training process.

*"Reward plotting**:***
We need to be able to graphically see the reward function in order to evaluate the effectiveness of our system.

Policy replay:
Flow includes tools for visualizing trained policies using SUMO's GUI. This enables more granular analysis of policies beyond their accrued reward, which in turn allows users to tweak actions, observations and rewards in order to produce some desired behavior. The visualizers also generate plots of observations and a plot of the reward function over the course of the rollout.

Data collection and analysis:

Any Flow experiment can output its simulation data to a CSV file, emission.csv, containing the contents of SUMO's built-in emission.xml files. This file contains various data such as the speed, position, time, fuel consumption and many other metrics for every vehicle in the network and at each time step of the simulation. Once you have generated the emission.csv file, you can open it and read the data it contains using Python's csv library (or using Excel).This file is going to include a list of numerical results that we can plot.

[In case of using Sumo with Rllib agents, which is our case, we are going to obtain the following information in each group. These files are going to be directly located if we do not change the directory, in a folder call ray_results.]

Reward ploting:

RLlib supports reward visualization over the period of the training using the tensorboard command.

Policy replay:

The tool to replay a policy trained using RLlib is located at flow/visualize/visualizer_rllib.py. It takes as argument, first the path to the experiment results (by default located within ~/ray_results), and secondly the number of the checkpoint you wish to visualize (which correspond to the folder checkpoint_<number> inside the experiment results directory).

Data collection and analysis:

If you need to generate simulation data after the training, you can run a policy replay as mentioned above, and add the --gen-emission parameter." [19]

With all this information we can make tables and graphics comparing the different situation we want to simulate.

In the previous work, where the starting point system was created, they let the spawn rates of the Poisson process to be equal to each, so the average arrival rates for all four approaches were equal. This means that all vehicles approaching the lane where equally likely to take a left turn, go straight or take a right turn at the intersection.

The conclusion obtained in relation to the V2I communication situations and the no V2I communication were that the case where the agent has more information, the average velocity of the vehicles is increased and also the flow rates are increased. In their case they also try a composite reward obtaining that the CO2 emissions were reduced as well as the average wait and trip times and the driver stress metric.

Only traffic signal control:

First, we are going to show how the system becomes stable after training the agent with PPO:

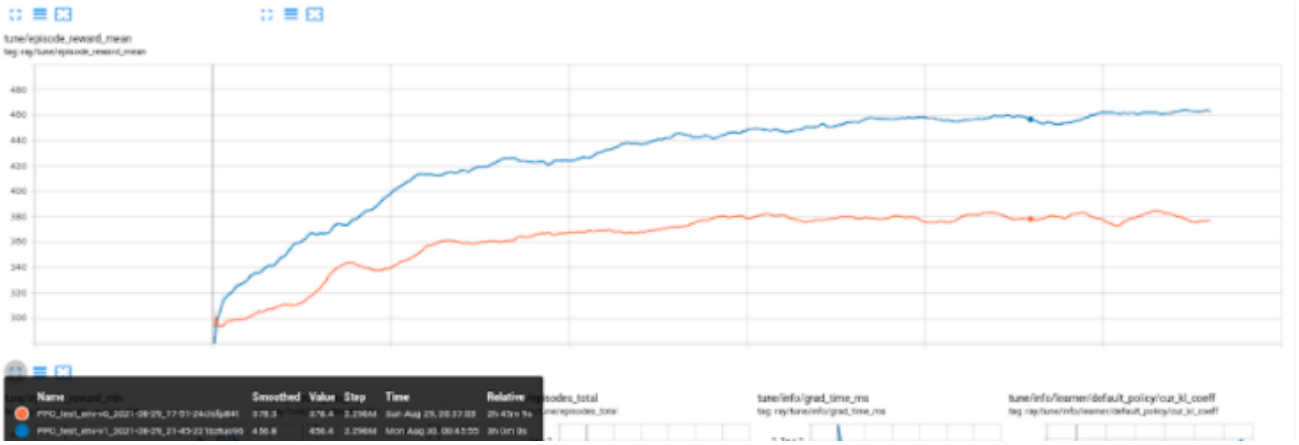

Figure 20: *V2I vs NO V2I trained with PPO only traffic signal control*

With this first graphics we can see that, when even only considering signal control, the mean reward is considerably bigger in case of having V2I. We can also appreciate that using PPO in this case is useful, using the default parameters and a little bit of time our system converges.

