

Université Paris Cité

UFR Mathématiques et Informatique

**Safe Reinforcement Learning for Navigating Complex Urban Environment**

Master 1 Réseaux et Systèmes Autonomes

Mathéo DESSAUVAGES – Jules CHERRION

Supervised by Xinchen HAN and Hossam AFIFI

Academic year 2022 – 2023

Table of contents

[1. Introduction](#_Toc135244534)

[2. Concepts and existing methods](#_Toc135244535)

[2.1 Neural Networks & Convolutional Neural Networks](#_Toc135244536)

[2.1.1 Neural Networks](#_Toc135244537)

[2.1.2 Convolutional Neural Networks](#_Toc135244538)

[2.2 Reinforcement Learning](#_Toc135244539)

[2.2.1 Markov Decision Process](#_Toc135244540)

[2.2.2 Partially Observable Markov Decision Process](#_Toc135244541)

[2.3 Safe Reinforcement Learning](#_Toc135244542)

[2.3.1 Constrained Markov Decision Process](#_Toc135244543)

[2.3.2 Constrained Policy Optimization](#_Toc135244544)

[2.3.3 Policy Learning with Constraints](#_Toc135244545)

[2.4 Deep Q-Learning](#_Toc135244546)

[2.5 CARLA](#_Toc135244547)

[3. Contribution](#_Toc135244548)

[3.1 Carla setup](#_Toc135244549)

[3.2 Neural Network model](#_Toc135244550)

[3.3 Deep Q-Learning](#_Toc135244551)

[3.4 Results](#_Toc135244552)

[3.5 Shortcoming](#_Toc135244553)

[4. Conclusion](#_Toc135244554)

[5. Bibliographie](#_Toc135244555)

# Introduction

For several years now, the development of autonomous vehicles has been rapidly expanding and they will be starting to appear inside urban environments soon. However, they must process their environment in real time to make decisions and have the safest behavior. So, this project consists in providing a solution using Safe Reinforcement Learning and applying it to a simulation.

To achieve this, we decided to use a simulator known as CARLA. It is a simulator that allows you to emulate a small city and vehicles having different types of captors running through them.

The first step for this project was to create an AI that can drive a car autonomously. Then after adding a safety layer to obtain a safe AI. However, we realized with our supervisor that given the time we had, it was too little for us to do both things.

We lost quite some time because all subjects were almost new to us. We indeed worked a bit on machine learning, but it was only the theorical basics without practical application. So, we were provided a lot of documentation which was not the easiest way to learn, and we spend almost a month to correctly understand the basics of safe reinforcement learning.

We also lost time on another project that we thought could have helped us achieving good results in reinforcement learning but it was totally different and the things we learned, except the base of how to use TensorFlow and Keras, were not usable. The project was about processing electricity consumption and predict the activation curves of different devices. We ended with good results using Remanent Neural Networks (RNN) but there weren’t suited for this project (we still used some but with disappointing results (see part 3.2)).

In result we only tried to produce an AI capable of driving a car inside a city created inside CARLA.

# Concepts and existing methods

## Neural Networks & Convolutional Neural Networks

### Neural Networks

Neural networks are a type of machine learning algorithm inspired by the structure and function of the human brain. They are organized in interconnected neurons divided into multiple layers. Each neuron obtains information from the previous layer then computes it and gives the result to the next layer and each connection has a weight. It is a powerful tool to process complex information or data.

### Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a class of neural networks used for pattern recognition such as in images or sequences. They are mainly composed of convolutional layers which apply some filters to the input data. There are very useful when you use inputs that are shaped like matrices (like an image). It’s one of the most used types of neural Network when working on image recognition or image processing because it mixes well the data of the matrix. In more simple terms it takes pixel with their surrounding into account which is better to determine pattern inside you image.

## Reinforcement Learning

Reinforcement Learning is a type of unsupervised machine learning that aims to train an agent to make decisions within an environment to maximize a reward score. The agent interacts with the environment in sequential steps series. At each step, the agent gets an observation of the environment and receive the reward based on the previous step action then decide the action to do and the environment receives the action done by the agent, then send the observation and the reward to the agent.

