Honor's programme report

Federico Vergallo, mat. 1815521

December 22, 2020

1 Introduction

In the setting of Autonomous Driving, the agent is required to learn new configurations of the environment, as well as to predict an optimal decision at each instant while driving in its environment [3]. In such scenario, a Reinforcement Learning setting, where the agent is required to learn and represent its environment as well as act optimally given at each instant, is suitable to solve the task. It is mostly for the task of motion planning in such a way to provide the most suitable action to take to the vehicle in a certain moment considering the environment.
The project developed aim at implementing a reinforcement learning algorithm for autonomous lane changing developed using SUMO simulator for an agent that has to run together with a fleet of autonomous safe vehicles using SUMO simulator. The results shows that the agent were able to decrease its collision rate while increasing its speed. The code is available here `https://github.com/federicovergallo/sumochanginglaneagent`.

1.1 Related Work

Most approaches uses occupancy grids to describe what's in the neighborhood of the vehicle in which each cell is filled with information of that specific vehicles as well as the measures of the ego vehicle w.r.t. that vehicle. In [1] use as state representation the following features: lateral position, longitudinal velocity, steering angle, throttle value, distance and velocity of each vehicle in the neighborhood of the ego vehicle. In [2] they simply put the velocity of the ego vehicles together with relative distance and velocity of the six surrounding vehicles. In [7] they put in the state only information regarding the current ego vehicle's lane and its target lane, in other words the lane that the agent is considering to go. In this way they claim that the agent focuses only on important regions. In [6] they do the same but considering four vehicles in the target lane and only the vehicle in front the ego one. More sophisticated is the representation in [4] and [5]. In [4], instead of using the regular occupancy grid as state representation, the they encode a variable number of vehicles (the maximum number perceived by the sensors) into an embedding vector. In [5] they encodes different kind of objects in the same encoding space and they integrate a Graph Network that aims at explicitly model road object interaction.
Regarding the action space, in most of the applications reviewed, the action were high level commands: stay in the lane, change the lane on the right or on the left. The control of the velocity was left to lower levels of the car's control.
Not many paper use some safety module that was usually left to the car's control module and to the simulator or in some sort integrated in the reward function, such as in [6] that penalizes the agent for being near to collision. In [2], after the network outputs the actions, a formal method checking module was activated in order to check if an action would be too dangerous for the agent.

1.2 Simulators

Based on my research, I tried two simulators that are very popular in the field: CARLA and SUMO.

1.2.1 CARLA

CARLA [8] is an high realistic open source simulator for autonomous driving research. It provides to the developer a wide range of sensors measurements that a car could get: LiDar, cameras, depth camera, depth sensor, and so on. CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely.
No doubt that CARLA is one of the best in the market and it's also used by big companies (Toyota is one of the founder). It provides more common used market sensors and its realness would make more clear the result of a trained algorithm. On top of that a big community of CARLA users and more tutorials are available.
The main issue with CARLA is that it's impossible to create a custom road and so, you can't try some particular setting. This is because is required to have a commercial version of a software called RoadRunner which is very expensive. Because of that I had to switch to SUMO.

1.2.2 SUMO

"Simulation of Urban MObility" (SUMO) [9] is an open source, highly portable, microscopic and continuous traffic simulation package designed to handle large networks. It allows for intermodal simulation including pedestrians and comes with a large set of tools for scenario creation.
Contrary to CARLA, SUMO has a basic graphics that will guarantee to the user a fast execution of algorithms. Sumo was born with the intention of simulating traffic scenarios mostly. Because of that, a wide range of urban elements are available to the user in order to customize the scene.
SUMO works together with another software: NetEdit. This allows to the user to design the road as prefers, with highway, crossings, stops, sidewalks, building and so on. It generates an XML file that describes the scene that, together with a configuration file describing the initial setting of vehicles, is needed to SUMO in order to run the simulation.
Furthermore, SUMO acts as TCP based server and allows multiple client to access to it and it's possible to access SUMO's information using an API called TraCi.