After our systems have been trained, we need to evaluate them. For that we make use again of ray, we use the line *compute.action* for, as we have said, to compute the actions from a trained agent. This method preprocesses and filters the observation before passing it to the agent policy [20]. A piece of code where the corresponding environment is registered and the trained simulation is restored in order to obtain the velocities in a particular situation that we defined using, as we have said, the trained environments, has been developed. As a result of that piece of code, we have obtained the velocities of the vehicles when using the trained data in a particular situation, the same one for both cases, and with that we have made some plots and conclusions.

We have obtained the mean velocities of the vehicles in the system in a predefined number of runs. Then, we have made a boxplot comparing both situations, and this is what we have obtained:

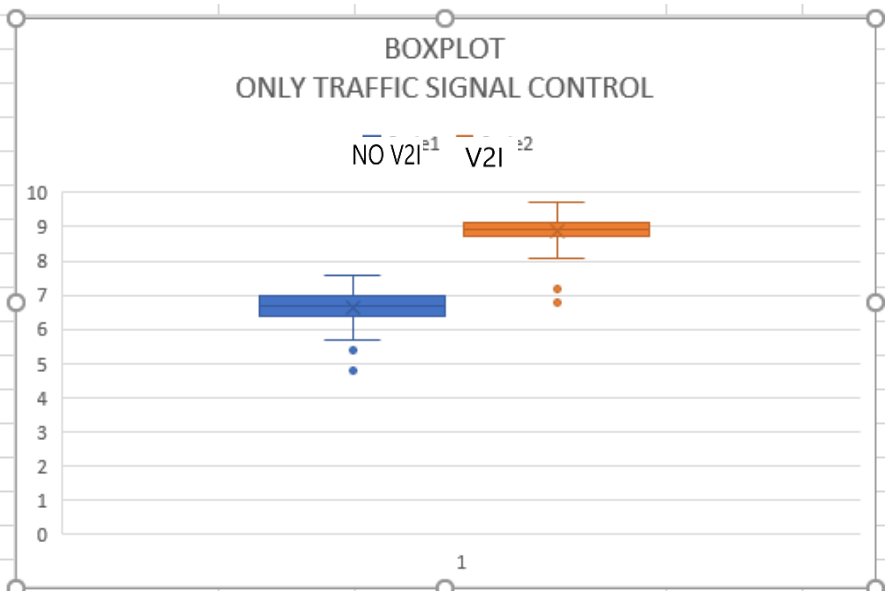

Figure 21: *Boxplot: Only signal control V2I VS NO V2I*

The velocity is in m/s.

We need to briefly explain why the numbers regarding the velocities are so low. If we go to the code, we see that it has been set that the reward is average_speed/max_speed. We have set to 20 m/s our velocity at the max speed for every lane. Again, if we go to the code, we can see that our training episodes have 1000 steps. If we do the following:

(The return in tensor board (approx. 380 and 420)/1000) *20

We obtain approx. after that formula the numbers that we have in the boxplot.

As we can see, the mean velocity is lower in case of not having Vehicle to Infrastructure communication. This is pointing that having V2I communication is going to allow us to have better scenarios in real life. If the velocity is higher and of course within the limits, we will have less stressed drivers and people spending less time in traffic with will not only reduce the CO2 emissions but that will mitigate the traffic congestion and all the effects associated that have been considered along this project.

Our first simulations have accomplished the fixed goal showing us that allowing the infrastructure knowing some parameters about the vehicles have some advantages in terms of reducing traffic problems. This, when we are speaking about only traffic signal control.

Once the only traffic signal control part has been analyzed, we are going to present the results for the system including the speed advisory part.