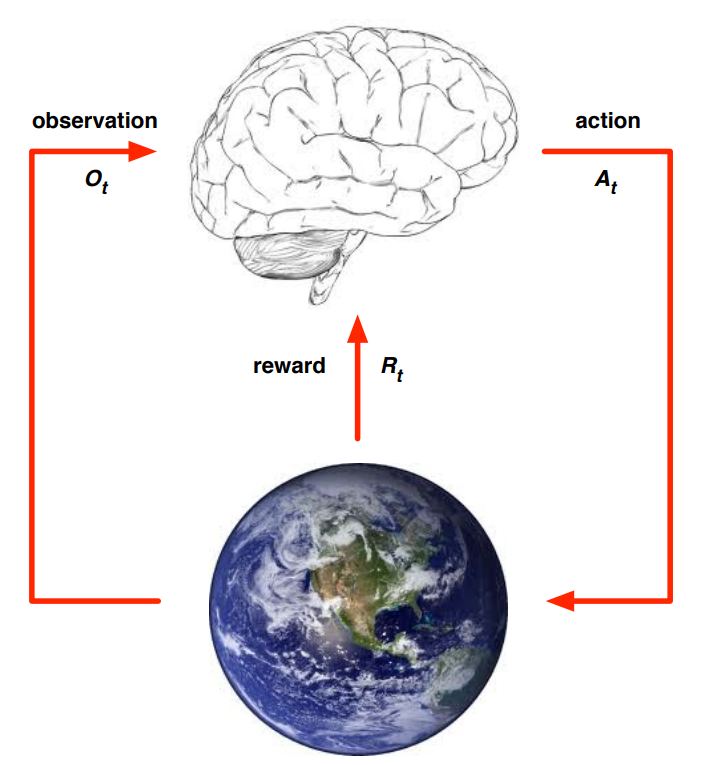


Figure 1: Representation of an agent (brain) and an environment (Earth) (Silver 2015)

Both agent and environment have a state for each step, the agent state contains the useful information to make decisions and the environment state contains the information for the observation and for the reward. If the agent state and the environment state are the same it is called Markov Decision Process (MDP), otherwise it is called Partially Observable Markov Decision Process (POMDP).

### Markov Decision Process

A Markov decision process can be used to describe an agent’s sequential decision and where the decision can lead to consequences or outcomes. It can be represented as a “tuple, , where is the set of states, is the set of actions, is the reward function, is the transition probability (where is the probability of transitioning to a state given that the previous state was and the agent took action in ), and is the staring state distribution. A stationary policy is a map from states to probability distributions over actions, with denoting the probability of selecting in state ” (Achiam, et al. 2017). When an agent chooses an action at a time step, the environment state changes using the transition function then it receives the reward corresponding to the state. The choice of future actions is determined by a discount factor value between zero and one that focuses on long-term when near one or immediate rewards when near zero.

### Partially Observable Markov Decision Process

POMDPs rely to cases where the agent does not have a full visibility on the environment components and values than it must try to determine its own state based on its observations. This self-determined state is only a possible true state because of the lack of information to check the accuracy of the calculations. Unlike MDP, this process makes an agent make decisions based on the belief that the state is a correct state.

## Safe Reinforcement Learning

Safe Reinforcement Learning is based on Reinforcement Learning but unlike the latter, the objective is not to maximize the reward score of an agent but to ensure that the decision it makes are in keeping with defined safety boundaries and constraints. Learning optimal and safe policies addresses the challenge to “balance between exploitation and exploration, especially when we need to improve reward performance while satisfying cost constraints” (Gu, et al. 2022). There are multiple approaches to satisfy this problem such as the Constrained Policy Optimization and the Policy Learning with Constraints. These two approaches use an extension of Markov decision Process called Constrained Markov Decision Process (CMDP).

### Constrained Markov Decision Process

The CMDP incorporates constraints inside the agent’s decision-making process so that it optimizes its behavior to respect some limitations or constraints. So that the CMDP can be defined as a tuple where is the set of cost functions as “” (Liu, Halev et Liu 2021). An action is then considered feasible if it satisfies all the necessary constraints and the optimal policy is the one that maximizes in a long-term way the reward score and satisfies all the constraints for each action.

### Constrained Policy Optimization

The Constrained Policy Optimization process involves defining the boundaries of what constitutes a safe behavior for the agent. It will then try to find the policy that maximizes the reward score while keeping up with the safety boundaries. The CPO is based on the research of the optimal policy using Markov Decision Process because for “large or continuous MDPs, solving for the exact optimal policy is intractable due to the curse of dimensionality” (Achiam, et al. 2017). So, the policy search tries to maximize the performance indicator and update the current policy.