2 Project

The project aims at developing a reinforcement learning application to make an agent drive safely in a condition of dense traffic. For doing so, SUMO was used to simulate the behaviour of the ego vehicle together with a fleet of autonomous vehicles and to train two model of RL algorithms, namely DQN and A2C.
The development flow was the following: creating the highway with NetEdit, get the parameters of the simulation, create a custom Gym environment and train the networks.

2.1 Highway building using NetEdit

To make a simple scenario, a simple circle highway with three lanes is developed. It's easy to create an highway with NetEdit: just put some points in the workspace that will create some portions of streets Fig. 1, then you can connect each portion to create curves Fig. 2. With NetEdit it's easy to control the parameters of the street, such as the maximum speed, the length and the number of lanes.

Figure 1: Highway is a connection of street portions.

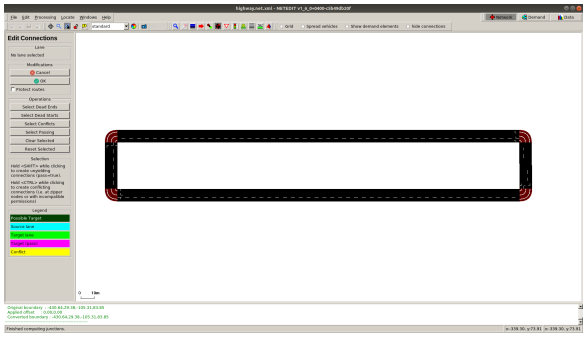

Figure 2: Final highway with proper curves.

2.2 Start the simulation using TraCi

NetEdit outputs and XML file that, together with another XML file for the objects specification, make up a config file tha SUMO takes as input. From SUMO one can start the simulation. For reinforcement learning purposes, SUMO's creator release a python library to get the parameters of the simulation, namely TraCi. This works by connecting the python script to SUMO in a TCP connection.
I created a custom Gym environment to provide state space, reward and observation based on the info that I was able to collect through TraCi.

2.3 Simulation details

The simulation aims at making the agent run in a dense traffic scenario without collision with other vehicles. To make a realistic scenario of traffic, 35 autonomous vehicles were added to the simulation other than the ego vehicle to have a moderate traffic.

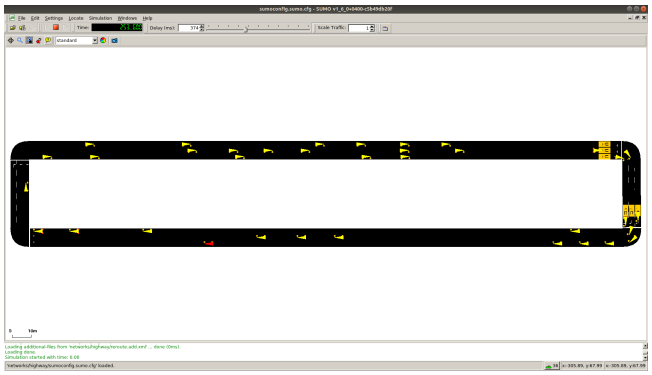

Figure 3: Traffic scenario. Red vehicle is the agent

SUMO allows to the user to control the safety level of the simulated autonomous vehicles by changing a parameter called `laneChangeModel`. This parameter tells to the autonomous vehicle to do or do not change lanes for the following reasons: strategic change lanes to continue the route, cooperative (change in order to allow others to change), speed gain (the other lane allows for faster driving) and obligation to drive on the right. For the simulated vehicles, the change lane mode was set to 256, meaning that they can do any lane change they want as long as it's safe. Meanwhile, the ego vehicle, represented in red in 3, has the `laneChangeModel` parameter set to 0 because the change decision must be taken by the network.

2.3.1 State space

As state space representation, I took inspiration from [7]. Starting from the position of the ego vehicle, by using Traci, I first collect all the vehicles in its neighborhood (the most close in a certain radius). The, for each vehicle, I represented it with the following information: lateral position, longitudinal speed, acceleration and distance. The ego vehicle is represented with the same features but augmented with the longitudinal position. According to this, the state space is represented with a vector of 37 features: 5 feature for the agent, 4 for each of the possible 8 vehicles around the ego vehicle.

2.3.2 Action space

The action space is simply made of three choice: stay in the current lane, change to the left and change to the right.

2.3.3 Reward function

Same as the state space, the reward function is borrowed from [7]. The function they propose aims at satisfying three features of driving: comfort, safety and efficiency.
The comfort parameter regards the evaluation of jerk (lateral and longitudinal direction): if the jerk is high means there have been a sudden change of acceleration, that will cause a discomfort.
The reward of the efficiency is made of two factor: one regarding the lane, in other words if the agent is in the desired lane or not, and one regarding the speed, in order to make the agent be as fast as possible.
The safety rewards regards the absence of collision: if collide it will be highly penalized meanwhile it receives a little reward at every step it does not collide. Those three macro reward function will be eventually added together weighted by some predefined weights.

2.4 Network details

Two types of network were trained: a DQN and an A2C. As mostly approach reviewed do, the network are implemented with a simple stack of fully connected layers. The Huber loss was used and Adam as optimizer. An epoch was considered concluded either after a collision or after 128 steps.

3 Results

As said two network was compared: DQN and A2C. Three run were made for both methods: a baseline run with parameters `num_vehicles=25`, `lr=0.001`, `layer1=64`, `layer2=128`, `layer3=64`, a second run with decreased weights in efficency and comfort reward so to weight more the safety and a third run same as the second but with `lr=0.01`.

3.1 A2C results

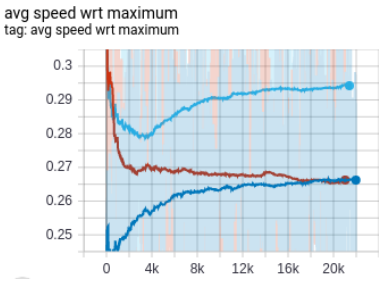

Figure 4: Avg speed wrt maximum in lane

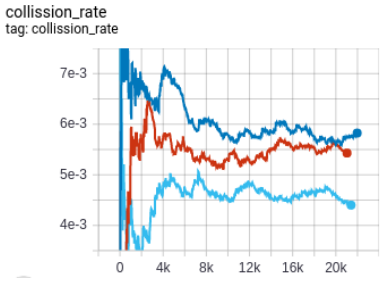

Figure 5: Collision rate

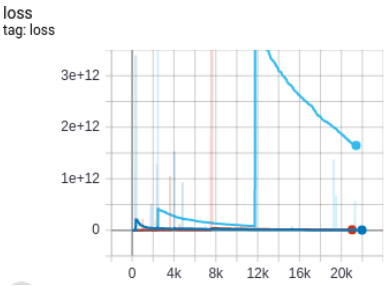

Figure 6: Loss

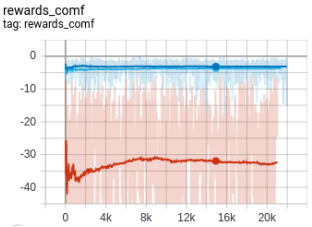

Figure 7: Reward comfort

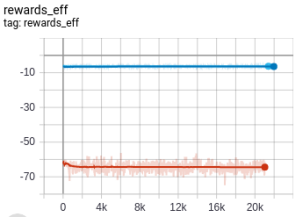

Figure 8: Reward efficiency

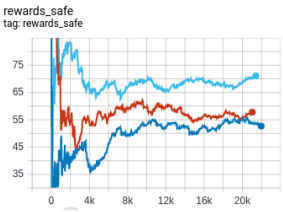

Figure 9: Reward safety

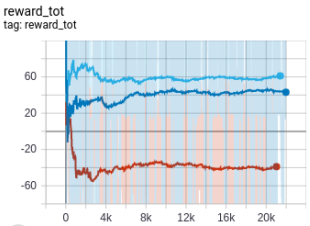

Figure 10: Total Reward

The blue line is the first run, the red one is the second and the light blue is the third one. As you can see the best one is the third, in terms of average speed with respect to the maximum velocity in lane, collision rate and total reward.

3.2 DQN results

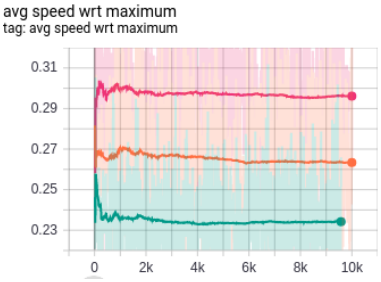

Figure 11: Avg speed wrt maximum in lane

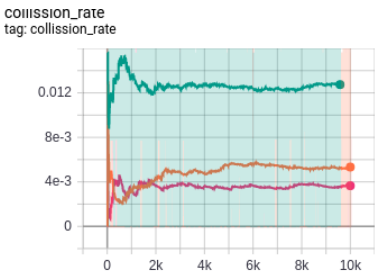

Figure 12: Collision rate

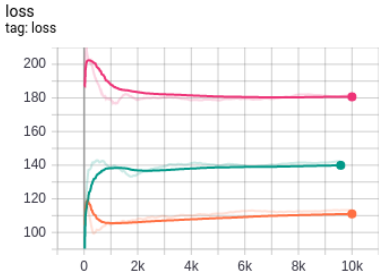

Figure 13: Loss

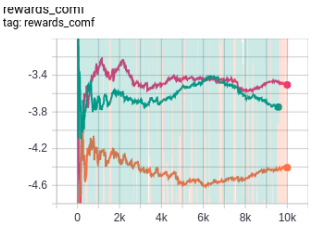

Figure 14: Reward comfort

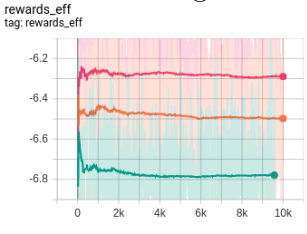

Figure 15: Reward efficiency

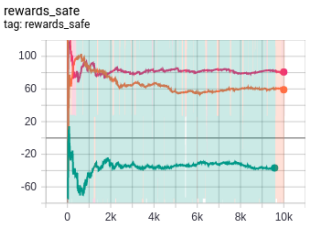

Figure 16: Reward safety

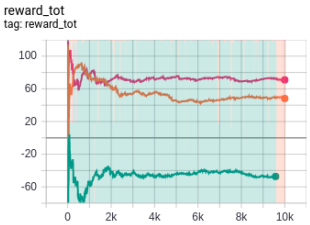

Figure 17: Total Reward

Here the green line is the first run, while the orange and pink line are the second and the third respectively. As A2C method, the third setting achieves better results. DQN shows an higher variance then A2C but an overall better performance.

4 Conclusion

As reviewed in the literature, Reinforcement Learning algorithm are getting more and more attention in the field of autonomous driving car as decision maker system. We reviewed methods for doing autonomous lane change safely. The last characteristics is usually delegated to some low level control system.
The project developed aim at implementing a reinforcement learning algorithm for autonomous lane changing developed using SUMO simulator. An agent, the ego vehicle, has to run together with a fleet of autonomous safe vehicles. The results shows that the agent were able to decrease its collision rate while increasing its speed. Qualitatively the tests shows that the agent collides few times and that it has understood to stay in the outermost line so to don't have to decrease the velocity near a bend so to have higher speed.

References

[1] Chen et al. "Autonomous Driving using Safe Reinforcement Learning by Incorporating a Regret-based Human Lane-Changing Decision Model". In: (2019).

[2] Mirchevska et al. "High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning". In: (2019).

[3] B Ravi Kiran et al. "Deep Reinforcement Learning for Autonomous Driving: A Survey". In: (2020).

[4] Huegle et al. "Dynamic Input for Deep Reinforcement Learning in Autonomous Driving". In: (2019).

[5] Huegle et al. "Dynamic Interaction-Aware Scene Understanding for Reinforcement Learning in Autonomous Driving". In: (2020).

[6] Wang et al. "Human-like Decision Making for Autonomous Driving via Adversarial Inverse Reinforcement Learning". In: (2020).

[7] Ye et al. "Automated Lane Change Strategy using Proximal Policy Optimization-based Deep Reinforcement Learning". In: (2019).

[8] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pages 1–16.

[9] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. "Microscopic Traffic Simulation using SUMO". In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. `https://elib.dlr.de/124092/`.