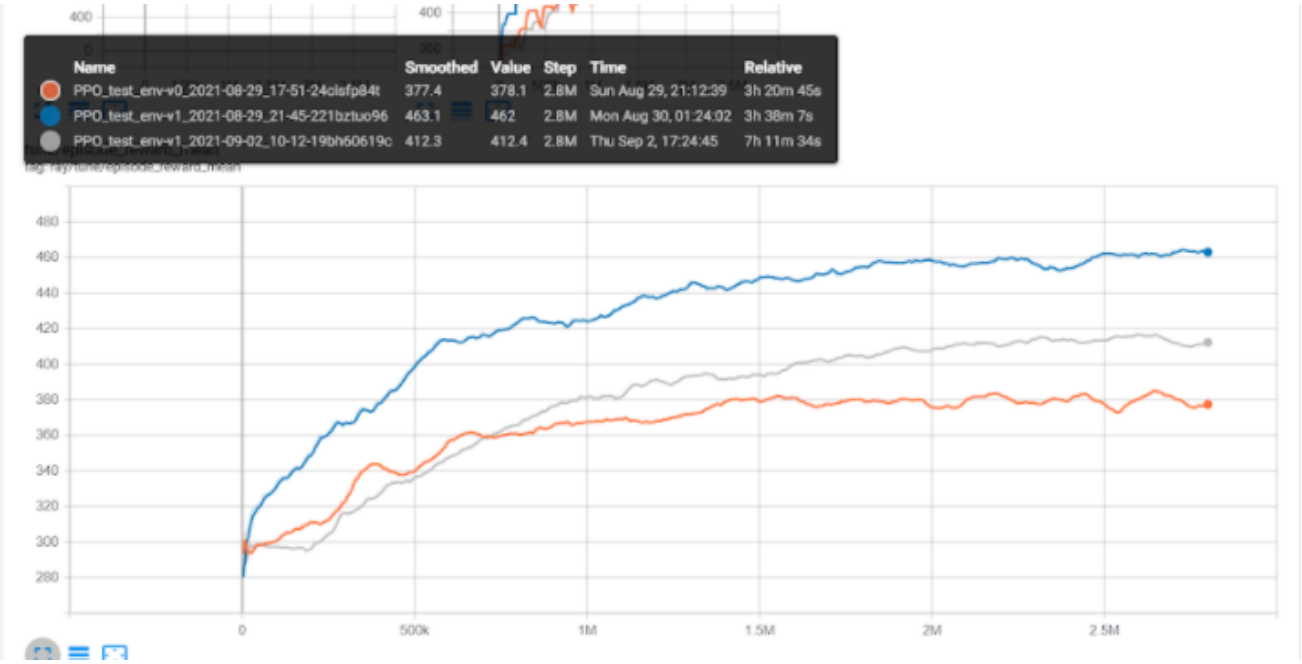

Figure 22: *Results of only traffic signal control Vs joined system*

In this system the different lines correspond to the following:
- V2I only traffic signal control
- No V2I only traffic signal control
- V2I joined optimization system

First of all, as we can see in the black box, the amount of time needed for the whole system including the V2I and the speed advisory is considerably bigger. We have a more complex system so more time for it to converge is needed. As we can see, the results are not as good as we expected. What we were looking for was a complex system that, when including the speed advisory part, it could give us better results with V2I, but what we have obtained is not that. The amount of time needed for each simulation in this case is quite large so the no V2I part has not been simulated as long as we are not obtaining what we wanted and the important part is to know the results of the whole system.

As the code has been written, the agent in the case of the complete system, had the ability of modifying the velocity of the cars. We are optimizing the speed of the vehicles per lane between 15 and 20 m/s. Regarding this, and remembering that in the previous case the max velocity was 20 m/s, what we should expect from the simulations is at least the same behavior than in the previous case.

We can say that the simulations have been done with the default configuration parameters like the previous one but the fact of having a more complex system that requires more time and resources, is telling us that a better adjustment into the parameters should have been done. We can go through the definition of all parameters in order to know how to adjust them and see how the algorithm works to find a solution that could fit our system. Even though no more simulations have been done, some pages have been read in order to understand the adjustments that we could do. Including the Ray page [20] and [42].

11. CONCLUSIONS

This project has been a very big challenge. We have been able to go through the current state of Deep Reinforcement Learning in general and in particular for traffic signal control. We have also been able to appreciate how our problems in traffic affect many other aspects of our lives, direct and indirectly.

The implementation of V2I technologies with the development of new technologies has shown to be also very important. We have seen in the results that when the infrastructure is able to have information of the vehicles the results are considerably better. This implementation is not immediate or cheap but it is an emerging solution that is going to change many lives.

In relation to the development of the whole system. As far as we have been able to reach, we have seen that giving the vehicles the possibility (speed advisory) of increasing or decreasing their velocities, could give also quite good results but we do not have reach a simulation where that can be proved. Of course, this complex system needs a great number of resources and time but if a good solution could be found this solution to traffic congestion problems could give us great results.

In this case only a single intersection has been simulated and even with this little system the number of resources and time, not just for training, but for preparing and analyzing the results has been enormous. Programming is not simple and developing systems for big cities is not going to happen from one day to another but there is always a beginning. We need to keep always is mind that changing the first link of the chain sometimes is not as easy as, but it can really change the world.

Bibliography

[1] Maxim Egorov. Multi-agent deep reinforcement learning. *Stanford University*, 2016.

[2] Alvaro J.Calle-Laguna, Jianhe Du, Hesham A. Rakha. Computing optimum traffic signal cycle length considering vehicle delay and fuel consumption. *Transportation Research Interdisciplinary Perspectives*, 3:100021, 2019.

[3] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[4] Marr Bernard. What is deep reinforcement learning. *Forbes*, 2020.

[5] Johannes V. S. Busch, Vincent Latzko, Martin Reisslein, and Frank H. P. Fitzek. Optimisedtraffic light management through reinforcement learning: Traffic state agnostic agent vs.holistic agent with current v2i traffic state knowledge. *IEEE Open Journal of Intelligent Trans-portation Systems*, 1:201–216, 2020.

[6] Unknown. The true cost of congestion. *International fleetworld*, 2017. (jhmb, 2020)

[7] Allan M de Souza, Celso ARL Brennand, Roberto S Yokoyama, Edmundo RM Madeira, Leonardo A Villas, and Erick A Donato. Traffic management systems: A classification, review, challenges, and future perspectives. In *2017 International Journal of Distributed Sensor Networks*, volume 13(4), 10 2016.

[8] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

[9] S. Djahel, N. Jabeur, R. Barrett and J. Murphy, "Toward V2I communication technology-based solution for reducing road traffic congestion in smart cities," *2015 International Symposium on Networks, Computers and Communications (ISNCC)*, 2015, pp. 1-6, doi: 10.1109/ISNCC.2015.7238584.

[10] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

[11] Deepeka Garg, Maria Chli, and George Vogiatzis. Deep reinforcement learning for autonomous traffic light control. In *2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*, pages 214–218, 2018.

[12] Unknow. Inrix. Global traffic scorecard. *Inrix*, 2021.

[13] Rajitha Jayasinghe. Reinforcement learning based traffic optimization at an intersection with glosa. *Technische Universitat Chemnitz*, 2019.

[14] Sergios Karagiannakos. The idea behind actor-critics and how a2c and a3c improve them. *Sensors*, 2018.

[15] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. Deep reinforcement learning for traffic light control in vehicular networks. *CoRR*, abs/1803.11115, 2018.

[16] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. A deep reinforcement learning network for traffic light cycle control. *IEEE Transactions on Vehicular Technology*, 68(2):1243–1253, Feb 2019.

[17] Campos Luis. Deep reinforcement trading. *ETS Asset Managment Factory*, 28/11/2018.

[18] Tom Silver and Rohan Chitnis. Pddlgym: Gym environments from pddl problems, 2020.

[19] The Flow Team. Flow, 2020.

[20] The Ray Team. Ray, 2020.

[21] Unknown. Gym. OpenAI, 2020.

[22] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. A survey on traffic signal controlmethods, 2019.

[23] Cathy Wu, Kanaad Parvate, Nishant Kheterpal, Leah Dickstein, Ankur Mehta, Eugene Vinit-sky, and Alexandre M Bayen. Framework for control and deep reinforcement learning intraffic. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*,pages 1–8, 2017.

[24] Zehui Xiong, Yang Zhang, Dusit Niyato, Ruilong Deng, Ping Wang, and Li-Chun Wang. Deep reinforcement learning for mobile 5g and beyond: Fundamentals, applications, and challenges. *IEEE Vehicular Technology Magazine*, 14(2):44–52, 2019.

[25] Jiachen Yang, Jipeng Zhang, and Huihui Wang. Urban traffic control in software definedinternet of things via a multi-agent deep reinforcement learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 22(6):3742–3754, 2021.

[26] Changxi You, Jianbo Lu, Dimitar Filev, and Panagiotis Tsiotras. Autonomous planning and control for intelligent vehicles in traffic. *IEEE Transactions on Intelligent Transportation Systems*, 21(6):2339–2349, 2020.

[27] Fanyu Zeng, Chen Wang, and Shuzhi Ge. A survey on visual navigation for artificial agents with deep reinforcement learning. *IEEE Access*, PP:1–1, 07 2020.

[28] Junjie Zeng, Rusheng Ju, Long Qin, Yue Hu, Quanjun Yin, and Cong Hu. Navigation in unknown dynamic environments based on deep reinforcement learning. *Sensors*, 19:3837,09 2019.

[29] Ding Z., Dong H. (2020) Challenges of Reinforcement Learning. In: Dong H., Ding Z., Zhang S. (eds) Deep Reinforcement Learning. Springer, Singapore.

[30] Kaelblig L.P., Littman M.L., Moore A.W. Reinforcement Learning: A Survey. *Journal of Artificial Research,* May 1, 1996.

[31] Fenjito Y., Benbrahim H. Deep reinforcement learning overview of the state-of-the-art. *Journal of Automation Mobile Robotics and Intelligent Systems.* 2018.

[32] Yuxi Li. Deep Reinforcement Learning: An Overview. *Cornell University.* 2018.

[33] Arulkumaran Kai, Deisenroth Marc Peter, Brundage Miles, Bharath Anil Anthony. A Brief Survey of Deep Reinforcement Learning. *IEEE Signal Processing magazine, special issue on Deep Learning for image understanding.* 2017.

[34] Wan1 Chia-Hao, Hwang Ming-Chorng. Value-based deep reinforcement learning for adaptive isolated intersection signal control. *IET Intelligent Transport Systems.* 2018

[35] Silver David, Lever Guy, Heess Nicolas, Degris Thomas, Wierstra Daan, Riedmiller Martin. Deterministic Policy Gradient Algorithms. *Proceedings of Machine Learning Research*. PMLR 32(1):387-395, 2014.

[36] Mark Lee. *Incomplete Ideas.* 2005-01-04

[37] The Sumo Team. Sumo 2020

[38] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, Pieter Abbeel. "Benchmarking Deep Reinforcement Learning for Continuous Control". *Proceedings of the 33rd International Conference on Machine Learning (ICML), 2016.*

[39] The Anaconda Team. 2020

[40] The OpenAI Team. 2015-2021

[41] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, Tsuhan Chen. *Recent advances in convolutional neural networks.*

[42] II Sourcel, *Best Practices when training with PPO*. Github 2021.

[43] Vishnu Vijayan PV. Deep Reinforcement Learning: Artificial Intelligence, Machine Learning and Deep Learning-Introduction, Overview and Contrast for Beginners. *Medium.* 2020.

[44] The python team. Python. 2021

[45] Yufeng G. Which Python package manager should you use?. *Medium*. 2017