### Policy Learning with Constraints

This approach consists in letting the agent learn policies while interacting with the environment while knowing and satisfying safety constraints. “The Constrained Markov Decision Process (CMDP) becomes an important model for constrained sequential decision-making problems. In a CMDP, the objective is to maximize long-term reward while keeping certain costs under their respective constraints” (Liu, Halev et Liu 2021). The idea is to guide the learning process to make sure that the agent will evolve in an environment and behave so that it remains within reasonable boundaries.

## Deep Q-Learning

Deep Q-Learning is a Reinforcement Learning algorithm that uses Q-Learning and Deep Neural Networks. Q-Learning is an algorithm that train an agent to choose an optimal policy for the decisions it will make. The agent tries to maximize the reward score by testing which of the possible actions is the better. At the beginning the agent has not any knowledge of the environment and take random actions then store the result and the associated state in a table, as the agent is learning it starts to choose actions that give better rewards according to the state. To achieve this goal, the Bellman equation is very useful because it expresses the relationship between the value of a state-action pair and the values of the neighbors’ pairs. The equation is the following: where is the value of the state and the action pair, is the immediate reward obtained by doing action in state , is the discount factor for future rewards and is the maximum value among the possible actions ’ in the next state .

## CARLA

CARLA is the open-source simulator we used to apply our reinforcement learning algorithms. It provides 3D urban environments and different scenarios for autonomous driving simulation. This simulator is used by researchers and developers to test and evaluate their methods and algorithms inside a virtual system that aims to have the nearest from reality behavior. To implement these methods and algorithms it is necessary to be familiar with the sensors feature provided by CARLA. For instance, the cameras offer images colors analysis, depth information and features about the elements inside the field of view, the radar gives the distance, the velocity, and the angle of detected objects in the map, the LiDAR measures time between the emission of laser beams and when they are bouncing back after hitting objects. The sensors set also includes GPS sensors and Inertial Measurement Units (IMU) used in autonomous vehicles to get their velocity, acceleration, and orientation. The implementation of the algorithms uses the Python API provided by CARLA which is also used for the default scenarios.

# Contribution

## Carla setup

For the project we use the default map of CARLA 0.9.13 which is a small city with large main road and few small streets (Figure 2). As for the car we use a Tesla Model 3 (Figure 3).



Figure 2: CARLA’s environment

Figure 3: Tesla model 3 in CARLA

On the car we setup two sensors, a RGB camera and a radar both situated at the front of the car. The RGB camera returns an array of pixel values, each pixel has 4 color values (RED, GREEN, BLUE, ALPHA) and the radar has an array of points where each point has 4 values (altitude, azimuth, depth, velocity). We will use these data to feed our neural network.

It’s important to note that CARLA tries to simulate a realistic physical behavior so its environment includes lots of parameters that can alter the capacity of the AI. You have wind, weather (rain, cloud, sun) that can modify the data your sensors get and the control of the car. When the sun is down your RGB camera will have higher levels of RED and less of BLUE, rain might create “artifacts” for the camera and so on. To avoid having a too complex environment for us we decided to just have one weather for our trainings (the sun position is different for each but no rain, cloud, or wind).

With some script given by CARLA you can also spawn other cars that are directly connected to CARLA core and run as NPC in a game to simulate a realistic urban traffic. We also decided to not use this option to avoid having a too complex situation and save a bit of resources for our computers. CARLA consumes a lot of resources alone (Figure 3) even with a good computer so we avoided overloading our computer so that we could avoid having endless hours of training.

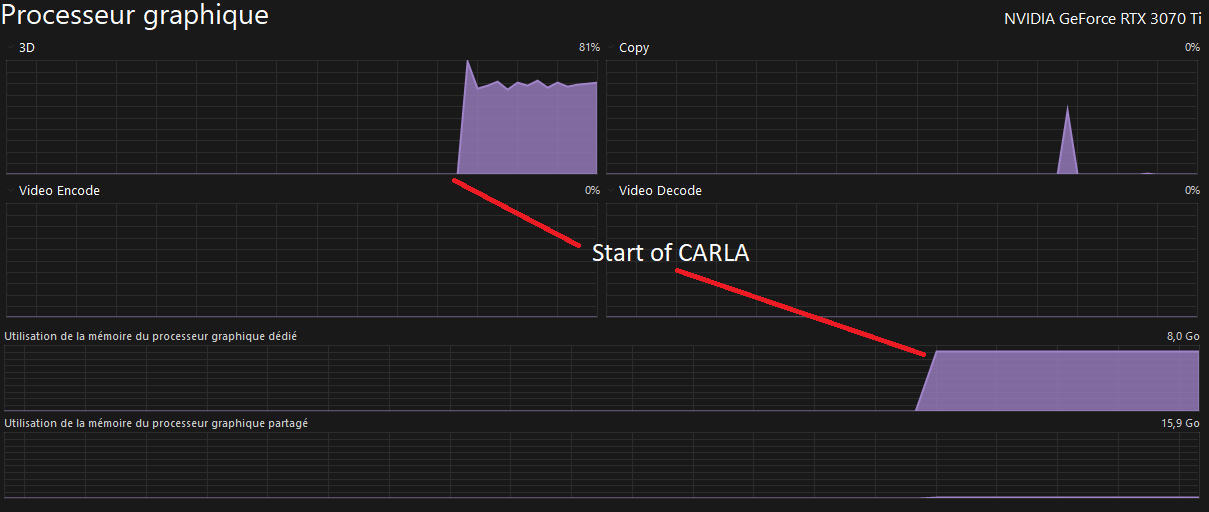


Figure 4: CARLA ressources consumption on a RTX 3070 Ti

## Neural Network model

To achieve our goal, we used a Convolutional Neural Network with a slightly modified architecture to take in consideration the two inputs (RGB camera and radar) (Figure 5).

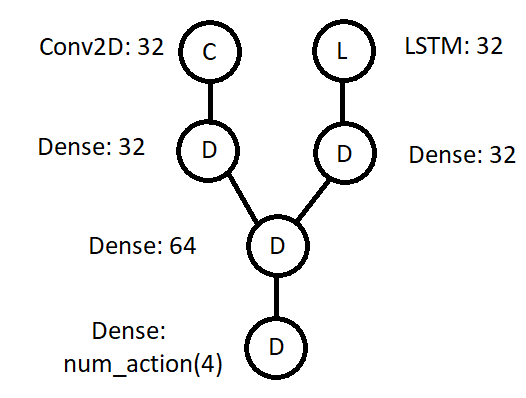


Figure 5: Neural Network's architecture

We inserted the data from the camera inside the Conv2D layer and the data from the radar inside the LSTM layer. We use this type of layers so that the network doesn’t “forget” what the last inserted data was. Then we pass the results to dense layers to process them and then concatenate the output to the 64 neurons dense layer. Finally, we have one last dense layer where the number of neurons is the number of actions we have (4 here). All layers have a ‘ReLU’ activation except for the last one which is ‘linear’ because we want to be able to see a clear difference between the outputs to choose what are the best actions to train our model. The Optimizer is Adam with a learning rate of 0.001.

The use of a LSTM layer is disputable here, we expected that if you have a point in the radar that moves toward you (or if you move toward it) it will always appear at each step before you crash. So, the thought that having a “long-short term memory” layer might be useful because we can see an evolution in some points as a time data-series. However, you will always have multiple points that are not useful at all, and we highly doubted that the model would be able to interpret each point evolution correctly. It could be a project because of the random number of points you get and their position in the array changes every step. You have a very high complexity problem that might need data from another sensor to determine. But this is not our project.

To effectively train our model we used Deep Q-Learning which we will explain properly.

## Deep Q-Learning

You can see CARLA as the body, the Neural network as the brain and Deep Q-Learning is the trainer that educates the brain. We have four possible actions and after taking one, given the situation the car is in we decide of a reward which is simply a numerical value. The four actions possible are:

* Go straight with 100% gas and 0% steer.
* Go right with 50% gas and 50% steering.
* Go left with 50% gas and 50% steering.
* Break with 50% of break power.

And we defined the reward based on the speed the vehicle has:

* Inferior to 10hm/h: -0.05
* From 10 km/h to 19km/h: 0.05
* From 20 km/h to 29km/h: 0.1
* From 30 km/h to 39km/h: 0.2
* From 40 km/h to 49km/h: 0.05
* Superior to 50 km/h: -0.2

And if the vehicle collided with something (reward = -1.0)

We also tried to give reward with traffic lights however it didn’t give out good results, we might need to change the value of those rewards but for simplicity we decided to ignore those.

To choose an action there are two possibilities, the first one is taking a random action, the second one is to ask the neural network to compute an answer and to select the output with the higher value. To determine the method to choose we have an epsilon value that decrease in time, the lower its value is the more you ask your neural network to select an action. After choosing an action, doing it and give a reward you save all the data of the current step to train your neural network later using Bellman equation. We update the weights of our model depending on a set frequency.

## Results

It is hard to interpret the results we had. At the end of the training, we had an average reward of 0.5 which is not that bad for us that just started learning AI and CARLA.

The model did learn something which was reassuring, our work was not for naught, but the results were not very good given our expectations. The car, most of the time started to turn and spin endlessly on itself. Well not endlessly because given a certain time it always ended with a collision on something.

This result is quite logical, the better way to go at a steady place without crashing is to turn around in circle if there is enough space for you to do it.

## Shortcoming

We compared our results with a version a bit like ours but that is only using a RGB camera and has different reward’s policy and network (Kinsley s.d.). In the end we ended up with the same results, a car spinning round. Given that their neural network is simpler than our and the data they collect fewer we probably could had done better.

Also, we had trouble with the resources. As said earlier CARLA is a simulator that requires a lot of computing power. If you add the computing power needed to train a neural network, it becomes rapidly a problem even more when you try to debug and check your result. You must wait for a whole night (with your computer as a heater) to get result and discover that it’s not good because of a simple mistake and need to restart from scratch with is very time consuming.

# Conclusion

We didn’t achieve autonomous driving at all, but we learned a lot during this project. The results were a bit disappointing given the time and resource invested but we see a lot of ways to improve them.

Firstly, we should have defined a spawning point for our vehicle and trained it from it, this would provide a static environment. We would then try to train it in other environments to generalize. We also might have focused on having one sensor like the RGB camera but do different process on the image received to isolate interesting features instead of feeding the image data only with a simple process of scaling the value of each color between one and two.

The data that the radar gives are also very hard to interpret for a neural network because the size of your set varies for each input which increases the difficulty of finding patterns for the NN (neural network). At the beginning we thought that it was a good idea because for us it is simple with it to determine with a simple algorithm that there is an object is approaching in a worrying range. However, something that might be simple for an algorithm can sometimes be hard to interpret for a NN if you can’t give it any clue about how to use it. Which we didn’t do because we didn’t know how to do.

To avoid having the car driving in circle we should have done some sort of waypoints to give more rewards or even give rewards based on the distance to the point the car appeared.

One thing that we learned from this project is that to learn to navigate in a complex environment the best way might be to learn in a simple controlled environment to get a good understanding of the environment then do more complex training with the same model but still in a controlled environment. It is possible to achieve it with CARLA given its valuable API. However, this might be useful to see which direction is the better to train an autonomous car, but we highly doubt that a model trained on CARLA would be able to drive in real life.

One thing that we also didn’t do at all is image recognition and this, for me (Jules) might be the best for autonomous driving. I don’t think deep Q-learning alone is the answer to autonomous driving. For me the best way is to have solutions capable of gathering the most useful information possible as the speed limit, potential danger etc.… Then process them (maybe with deep Q-learning) to have a good result. We shouldn’t let an AI try to get those data; we need to break down this big problem of autonomous driving into multiple simple problems (getting all the data) then you can have an AI to do the job. And one of the best ways to get those data today is image recognition for now. You have signs on every road of every country that have the data you need in a formatted way. It would be very easy nowadays to get the meaning of a sign with image recognition.

Even if the results were not satisfying, we learned a lot of things during this project about neural networks that will be very useful in the future. You learn more if you do mistakes and this project is a good example of it.

# Bibliographie

Achiam, Joshua, David Held, Aviv Tamar, et Pieter Abbeel. 2017. «Constrained Policy Optimization.»

Gu, Shangding, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, Yaodong Yang, et Alois Knoll. 2022. «A Review of Safe Reinforcement Learning: Methods, Theory and Applications.»

Kinsley, Harrison. s.d. «Introduction - Self-driving cars with Carla and Python.» https://pythonprogramming.net/introduction-self-driving-autonomous-cars-carla-python/.

Liu, Yongshuai, Avishai Halev, et Xin Liu. 2021. «Policy Learning with Constraints in Model-free Reinforcement Learning: A Survey.»

Silver, David. 2015. «Introduction to Reinforcement Learning with david Silver.» *DeepMind.* https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver.