Universitat Oberta de Catalunya

Open University of Catalonia (UOC) Master's Degree in Data Science

MASTER'S THESIS

Area: M2.879 - TFM - Area 2 - Aula 1

Reinforcement Learning in Autonomous Vehicles with Limited Input

Author: Raül Villalba Rodríguez

Tutor: Raúl Parada Medina

Professor: Ismael Benito Altamirano

Tarragona, January 9, 2024

Credits/Copyright

A page with the specification of credits/copyright for the project (either application on one side and documentation on the other, or unified), as well as the use of third-party trademarks, products or services (including source code). If a person other than the author collaborated on the project, their identity and what they did must be explicitly stated.

Below is the most common case, but it can be modified for any other alternative:

FINAL PROJECT RECORD

Title of the project:	Reinforcement Learning in Autonomous Vehicles with Limited Input
Author's name:	Raül Villalba Rodríguez
Collaborating teacher's name:	Raúl Parada Medina
PRA's name:	Ismael Benito Altamirano
Delivery date (mm/yyyy):	01/2024
Degree or program:	Data Science Master's degree
Final Project area:	M2.879 - TFM - Area 2 - Aula 1
Language of the project:	English
Keywords	Automotive, Autonomous driving, Advanced Driver Assistance Systems (ADAS), Level L5 Autonomy, Reinforcement Learning, HD maps.

Dedication/Quote

A la família, por siempre estar ahí.

Abstract

The automotive industry is currently undergoing two simultaneous revolutions: electrification and autonomy. In recent years, various regulations that apply to all manufacturers have mandated the incorporation of advanced driver assistance systems (ADAS) into their vehicles. Manufacturers are already delving into the development of Level L3 autonomous functions that can be homologated. Additionally, numerous institutions and companies are pushing the boundaries, aiming to create prototypes that achieve Level L5 autonomy, enabling fully autonomous driving.

Within the scope of this project, we will delve into the cutting-edge developments in this field. Our objective is to implement and train a model utilizing reinforcement learning, and subsequently, compare its performance with other models. To ensure safety throughout the process, the training will be carried out within simulated environments. However, our ultimate goal is to seamlessly integrate the trained model into a real-world vehicle.

Keywords: Automotive, Autonomous driving, Advanced Driver Assistance Systems (ADAS), Level L5 Autonomy, Reinforcement Learning, HD maps.

Resumen

La industria de la automoción está experimentando actualmente dos revoluciones en paralelo: la electrificación y la autonomía. En los últimos años, diversas regulaciones que aplican a todos los fabricantes han introducido la obligación de incorporar diferentes sistemas avanzados de asistencia a la conducción (ADAS, por sus siglas en inglés), las cuales los fabricantes están explorando, y algunas de estas funciones ya cumplen con los requisitos para obtener la homologación de nivel L3 de autonomía. Además, numerosas instituciones y empresas están explorando ir más allá, buscando desarrollar prototipos que alcancen el nivel L5 de autonomía, que representa la conducción autónoma total.

En este proyecto, se analizará el estado del arte, con el objetivo de implementar y entrenar un modelo utilizando aprendizaje por refuerzo y luego compararlo con otros tipos de modelos. Por razones evidentes de seguridad, el entrenamiento se llevará a cabo en entornos de simulación. No obstante, nuestra meta final es integrar el modelo entrenado en un vehículo real.

Palabras clave: Automoción, Conducción autónoma, Advanced Driver Assistance Systems (ADAS), Nivel L5 de autonomia, Aprendizaje por refuerzo, Mapas de alta definición.

Contents

List of Figures

Chapter 1

Introduction

1.1 Proposal description and motivation

The prospect of autonomous vehicles revolutionizing the future of transportation is a compelling vision. It has been estimated that these advanced vehicles could potentially save 10 million lives per decade on a global scale (1). Such a statistic is nothing short of staggering, given that traffic accidents remain a leading cause of fatalities worldwide. While achieving complete autonomy in vehicles is undoubtedly a big challenge, it is precisely this profound potential for life-saving impact that makes the journey to full autonomy unquestionably worth the effort.

As we navigate this complex journey towards fully autonomous vehicles, it is crucial to recognize that the benefits extend not only to saving lives but also to minimizing congestion, reducing emissions, and enhancing mobility for individuals with disabilities or limited access to transportation.

Technically, an autonomous vehicle needs the same main parts as a human needs to drive: The capacity to perceive the environment (perception), formulate decisions regarding actions to be taken (decision-making or planning), and subsequently execute those chosen actions (action).

In terms of perception, autonomous vehicles use advanced object detection algorithms that carefully examine data from various sensors placed strategically on the vehicle. This complex system helps the vehicle understand its surroundings accurately.

Regarding action, the vehicle's capability to turn plans into real moves, like steering and using the pedals, relies on a good grasp of how the vehicle behaves. This means turning what it wants to do into actual actions, ensuring safe and efficient navigation.

While the areas of perception and action continually evolve and can indeed be optimized further, the biggest challenge for autonomous vehicles are decision-making and planning. This complex process includes analyzing lots of data, predicting changing conditions, and deciding on a series of actions for safety, following traffic rules, comfort and efficiency.

In essence, while advancements in perception and action are crucial, the core strength of autonomous vehicles is their ability to handle complex planning and make quick decisions, resembling the judgment of human drivers. This aspect represents the main challenge to achieving fully autonomous vehicles.

Therefore, this project will center its efforts on delving deep into the potential of leveraging reinforcement learning techniques during the crucial planning phase, aiming to enhance decision-making processes.

1.2 Goals

- To understand an existing AV system and finding the replaceable modules.
- To develop a reinforcement learning model suitable for autonomous driving, minimizing the model's input.

1.3 Sustainability, diversity and ethics

Autonomous driving and Advanced Driver Assistance Systems (ADAS) aim to enhance road safety. However, from a sustainability perspective, the development of sophisticated artificial intelligence models and the simulation of real-world environments require significant computational resources. This often involves the extensive use of GPUs over prolonged periods, resulting in substantial electrical energy consumption and subsequent carbon emissions, contributing significantly to the overall carbon footprint.

The real-world testing and validation of these trained models further escalate energy consumption. Testing and validation are crucial to ensure the safety of the models, so there are limitations in reducing this energy usage. Nevertheless, optimizing the training phase becomes pivotal in reducing the overall carbon footprint of the process.

In this project, the carbon emissions generated by the training and simulation will be measured, trying to find metrics for its impact.

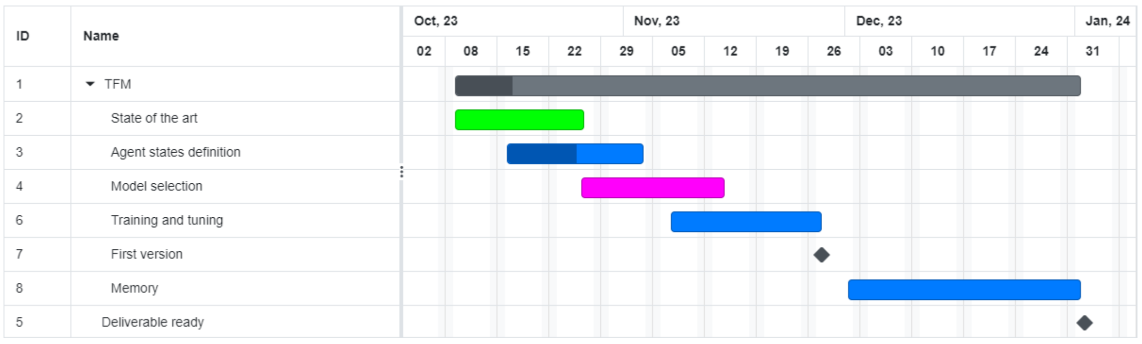
1.4 Methodology and planning

Initially, a review of the current state of the art will be conducted, encompassing both in the domain of autonomous driving and the specific application of reinforcement learning within the autonomous driving context.

Simultaneously, particular emphasis will be placed on the critical aspect of defining the states in the context of reinforcement learning. The definition of states plays an integral role in shaping the learning process of the autonomous system. Hence, it is key to define a proper state representation that the reinforcement learning agent will encounter during its training and operational phases.

The next step involves selecting the appropriate model. Various models will be tested, concluding with the training and parameter tuning of the chosen model.

Once the initial version of the model is prepared, the memory of the project will be written. In the meantime, potential improvements to the model can be explored and executed.



ID	Name
1	TFM
2	State of the art
3	Agent states definition
4	Model selection
6	Training and tuning
7	First version
8	Memory
5	Deliverable ready

Figure 1.1: Gantt diagram

Chapter 2

State of the art

2.1 Autonomous driving

Within the domain of autonomous vehicles, just as a human driver relies on a set of fundamental components to navigate the road, self-driving vehicles also depend on those essential elements. They must have the ability to perceive their surroundings (perception), make calculated decisions about their next moves (decision-making or planning), and efficiently translate those choices into tangible actions (action). These three elements can also be decomposed into more precise characteristics:

2.1.1 Perception

Autonomous vehicles use various sensors, including LiDAR, radar, cameras, and ultrasonic sensors to perceive their surroundings. Advances in sensor technology have greatly improved perception capabilities.

- **LiDAR:** LiDAR sensors emit laser pulses and measure the time it takes for the light to bounce back. Algorithms process this data to create a 3D point cloud of the vehicle's surroundings, allowing the car to detect objects and obstacles with high precision.
- **Radar:** Radar sensors use radio waves to detect objects and their velocities. Signal processing algorithms are used to filter and analyze radar data to identify objects and track their movements.
- **Cameras:** Vision-based perception is vital for recognizing objects and interpreting traffic signs and signals. Convolutional Neural Networks (CNNs) and other deep learning algorithms are used for tasks like object detection, image segmentation, and optical flow estimation.

- **Ultrasonic Sensors:** These sensors are used for short-range perception and parking assistance. They work by emitting sound waves and measuring the time it takes for the sound to bounce back. Algorithms process this data to detect nearby objects.

Each of these sensors, along with their respective algorithms, may have noise and errors (covariance). In order to minimize such covariance, sensor fusion (or data fusion) algorithms are applied. This technique combines data from various sensors (e.g., Lidar, radar, cameras) to improve the accuracy and reliability of perception. Kalman filters and more advanced methods like the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) are used for sensor fusion.

2.1.2 Planning

- **Localization and Mapping:** High-Definition maps, GPS, and simultaneous localization and mapping (SLAM) techniques are crucial for autonomous vehicles to understand their location within the environment.
  - **HD (High-Definition) Maps:** HD maps provide a highly detailed and precise representation of the environment. They typically include information about lane boundaries, road curvature, traffic signs, traffic signals, lane changes, and other road attributes. HD maps are created through a combination of surveying techniques, GPS data, and data from sensors on autonomous vehicles. These maps serve as a reference for the vehicle's position and help it make decisions, especially in complex urban environments.
  - **SLAM (Simultaneous Localization and Mapping) Maps:** SLAM is a technique used to build and update maps of an environment in real-time while also tracking the vehicle's position within that map. SLAM is particularly useful in scenarios where there are no existing HD maps, such as off-road or dynamic environments. SLAM systems use a combination of sensors, such as Lidar, cameras, and IMUs (Inertial Measurement Units), to continuously update the map and determine the vehicle's location.
- **Decision-making and Planning:** Path planning algorithms and controllers are responsible for making driving decisions, such as navigating obstacles and staying within lanes. These are the most common approaches:
  - **Rule-Based Systems:** Many autonomous vehicles use rule-based decision systems, where predefined rules are applied to make driving decisions. These rules cover

standard traffic regulations and common driving behaviors. However, these systems can be limited in handling complex and unforeseen situations.

- **Reinforcement Learning (RL):** RL has gained popularity in decision-making for autonomous cars. Agents learn optimal driving policies through trial and error in simulated environments.
- **Imitation Learning:** Imitation learning is used to mimic human driving behavior. Algorithms are trained on human driving data to imitate the decision-making process. This approach helps vehicles handle real-world scenarios effectively.
- **Behavior Planning:** Behavior planning involves high-level decision-making, where the vehicle decides how to navigate, merge, or overtake other vehicles. Techniques such as finite state machines (FSMs) and decision trees have been used for behavior planning.

2.1.3 Action

Once a decision has to be executed, different pedal and steering wheel commands are sent, taking into account the vehicle's dynamics and other vehicle particularities.

2.2 Reinforcement learning for autonomous driving

Reinforcement learning is a subset of machine learning where agents learn to make sequential decisions by interacting with an environment. In the context of autonomous driving, RL can be used for various tasks:

- **End-to-End Driving Control:** Using RL to directly learn driving policies, bypassing the need for traditional rule-based systems.
- **Trajectory Planning:** RL algorithms can plan optimal trajectories in dynamic environments.
- **Behavior Prediction:** Predicting the behavior of other road users is crucial for decision-making.

2.2.1 Simulators

An essential aspect to keep in mind is that training autonomous vehicles in actual on-road situations is not viable, as it poses significant safety risks. Instead, self-driving cars undergo training within a controlled and risk-free simulator environment.

Some examples of simulators are:

- **CARLA (Car Learning to Act):** An open-source simulator developed by Intel and used for training autonomous driving agents. CARLA provides a realistic urban environment for testing various scenarios.
- **AirSim:** Developed by Microsoft, AirSim is an open-source simulator for training autonomous systems. It offers a wide range of environments, including urban, rural, and industrial settings.
- **Gazebo:** Gazebo is a widely used open-source robotics simulator that can be adapted for autonomous vehicle training. It provides realistic physics and sensor simulation.
- **MATLAB/Simulink:** MATLAB provides simulation tools that can be used for training and testing autonomous vehicle control algorithms.
- **DriveConstellation:** Developed by NVIDIA, DriveConstellation is a cloud-based simulator that uses powerful GPUs to test self-driving car algorithms and perception systems.

2.2.2 Models

2.2.2.1 Partially Observable Markov Decision Process (POMDP)

A Partially Observable Markov Decision Process is a framework that extends Markov Decision Processes to handle situations where decision-makers have incomplete and noisy observations, making it a valuable tool in solving complex decision-making problems in uncertain environments.

2.2.2.2 Deep Q-learning

Deep Q-Network (2) is a reinforcement learning algorithm that combines Q-Learning with deep neural networks to let RL work for complex, high-dimensional environments, where the number of states cannot be managed with a table.

Chapter 3

Base system

3.1 AV system

An existing Autonomous Vehicle (AV) model will be used as a base for this project. This chapter will detail the model's structure and components. The implementation of this AV relies on the Robot Operating System (ROS).

3.1.1 ROS

The Robot Operating System (ROS) is a robust, open-source middleware framework designed explicitly for robotics. Functioning as a comprehensive toolset and communication infrastructure, ROS enables the development, coordination, and management of complex robotic systems.

At its core, ROS offers a modular architecture that facilitates the creation of interconnected software components called nodes. These nodes execute specific tasks, from sensor data processing to high-level decision-making algorithms, and communicate consistently through a publisher-subscriber messaging system (the elements where the data is published are called "topics") or via services.

One of ROS's fundamental strengths lies in its abstraction of hardware complexities. It provides a standardized interface for interfacing with various sensors, actuators, and hardware components.

Moreover, ROS thrives on an expansive and engaged community of developers and researchers who contribute to its continual evolution. This collaborative ecosystem encourages the sharing of code, libraries, and best practices, accelerating innovation and advancements in robotics technology.

3.1.2 RViz

RViz is a powerful 3D visualization tool within the Robot Operating System (ROS) framework. It offers a graphical interface to visualize and interact with various data generated by robots, including sensor data, robot models, trajectories, and more. This tool helps developers and users in comprehending and debugging the behavior of robotic systems through an intuitive and customizable visual representation.

In this project, it is used to visualize both the map and the position of the ego vehicle within that map.

3.1.3 Software structure

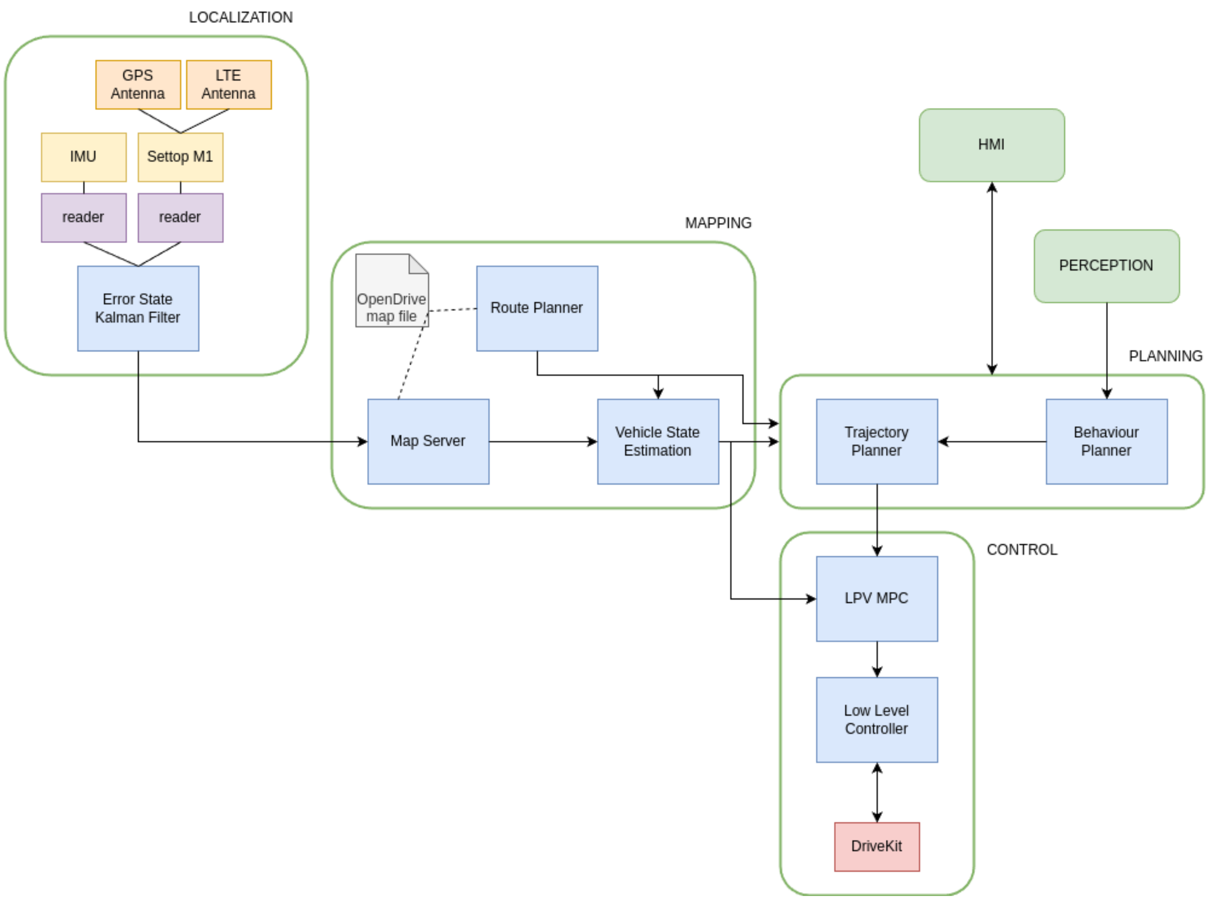


Figure 3.1: Existing AV model Software structure. Source: Author's elaboration

As shown in Figure 3.1, the model is composed of four different software modules: Localization, mapping, planning and control.

3.1.3.1 Localization

The localization module provides a solution to the positioning of the vehicle based on GNSS (Global Navigation Satellite System). The Kalman filter algorithm provides position stability by minimizing jumps in position that can occur during direct reading of GNSS data. To do so, it uses an IMU (inertial measurement unit).

For this project, this module will be executed on simulation, meaning that the global position of the vehicle will be directly simulated.

3.1.3.2 Mapping

The global position of the vehicle is sent to the mapping module. The map server transforms the global position into different Cartesian coordinate systems referenced to the OpenDrive map that has been configured. In addition, the map server can be queried to obtain different information such as road signs, road geometry, etc.

Another important component within the mapping module is the route planner. The route planner receives the current vehicle state and the target destination. It analyses the map along with the input data and chose the best route of roads and lanes to reach the target destination. The best route is the shortest path in terms of distance.

The vehicle state estimator has the objective of providing the position, orientation, accelerations, etc. of the ego vehicle as reliable as possible. On the real-world environment, this data is gathered directly from the vehicle's CAN (corrected with the IMU). For this project, it will be the result of the simulation.

3.1.3.3 Planning

The planning module has two main components, the decision or behaviour planner and the trajectory planner. The behaviour planner filters the objects received by the perception system evaluating those that can influence the current route of the ego vehicle. Taking into account the traffic signals and the surrounding vehicles and VRUs, the behaviour planner controls the trajectory planner. The trajectory planner generates and discretizes splines for position and velocity profiles. Following several cost functions and restrictions, it searches for the best trajectory.

3.1.3.4 Control

The control module is divided into two components. The high-level control that receives the current state of the ego vehicle and the trajectory and calculates the action commands necessary to carry out such trajectory. Then comes the low-level control that receives these actuation commands and adapts them to the specific platform being controlled. The high-level controller is based on a linear parameter varying MPC (model predictive control) plus Stanley formulas for slow velocities. On the other hand, the low level controller implements PIDs and lookup tables.

Chapter 4

What to replace

The aim of this project is to replace only the decision-making modules, while preserving the rest of the system's components, so that it could be easily integrated on the current car setup.

The following approaches have been studied:

4.1 First approach: replace only path planning

The first approach was to only replace the path planning component. It means that the new component should take as input the vehicle state estimation, the suggested route from the route planner and the perception results, and generating a trajectory as output for the LVP MPC.

In a first loop, for the shake of simplicity, the route and the perception obstacles were not considered as input, meaning that the input was only the vehicle state.

4.2 Second approach: replace path planning and LVP MPC

On the second approach, both the path planning and the LVP MPC components were substituted by the new component. It means that the new component takes as input the vehicle state, and returns as output an acceleration value and a steering angle, which are sent to the low level controller.

Chapter 5

How to interconnect

5.1 ROS communication

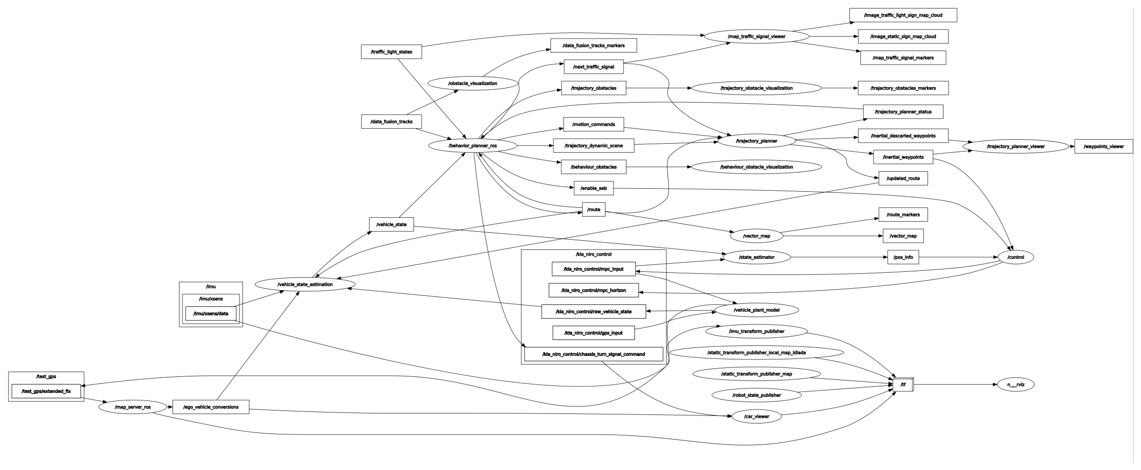

/traffic_light_states
/data_fusion_tracks_markers
/map_traffic_signal_viewer
/image_traffic_light_sign_map_cloud
/image_static_sign_map_cloud
/map_traffic_signal_markers
/next_traffic_signal
/obstacle_visualization
/trajectory_obstacles
/trajectory_obstacle_visualization
/trajectory_obstacles_markers
/trajectory_planner_status
/data_fusion_tracks
/motion_commands
/trajectory_planner
/inertial_descarted_waypoints
/behavior_planner_ros
/trajectory_dynamic_scene
/inertial_waypoints
/trajectory_planner_viewer
/waypoints_viewer
/behaviour_obstacles
/behaviour_obstacle_visualization
/updated_route
/enable_aeb
/route
/route_markers
/vehicle_state
/vector_map
/vector_map
/kia_niro_control
/state_estimator
/pos_info
/control
/kia_niro_control/mpc_input
/imu
/imu/xsens
/imu/xsens/data
/vehicle_state_estimation
/kia_niro_control/mpc_horizon
/kia_niro_control/raw_vehicle_state
/vehicle_plant_model
/kia_niro_control/gps_input
/imu_transform_publisher
/kia_niro_control/chassis_turn_signal_command
/static_transform_publisher_local_map_idiada
/test_gps
/test_gps/extended_fix
/static_transform_publisher_map
/tf
n__rviz
/robot_state_publisher
/map_server_ros
/ego_vehicle_conversions
/car_viewer

Figure 5.1: Existing AV model. Simulation ROS communication. Source: Generated using ROS package rqt_graph

In the figure 5.1 the ROS communication graph is represented. Circular elements denote the nodes, while squared elements represent the topics.

For the first approach, the nodes that have to be replaced are 'trajectory_planner' and 'behavior_planner_ros'. The new node (carEnv) subscribes to the 'vehicle_state' topic and provides 'inertial_waypoints' as output. The next figure shows the resulting communication graph for the first approach:

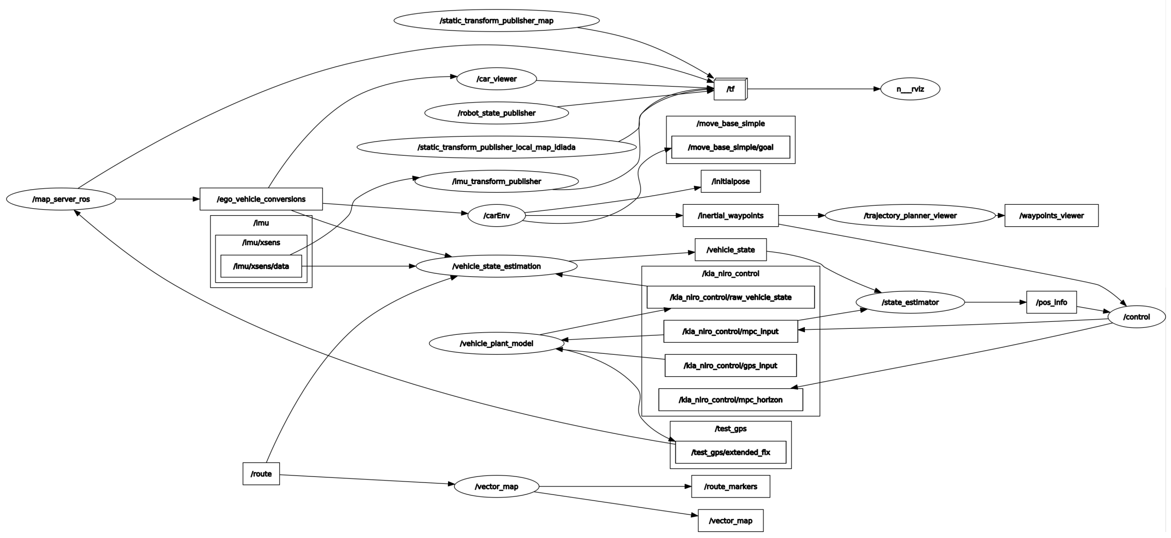


Figure 5.2: First approach. Simulation ROS communication. Source: Generated using ROS package rqt_graph

For the second approach, the node 'control' (which contains the LVP MPC) is also substituted by the new node 'carEnv'. Therefore, the node 'carEnv' is the node in charge of publishing messages on the topic 'kia_niro_control/mpc_input'.

The messages published on the topic 'kia_niro_control/mpc_input' have the following structure:

- left_steer_angle_command, of type float, which is the steering angle in radians.
- acceleration_command, of type float, acceleration in m/s^2
- brake_deceleration_command, deceleration in m/s^2

Acceleration and deceleration commands are separated in case that an AV needs to apply both (e.g. holding on hill to avoid roll-back during start).

The next figure shows the resulting communication graph for the second approach:

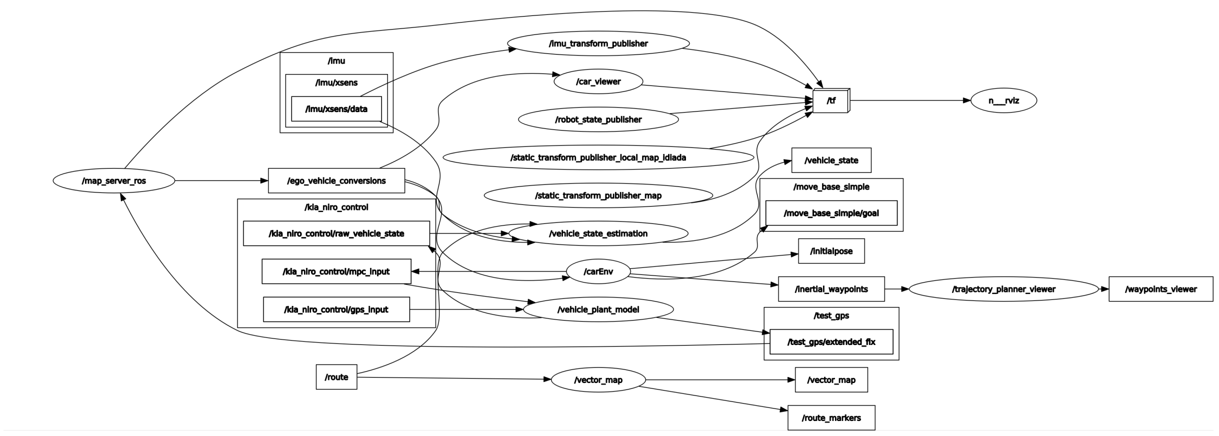


Figure 5.3: Second approach. Simulation ROS communication. Source: Generated using ROS package rqt_graph

5.2 CARLA

The initial plan involved utilizing the CARLA simulator to train the model. CARLA offers collision and lane invasion events, which were intended for integration into the reward system. However, the final implementation no longer incorporates these events, making CARLA unnecessary for training purposes. Nevertheless, CARLA serves the purpose of providing a visual representation of the entire process.

To integrate CARLA into the complete pipeline, the CARLA ROS bridge (3) is utilized. CARLA ROS bridge enables the possibility of communicating CARLA with ROS, and vice versa. The information from the CARLA server is translated to ROS topics and the messages sent between nodes in ROS get translated to commands to be applied in CARLA.

The next figure represents the ROS communication with CARLA integrated:

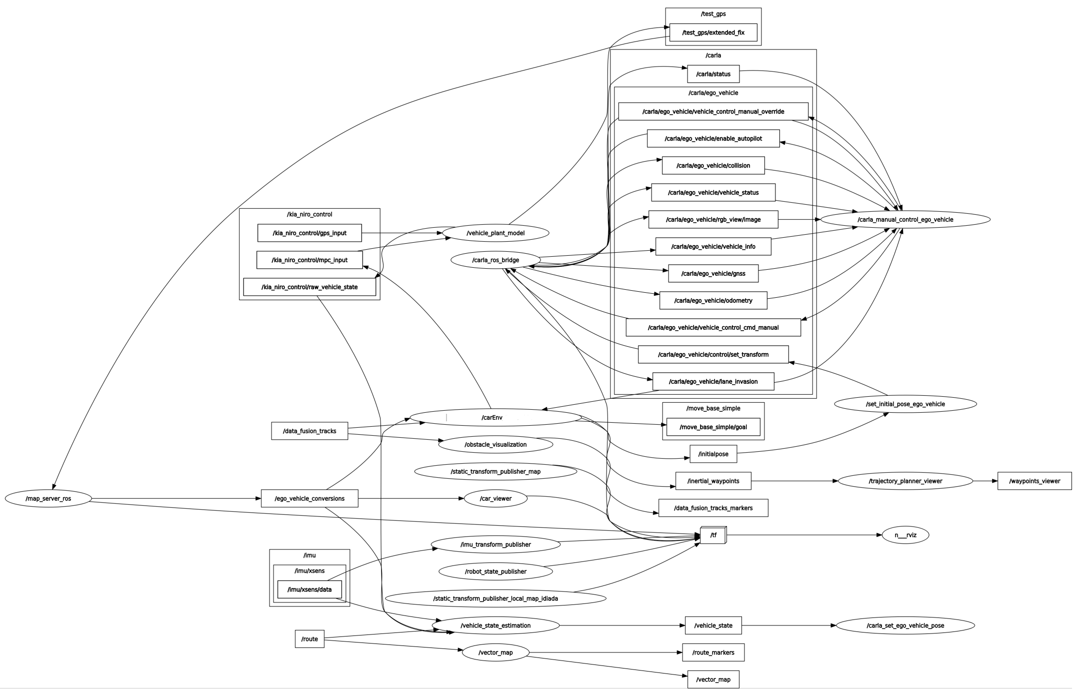

/test_gps
/test_gps/extended_fix
/carla
/carla/status
/carla/ego_vehicle
/carla/ego_vehicle/vehicle_control_manual_override
/carla/ego_vehicle/enable_autopilot
/carla/ego_vehicle/collision
/carla/ego_vehicle/vehicle_status
/carla/ego_vehicle/rgb_view/image
/carla/ego_vehicle/vehicle_info
/carla/ego_vehicle/gnss
/carla/ego_vehicle/odometry
/carla/ego_vehicle/vehicle_control_cmd_manual
/carla/ego_vehicle/control/set_transform
/carla/ego_vehicle/lane_invasion
/carla_manual_control_ego_vehicle
/kia_niro_control
/kia_niro_control/gps_input
/kia_niro_control/mpc_input
/kia_niro_control/raw_vehicle_state
/vehicle_plant_model
/carla_ros_bridge
/set_initial_pose_ego_vehicle
/move_base_simple
/move_base_simple/goal
/carEnv
/data_fusion_tracks
/obstacle_visualization
/initialpose
/static_transform_publisher_map
/inertial_waypoints
/trajectory_planner_viewer
/waypoints_viewer
/map_server_ros
/ego_vehicle_conversions
/car_viewer
/data_fusion_tracks_markers
/tf
n__rviz
/imu_transform_publisher
/imu
/imu/xsens
/imu/xsens/data
/robot_state_publisher
/static_transform_publisher_local_map_idiada
/vehicle_state_estimation
/vehicle_state
/carla_set_ego_vehicle_pose
/route
/vector_map
/route_markers
/vector_map

Figure 5.4: Simulation ROS communication with CARLA integrated. Source: Generated using ROS package rqt_graph

As shown on Figure 5.4, the vehicle state topic, 'vehicle_state', is taken by the node 'carla_set_ego_vehicle_pose', which communicates with the CARLA server, modifying the ego vehicle pose on CARLA.

Chapter 6

Implementation

6.1 First ideas

The input for the linear parameter varying model predictive control (LVP MPC) is a ROS custom message, with the following fields:

- waypointList, which is an array of geometry_msgs/Pose2D.
- curvatureList, which is an array of floats.
- velocityList, which is an array of floats.

Therefore, the input for the LVP MPC is a discretized trajectory. The first idea was to get splines, specifically, cubic Bézier curves (4) as output.

6.1.1 Bézier curves

The cubic Bézier curves are represented as follows:

$$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3, t \in [0,1]$$

For discretization, since a ROS message of type geometry_msgs/Pose2D is composed by (x, y, θ), the Bézier curves for x and y have to be computed individually, and the value of θ has to be computed individually, as $\theta = arctan2(x - x_0, y - y_0)$, where (x_0, y_0) is the previous point.

6.1.2 Curvature

The curvature of a Bézier curve is given by:

$$k(t) = \frac{dx * ddy - ddx * dy}{\sqrt{(dx * dx + dy * dy)^3}}$$

Where dx, dy, dxx, dyy are the first and second derivatives (respectively), of the x and y Bézier curves at point $t \in [0, 1]$

6.1.3 Velocity

The velocity list was the result of applying a linear acceleration.

6.1.4 RL model and its output

In order to be able to generate the defined trajectories, the output should be composed by: P_1, P_2, P_3 and acceleration. As the points are in $\mathbb{R}^2$, the reinforcement learning model's output should be an array of seven real numbers. For having rational numbers as output, the action space should be continuous, meaning that the model has to be a policy based reinforcement learning method. Considering the dimension of the input considered for the model, 15 numbers (it will be explained in following chapters), the chances of training successfully a model with such big action space are so small that this approach was discarded.

Finally, the selected reinforcement learning model is Deep Q-Network (DQN).

6.2 RL model

The Deep Q-Network (DQN) represents a significant leap in reinforcement learning by merging the capabilities of deep neural networks with the foundational principles of Q-learning, a classical algorithm in reinforcement learning. At its essence, DQN aims to approximate the Q-function, a critical element in reinforcement learning that estimates the cumulative future rewards an agent expects by taking specific actions in particular states within an environment.

One of its biggest strengths is its ability to handle high-dimensional state spaces, enabling the learning process directly from raw, unprocessed data—such as pixel values from images—without the need for manual feature extraction. By leveraging deep neural networks, especially convolutional neural networks (CNNs) for their prowess in visual data processing, DQN navigates complex environments that would overwhelm conventional Q-learning approaches due to the vast number of possible states and actions.

The architecture of a DQN typically comprises multiple layers of neurons, often CNN layers followed by fully connected layers, facilitating the approximation of the Q-function. Training DQN involves iteratively updating the network's parameters to minimize the discrepancy between predicted and target Q-values for various state-action pairs encountered during exploration.

However, two fundamental strategies set DQN apart and contribute significantly to its success: experience replay and the use of target networks. Experience replay involves storing the agent's experiences (comprising state, action, reward, and next state transitions) in a replay memory buffer. During training, these experiences are randomly sampled, breaking temporal correlations and allowing for more efficient learning from past experiences. This technique significantly improves sample efficiency, stabilizes learning, and promotes better exploration.

Additionally, DQN employs the concept of target networks, which entails the creation of two neural network instances: the primary network and the target network. The target network is a copy of the primary network that maintains fixed Q-value targets for a certain period before periodically updating its parameters. This approach helps stabilize the learning process by providing more consistent and less volatile Q-value estimates, preventing divergence or oscillation during training.

In summary, the Deep Q-Network (DQN) represents a groundbreaking fusion of Q-learning with deep neural networks, enabling efficient learning in high-dimensional state spaces. By incorporating techniques like experience replay and target networks, DQN achieves remarkable stability, sample efficiency, and the capacity to handle complex environments, making it a foundational model in the domain of deep reinforcement learning.

6.3 First approach

Once the idea of generating splines as output was dismissed, an alternative approach was required. Considering that the model's output needs to be a single integer value, and that the LVP MPC requires a trajectory as input, the following strategy was adopted:

We consider four possible actions:

- Accelerate: A positive longitudinal acceleration is applied.
- Brake: A negative longitudinal acceleration is applied.

- Turn right: A 90º steering angle is applied.
- Turn left: A -90º steering angle is applied.

In order to translate these actions to trajectories, the following strategy was adopted:

For breaking and acceleration actions, the trajectory is straight in both cases. From the 'vehicle_state' message, the position (x, y) and the heading angle of the ego vehicle are available. A distance d is added to the current point, so that the point $p' = p + d$ will be the end of the trajectory. For representing a cubic Bézier curve, four points are needed: P_0, P_1, P_2 and P_3. P_0 is the starting point p, and P_3 is the ending point p'. In order to get P_1 and P_2, $d/4$ and $d/2$ are added respectively to P_0. I.e., $P_1 = P_0 + d/4$ and $P_2 = P_0 + d/2$. Since the working real space is $\mathbb{R}^2$, it is needed to decompose the increment d into its x and y components. In other words, a translation of distance d is applied into the direction θ, the heading angle. Hence, the general representation of the points is:

$$P_i = (P_{0x} + \frac{d\cos\theta}{j}, P_{0y} + \frac{d\sin\theta}{j}), \forall(i,j) \in \{(1,4),(2,2),(3,1)\}$$

For turning actions, the points needed are as shown on the following Figure:

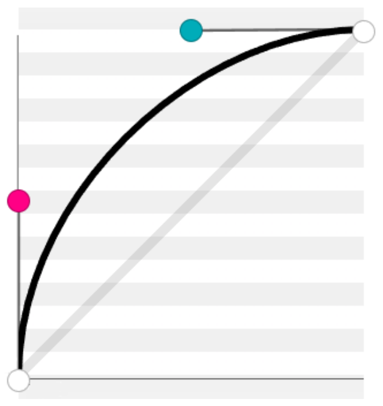


Figure 6.1: Turning trajectory. Source: Author's elaboration

If the white left bottom point is considered as P_0, and the white right top point as P_3, the figure represents a turn right action. In that case, the red point is P_1 and the blue point is P_2. Alternatively, if the white right top point is considered as P_0, the blue point would be P_1, the red point would be P_2 and the white left bottom point would be P_3.

For the right turn, the points would be computed iteratively as follows:

$$P_1 = (P_{0x} + \frac{d\cos\theta}{2}, P_{0y} + \frac{d\sin\theta}{2})$$

$$P_3 = (P_{0x} + d\cos(\theta - \frac{\pi}{4}), P_{0y} + d\sin(\theta - \frac{\pi}{4}))$$

$$P_2 = (P_{3x} + \frac{d\cos(\theta + \frac{\pi}{2})}{2}, P_{3y} + \frac{d\sin d(\theta + \frac{\pi}{2})}{2})$$

Basically, to get P_1 what is needed to do is to increment $\frac{d}{2}$ as did for the straight trajectories. To get P_3 an increment of distance d is needed but within a $45^{\underline{o}}$ degrees rotation. Finally, to get P_2, an increment of $\frac{d}{2}$ to the point P_3 is required, within a $90^{\underline{o}}$ rotation (to the opposite direction to the rotation applied to get P_3).

Graphically:

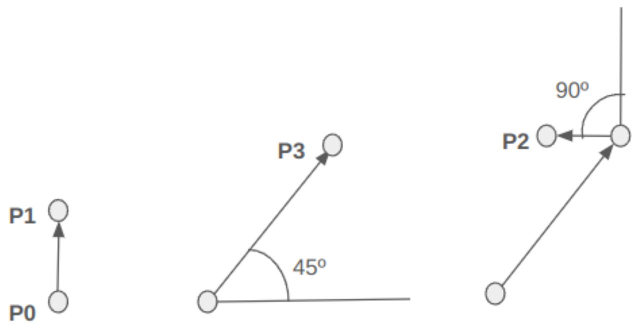


Figure 6.2: Right turn Bézier curve points generation. Source: Author's elaboration

For the left turn, the logic is the same but the rotation angles are the opposite:

$$P_1 = (P_{0x} + \frac{d\cos\theta}{2}, P_{0y} + \frac{d\sin\theta}{2})$$

$$P_3 = (P_{0x} + d\cos(\theta + \frac{\pi}{4}), P_{0y} + d\sin(\theta + \frac{\pi}{4}))$$

$$P_2 = (P_{3x} + \frac{d\cos(\theta - \frac{\pi}{2})}{2}, P_{3y} + \frac{d\sin d(\theta - \frac{\pi}{2})}{2})$$

Graphically:

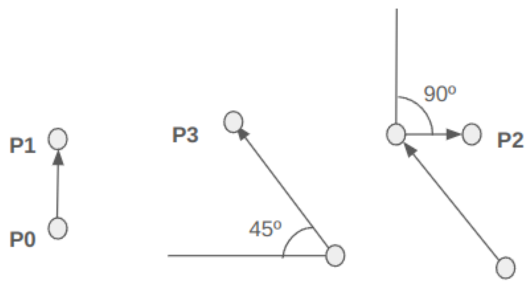


Figure 6.3: Left turn Bézier curve points generation. Source: Author's elaboration

6.3.1 Reason to discard

As explained, the generated output with this approach was a trajectory, which was sent to the LVP MPC, who computed the throttle, brake and steering angle needed to follow such trajectory. In reinforcement learning, it is needed to quantify how "good or bad" an action is, meaning that an evaluation of the result of applying an action is required. With this approach, if the effects a certain action has on the system are to be noticed, the frequency of intervention (how fast an action is taken) needs to be low, so that the system has time to apply the action, and once it has been applied, the results could be evaluated. In an autonomous driving system, the actuation frequency is critical. For example, driving at 50 kph, a latency of 500 ms implies that the vehicle is moving almost 7m without control. At the testing section, this will be analyzed in depth with examples.

6.4 Second approach

On the second approach, the LVP MPC is not used. In that case, instead of publishing trajectories, what is needed to be published are the throttle, brake and steering angle instructions. As defined on section 5.1, the message 'kia_niro_control/mpc_input' is composed by three different fields: 'left_steer_angle_command', 'acceleration_command' and 'brake_deceleration_command'. Considering that, the different actions have been defined as:

6.4.1 Accelerate

- left_steer_angle_command = 0

- acceleration_command = 2
- brake_deceleration_command = 0

6.4.2 Brake

- left_steer_angle_command = 0
- acceleration_command = 0
- brake_deceleration_command = 2

6.4.3 Right turn

- left_steer_angle_command = $-\pi/2$
- acceleration_command = 0
- brake_deceleration_command = 0

6.4.4 Left turn

- left_steer_angle_command = $\pi/2$
- acceleration_command = 0
- brake_deceleration_command = 0

Note that the trajectories defined on the first approach are still implemented and published on this second approach. They are used for visualization, so that it is easier to understand the behavior of the vehicle. The LVP MPC is not used, so the trajectories are not used for anything else than visualization.

6.5 State space

The state space is defined by the 'vehicle_state' message. The 'vehicle_state' defines the ego vehicle position within an HD map, its speed and accelerations:

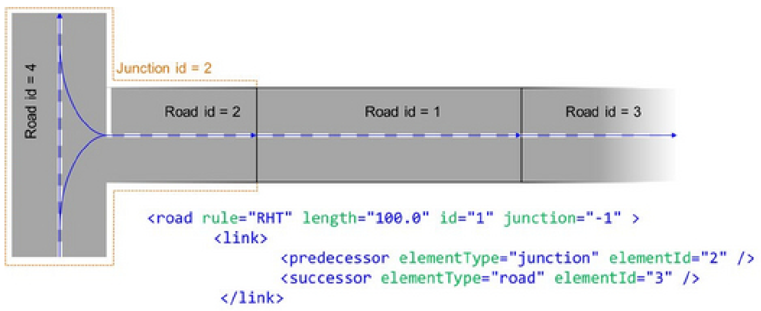
- x, coordinate x on the map
- y, coordinate y on the map

- z, coordinate z on the map
- heading, heading of the vehicle
- k, curvature of the lane at the current position
- track_id, road sector (road_id)
- s, longitudinal position within the road sector
- lane_id, lane index within the road sector
- velocity, module of the speed
- longitudinal_velocity, longitudinal speed
- lateral_velocity, lateral speed
- angular_velocity, angular speed
- acceleration, module of the acceleration
- longitudinal_acceleration, longitudinal acceleration
- lateral_acceleration, lateral acceleration

6.5.1 HD Maps

The ASAM (Association for Standardisation of Automation and Measuring Systems) Open-DRIVE format (5) defines the HD maps used within this project.

At the Figure 6.4 it is shown how a road is split into different sectors. Each one of those sectors has its own 'road_id' (identified as 'track_id' on the ROS messages).



```
<road rule="RHT" length="100.0" id="1" junction="-1" >
    <link>
        <predecessor elementType="junction" elementId="2" />
        <successor elementType="road" elementId="3" />
    </link>
```

Figure 6.4: Road sectorization. Source: ASAM OpenDRIVE

When the ego vehicle is in one of those sections, it is located within the sector with the parameters s (longitudinal position) and t (lateral position), as represented in Figure 5.4.

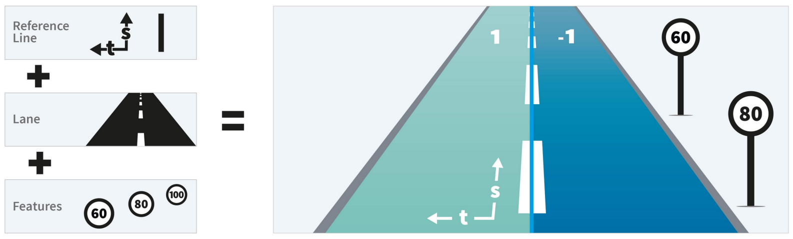


Figure 6.5: Road sector localization. Source: ASAM OpenDRIVE

Finally, it is important to mention the lane identification system. For each sector, the right lanes have negative id. The center most lanes have the index 1 or -1 depending on the side, the second line has index 2 or -2, and so on. The orientation of the lane is identified by the parameter s. The sector starts at $s = 0$, and increases its value towards the orientation of the road sector.

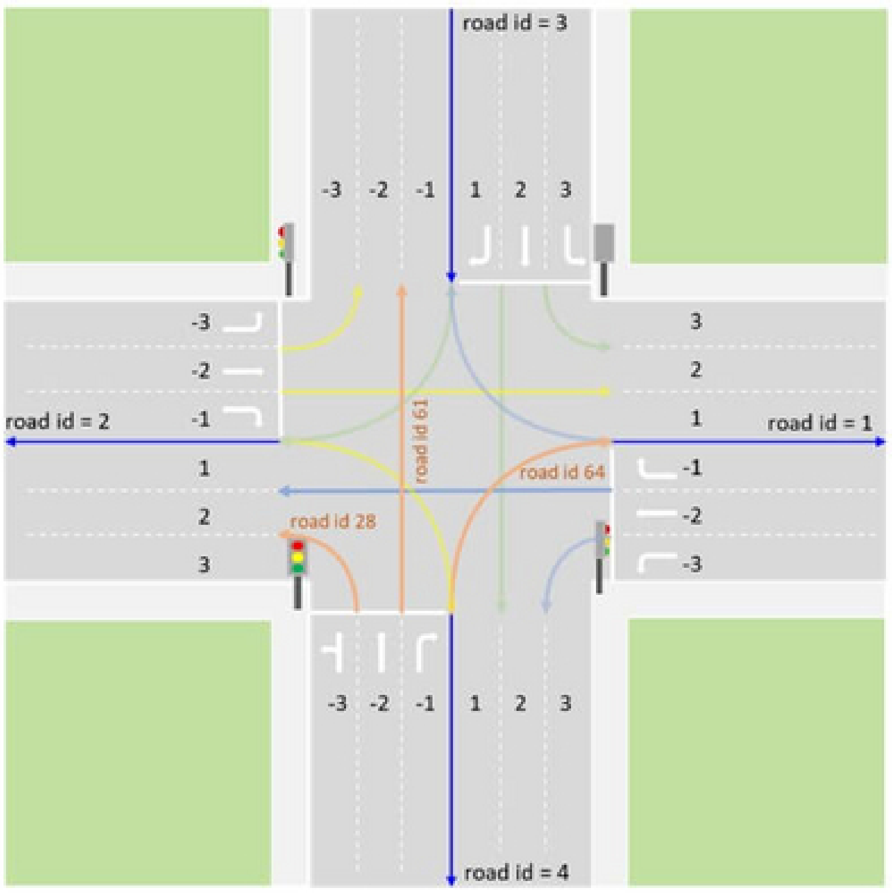


Figure 6.6: Intersection. Lane identification. Source: ASAM OpenDRIVE

One important aspect to mention, is that the ego vehicle can be at more than one sector at once, for example, at an intersection, as shown at Figure 6.6.

6.6 Reward system

6.6.1 First ideas

The initial plan involved utilizing CARLA's collision and lane invasion events. Every time there is a collision, the episode ends with a high negative reward, while lane invasions would yield a negative reward for the corresponding action. However, after conducting tests, this approach was discarded for several reasons.

The reasons for discarding using the CARLA events are: Given that the training only involves the ego vehicle without interactions with other vehicles, the only possible source of crash is the ego vehicle going out of the track, so being able to identify if the car goes out of the track has the same meaning as a collision event, in fact, going out of the road is more restrictive, which implies a homogeneous treatment of the events.

On the other hand, the challenge with the lane invasion event lies in its limited capability to discern whether the car has transitioned from the correct lane to an incorrect one or vice versa. It means that if the training is done with this event rewarding negatively, the car would learn that once it has crossed to the incorrect lane, it is preferable to remain there rather than returning to the correct lane.

Moreover, apart from these concerns, utilizing CARLA simulations demands high GPU usage, which is no longer essential for the training process. By eliminating their use in training, there is a significant reduction in energy consumption during the training phase.

6.6.2 Final reward system

- If the ego vehicle goes outside the road, reward = -100, and end of episode.
- If the ego vehicle stays stopped for more than 200 iterations, reward = -50, and end of episode. (Negative reward, but half the penalty than crashing)
- If speed lower than 1 km/h, reward = -3.
- If the ego vehicle drives on an incorrect lane, reward = -4. (More penalty than driving slowly)

- If the ego vehicle drives on the correct lane, but close to the edge, reward = -2. (Penalty, but not so big as low speed or driving on an incorrect lane)
- If the ego vehicle drives on the correct lane, reward = (speed / offset) / 5, where offset is the distance to the center of the lane. To avoid extremely big values, the offset is limited by 0.1.

Concurrent rewards are added. For example, stopping on an incorrect lane has a penalty of 7 points.

Two key points on the reward system are: how to identify if the ego vehicle goes outside the road, and how to determine if the ego vehicle is driving on a correct lane.

6.6.3 Going out of the road identification

In order to identify whether the ego vehicle is on track or not, the ROS topic 'ego_vehicle_conversions' is used. As shown at Figure 5.3, the topic 'ego_vehicle_conversions' is published by the node 'map_server_ros', which gets the ego vehicle's GPS position, and publishes ROS messages with the following structure:

- geo_coordinate, of custom type GeoCoordinate, which are the geographical coordinates.
- gps_velocity, of type float, which is the velocity according to the GPS.
- inertial_coordinate, of custom type InertialCoordinate, which are the coordinates with respect to the map.
- inertial_heading, of type float, which is the heading with respect to the map orientation.
- lane_coordinates, a list of elements of custom type LaneCoordinate, which is the information regarding the road sector, or 'road_id' as defined in section 6.5.1.
- fixed_inertial_height, of type float, which is the height with respect to the map.

The important part of this message is the element 'lane_coordinates'. In this part of the message, all the sectors where the ego vehicle is located are identified. If this list is empty, it means that the ego vehicle is not in any road sector, hence, the ego vehicle is out of track.

6.6.4 Correct/incorrect lane identification

The best way to identify if the lane the ego vehicle is driving is the correct one would have been to obtain the lane heading and compare it with the vehicle's heading. Unfortunately, such information is unavailable. Attempts were made to extract this heading information from the map data, with unsuccessful results. Consequently, the final solution is as follows:

As previously mentioned, on the topic 'ego_vehicle_conversions', it is available the information regarding the road sectors where the ego vehicle is located. Such information is represented in a list of elements of type LaneCoordinate. The type LaneCoordinate has the following fields:

- track_id, of type integer, the road sector. (As explained in section 6.5.1.)
- lane_id, of type integer, the lane id. (As explained in section 6.5.1.)
- s, of type float, the lane longitudinal coordinate. (As explained in section 6.5.1.)
- t, of type float, the lane lateral coordinate. (As explained in section 6.5.1.)
- offset, of type float, the offset from the center of the lane.
- h, of type float, the height regarding the map.

With this information, it is possible to determine whether the ego vehicle is positioned within a correct or incorrect lane. Considering the car cannot drive in reverse mode:

The previous 's' value is stored. If the current 's' value is greater than the previous 's' value, it means that the ego vehicle is driving following the orientation shown in Figure 6.5, thus, the correct lanes have negative id. Consequently, if the 'lane_id' value is negative, the ego vehicle is driving in a correct lane, otherwise, it is driving in an incorrect lane. Alternatively, if the value of 's' decreases, the correct lanes have positive id, so the value of 'lane_id' should be positive to indicate that the ego vehicle is located in a correct lane.

It is also important to remark that the 'offset' value is used to determine whether the ego vehicle is close to the road edge or not. It is considered that if the value of the field 'offset' is greater than 1, the ego vehicle is too close to the road edge, and it gets a negative reward, as previously explained.

6.7 Code

The code can be found on the next link: Code

Chapter 7

Testing

7.1 Parameters

The implementation has different parameters that can be tuned to improve the model behavior.

- Number of layers of the neural network.
- Number of neurons per layer on the neural network.
- Learning rate.
- Buffer size.
- Gamma value for Bellman's equation.
- Epsilon decay.
- Number of episodes used to fill the buffer before training.
- Neural network update frequency.
- Neural network synchronization frequency.

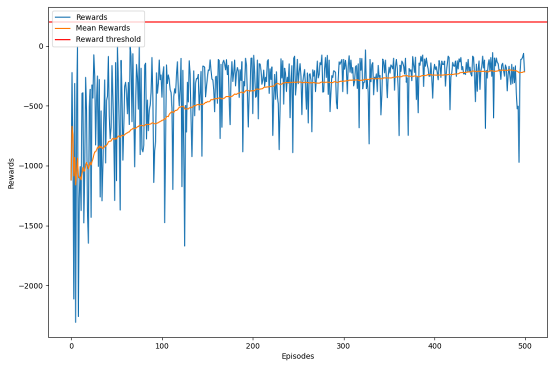
7.2 Testing with the first approach (Using trajectories and LVP MPC)

All test experiments conducted with this configuration were performed using initial reward systems. The negative reward when the vehicle is close to the edge was not implemented, and various approaches to the reward system were explored. A direct comparison with the final reward system is not viable due to disparities. The rewards allocated for each action differ, and the frequency of actuation is lower, resulting in a reduced total reward dimension in absolute value.

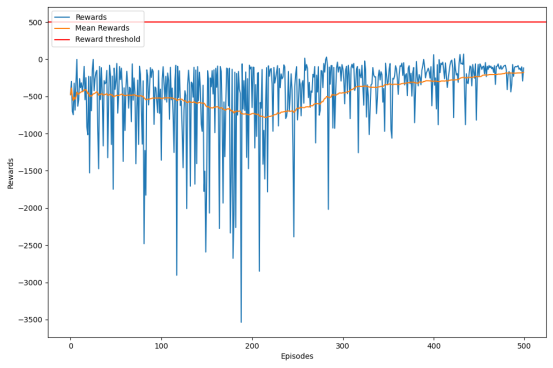
On the following video, it is shown the execution of the training with this approach: Training execution.

On the following figures, it is shown the reward evolution of some training experiments using the first approach. In blue is represented the actual total reward per episode, in orange, the average of the last 100 episodes, and in red, the reward threshold goal. When the average reward reaches the reward threshold goal, the training ends.

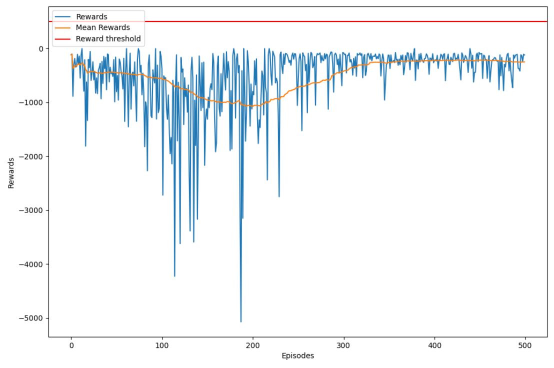
The following training experiments have been done with a duration of 500 episodes.



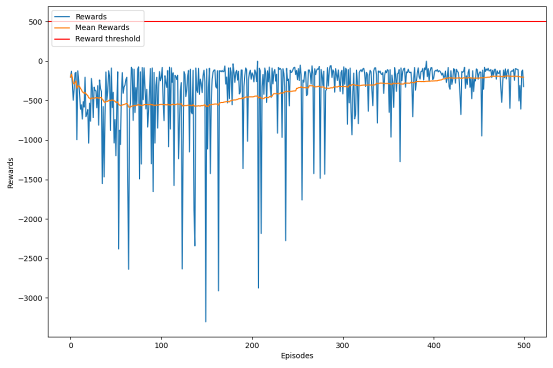
(a) Training 1



(b) Training 2

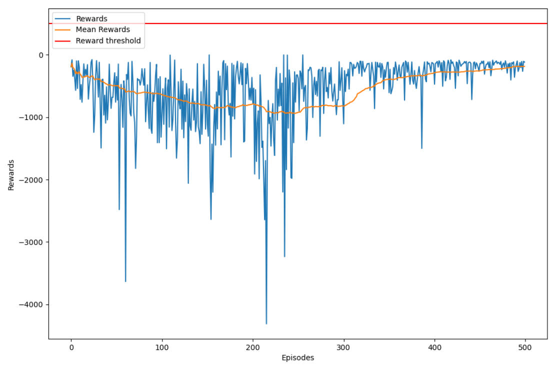


(c) Training 3

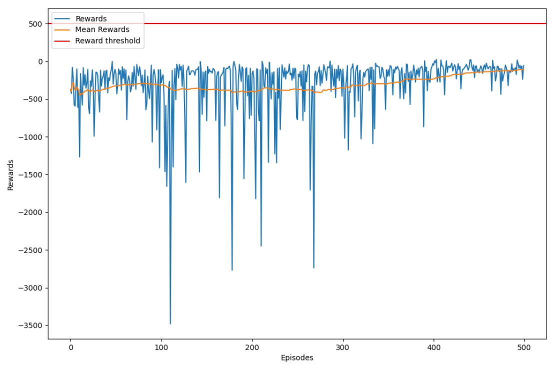


(d) Training 4

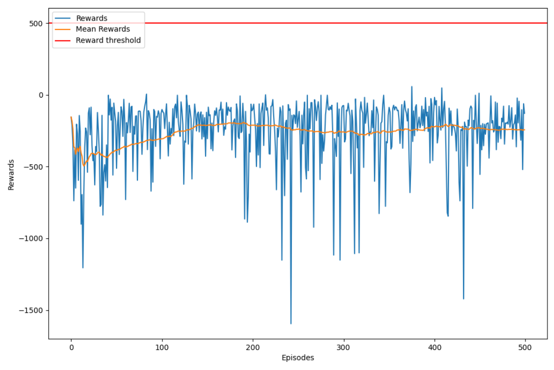
Figure 7.1: Reward evolution of four training experiments. Source: Author's elaboration



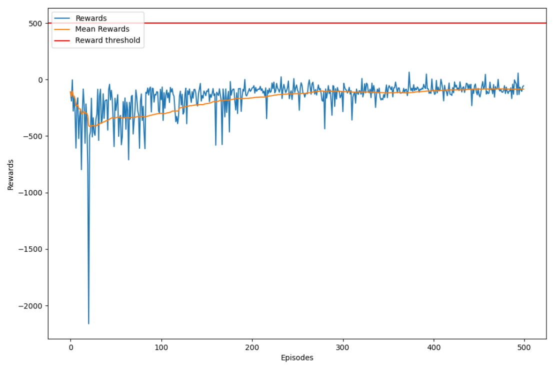
(e) Training 5



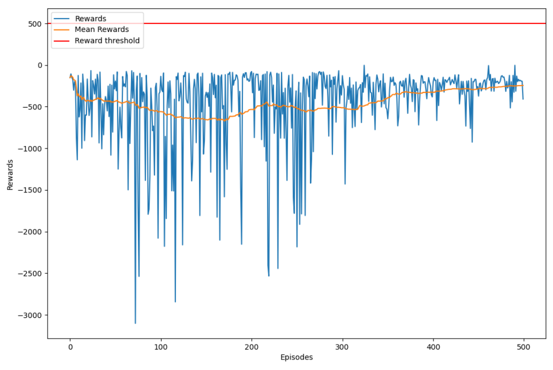
(f) Training 6



(g) Training 7



(h) Training 8



(e) Training 9

Figure 7.2: Reward evolution of five training experiments. Source: Author’s elaboration

The parameters for those training experiments are as follows:

- Training 1: learning rate: 0.01, num. layers: 2, num. neurons: 32, mem. size: 5000, eps. decay: 0.995, upd. freq.: 20, sync. freq.: 50
- Training 2: learning rate: 0.01, num. layers: 2, num. neurons: 16, mem. size: 10000, eps. decay: 0.995, upd. freq.: 20, sync. freq.: 50
- Training 3: learning rate: 0.01, num. layers: 2, num. neurons: 32, mem. size: 10000, eps. decay: 0.995, upd. freq.: 20, sync. freq.: 50
- Training 4: learning rate: 0.01, num. layers: 2, num. neurons: 16, mem. size: 2000, eps. decay: 0.995, upd. freq.: 20, sync. freq.: 50
- Training 5: learning rate: 0.01, num. layers: 2, num. neurons: 32, mem. size: 2000, eps. decay: 0.995, upd. freq.: 20, sync. freq.:50
- Training 6: learning rate: 0.01, num. layers: 2, num. neurons: 64, mem. size: 2000, eps. decay: 0.995, upd. freq.: 20, sync. freq.: 50
- Training 7: learning rate: 0.01, num. layers: 2, num. neurons: 16, mem. size: 50000, eps. decay: 0.995, upd. freq.: 20, sync. freq.: 50
- Training 8: learning rate: 0.01, num. layers: 2, num. neurons: 32, mem. size: 50000, eps. decay: 0.995, upd. freq.: 20, sync. freq.: 50
- Training 9: learning rate: 0.01, num. layers: 2, num. neurons: 64, mem. size: 50000, eps. decay: 0.995, upd. freq.: 20, sync. freq.: 50

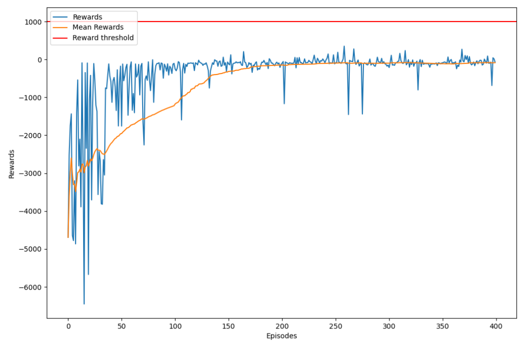
As it is shown on the previous figures, there is not a clear learning evolution. It seems there is a total reward value that the agent cannot surpass. After checking the trained agents, it was found that the stability point found by the training process is actually going out of the track as fast as possible. The agent decides that it is better to assume the negative reward of going out of the track, than drive and eventually increase the total negative reward. After some evaluations, it was noticed the following behavior: When the agent decides to follow a turning trajectory, it takes some time to start actually turning. It would be needed to wait for the whole trajectory to be followed before starting the next action. This arises two main problems: Firstly, depending on the speed, it would take a different amount of time to finish the whole trajectory, meaning that the frequency of actuation should be variable depending on the speed of the ego vehicle, and secondly, as stated on the section 6.3.1, the latency induced by this approach cannot be assumed by an autonomous vehicle.

On the following video, 10 runs of an agent trained with this approach are shown (training represented on Figure 7.6): Trained with first approach agent.

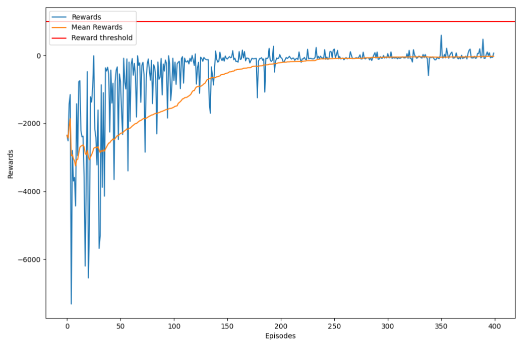
7.2.1 Testing at maximum frequency of actuation

The following testing experiments were conducted without introducing latency after each action. The reward system is the result of the testing from prior experiments, resulting in almost the final reward system. It was the final reward system without using the notion of ”too close to the edge” and without limiting the maximum reward (given that division by the offset might yield a significantly large result when the offset approaches zero).

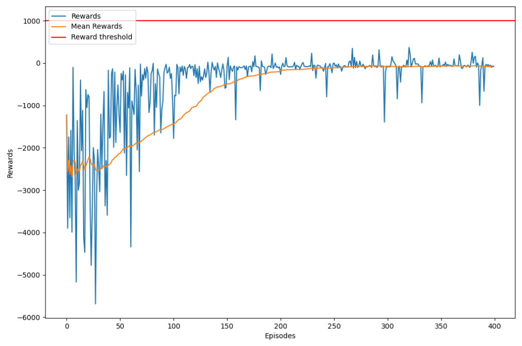
For the experiments with these modifications, stability at around 400 episodes of training on the maximum reward was detected, therefore, the training was limited to 400 episodes.



(a) Training 10



(b) Training 11



(c) Training 12

Figure 7.3: Reward evolution of three training experiments. Source: Author’s elaboration

The parameters for those training experiments are as follows:

- Training 10: learning rate: 0.01, num. layers: 4, num. neurons: 16, mem. size: 3000, eps. decay: 0.995, upd. freq.: 5, sync. freq.: 50
- Training 11: learning rate: 0.01, num. layers: 4, num. neurons: 32, mem. size: 3000, eps. decay: 0.995, upd. freq.: 5, sync. freq.: 50
- Training 12: learning rate: 0.01, num. layers: 4, num. neurons: 64, mem. size: 3000, eps. decay: 0.995, upd. freq.: 5, sync. freq.: 50

These training experiments yield into a new behavior on the agent. It learned that the best option is to drive straight, trying to increase the speed at maximum. On the following video, a trained agent behavior is shown: Second trained with first approach agent.

Finally, by changing the learning rate and increasing significantly the number of neurons, these results were achieved:

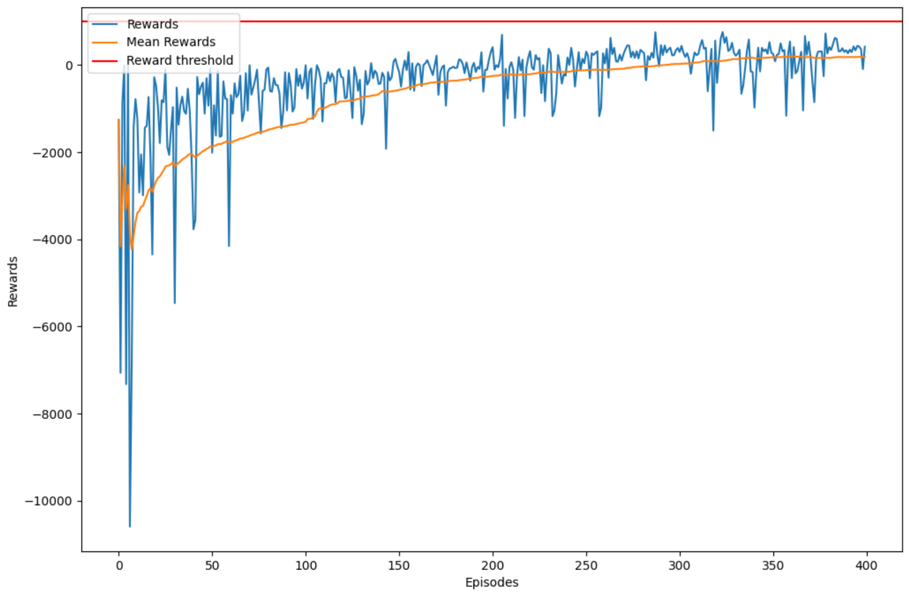


Figure 7.4: Training with: learning rate: 0.001, num. layers: 4, num. neurons: 512, mem. size: 3000, eps. decay: 0.995, upd. freq.: 5, sync. freq.: 50. Source: Author's elaboration

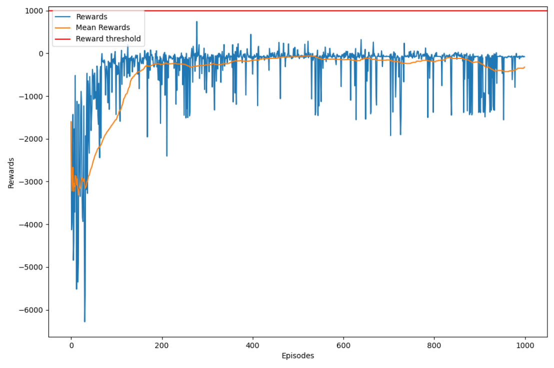
On the following video, the behavior of this agent is shown, which tries to stay on the lane: Trained with first approach best agent.

7.3 Testing with the second approach (Using throttle, brake and steering angle instructions)

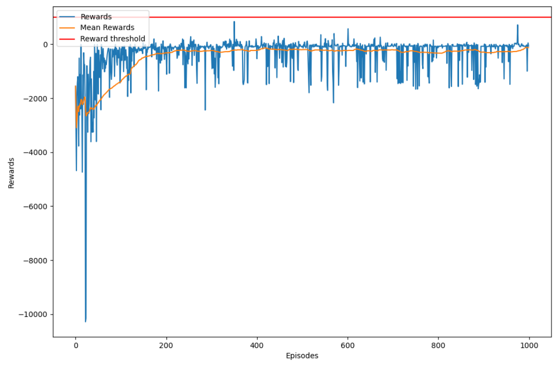
The following training experiments were executed using the second approach, where instead of using the LVP MPC to compute the acceleration and steering wheel angle, those values are sent directly to the low level controller.

Actually, just by changing the approach, a significant improvement on the behavior of the agents is noticed. On the following video, the execution of the same agent as the one on the last video is shown, using the same weights, but changing the approach: Same agent, using second approach.

The following testing executions were done using the second approach. The final system reward was decided during this testing phase: It was observed that the agent failed to rectify its trajectory before transitioning into an incorrect lane. In an attempt to address this issue, it was applied the negative reward when the ego vehicle approached too close to the edge of a correct lane. This adjustment made the agent to correct its trajectory sooner, minimizing unnecessary lane changes.

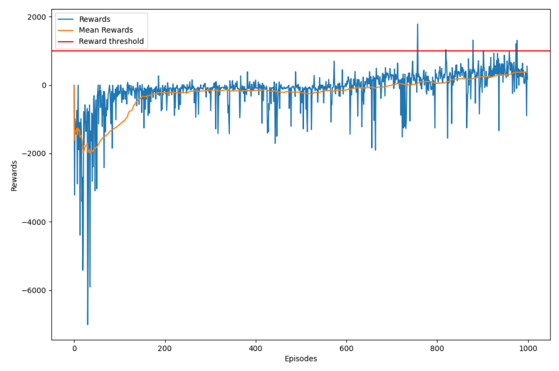


(a) Training 1

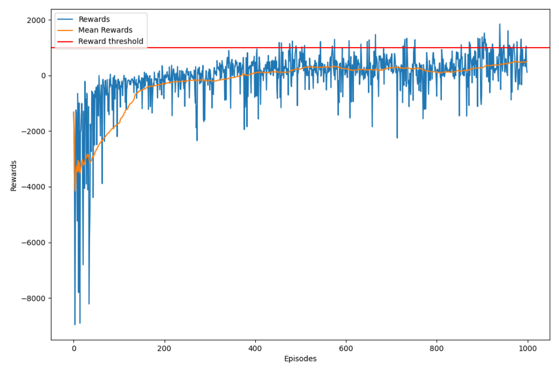


(b) Training 2

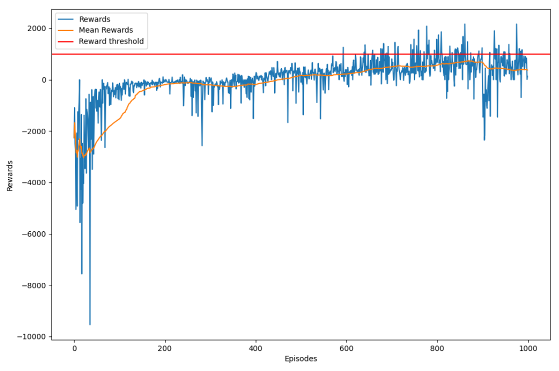
Figure 7.5: Reward evolution of two training experiments. Source: Author's elaboration



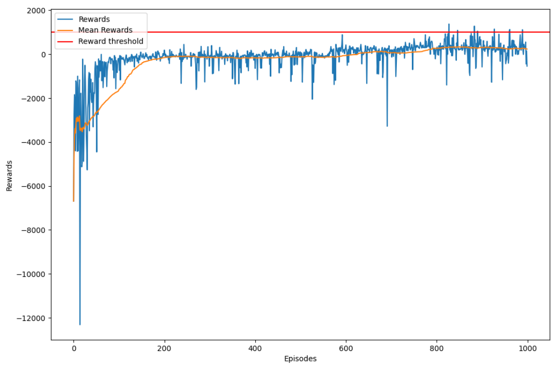
(c) Training 3



(d) Training 4



(e) Training 5



(f) Training 6

Figure 7.6: Reward evolution of four training experiments. Source: Author's elaboration

The parameters for those training experiments are as follows:

- Training 1: learning rate: 0.01, num. layers: 6, num. neurons: 256, mem. size: 3000, eps. decay: 0.995, upd. freq.: 5, sync. freq.: 50
- Training 2: learning rate: 0.01, num. layers: 6, num. neurons: 512, mem. size: 3000, eps. decay: 0.995, upd. freq.: 5, sync. freq.: 50
- Training 3: learning rate: 0.001, num. layers: 6, num. neurons: 256, mem. size: 3000, eps. decay: 0.995, upd. freq.: 5, sync. freq.: 50
- Training 4: learning rate: 0.001, num. layers: 6, num. neurons: 512, mem. size: 3000, eps. decay: 0.995, upd. freq.: 5, sync. freq.: 50

- Training 5: learning rate: 0.0005, num. layers: 6, num. neurons: 256, mem. size: 3000, eps. decay: 0.995, upd. freq.: 5, sync. freq.: 50

- Training 6: learning rate: 0.0005, num. layers: 6, num. neurons: 512, mem. size: 3000, eps. decay: 0.995, upd. freq.: 5, sync. freq.: 50

After these runs, it was observed that there was room for improvement by increasing the number of episodes. Consequently, the number of episodes was raised to 10k. This adjustment resulted in reaching the target of achieving a mean of 1000 points over the last 100 episodes in just over 1200 episodes.

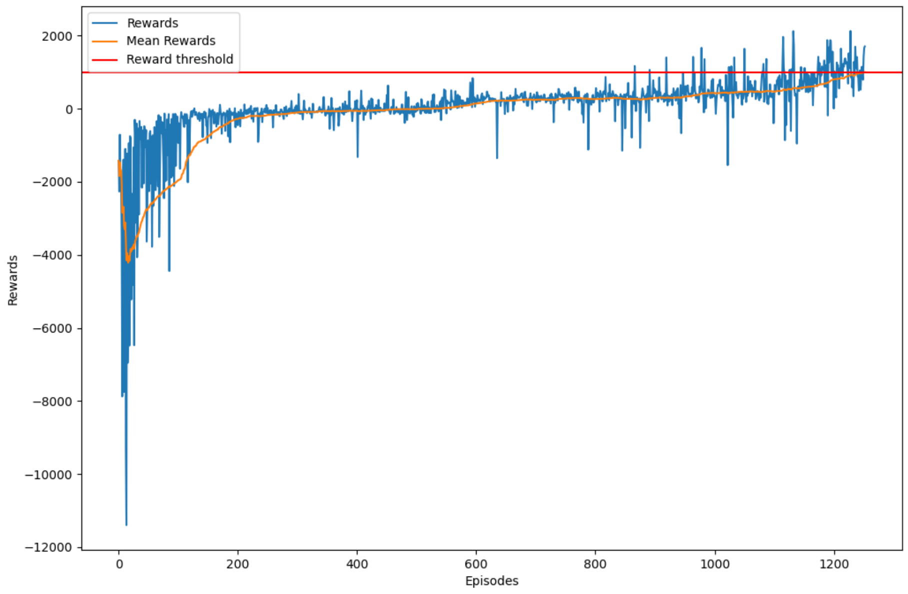


Figure 7.7: Training with: learning rate: 0.0005, num. layers: 6, num. neurons: 512, mem. size: 3000, eps. decay: 0.995, upd. freq.: 4, sync. freq.: 50. Source: Author's elaboration

Once the target of achieving a mean of 1000 points over the last 100 episodes was achieved, it was decided to explore the limits of the agent, by increasing the target to 10k. The final results led to a maximum mean reward on the last 100 episodes bigger than 2000 points.

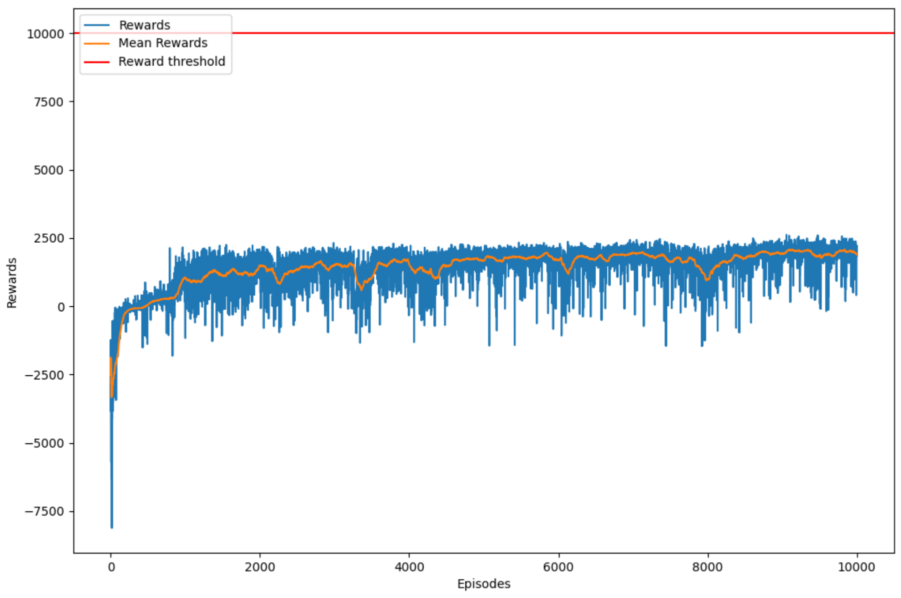


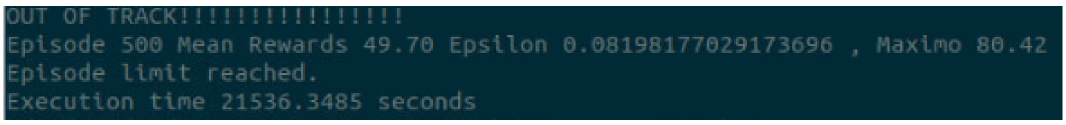
Figure 7.8: Training with: learning rate: 0.0005, num. layers: 6, num. neurons: 512, mem. size: 50000, eps. decay: 0.995, upd. freq.: 4, sync. freq.: 50. Source: Author's elaboration

On the following video, it is shown the behavior of the final trained agent: Best agent

7.4 Carbon emissions

To perform a CO_2 emissions evaluation, two different training sessions were conducted to measure their energy consumption. Both sessions involved training using the latest configuration. The first session was executed without launching CARLA, while the second session ran concurrently with the CARLA simulator. Both sessions consisted in 500 episodes.

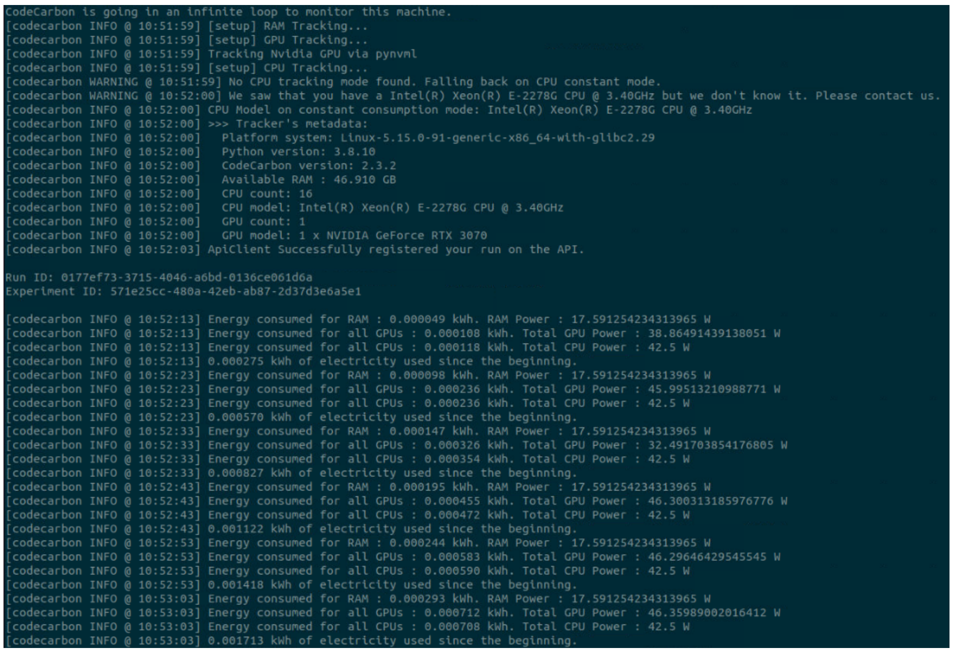
7.4.1 Training session without using CARLA



```
OUT OF TRACK!!!!!!!!!!!!!!!!
Episode 500 Mean Rewards 49.70 Epsilon 0.08198177029173696 , Maximo 80.42
Episode limit reached.
Execution time 21536.3485 seconds
```

Figure 7.9: Execution time for the first training session. Source: Author's elaboration

As can be seen in the Figure 7.9, the execution time for the first training session, was 21536 seconds, which are 5 hours and 59 minutes.

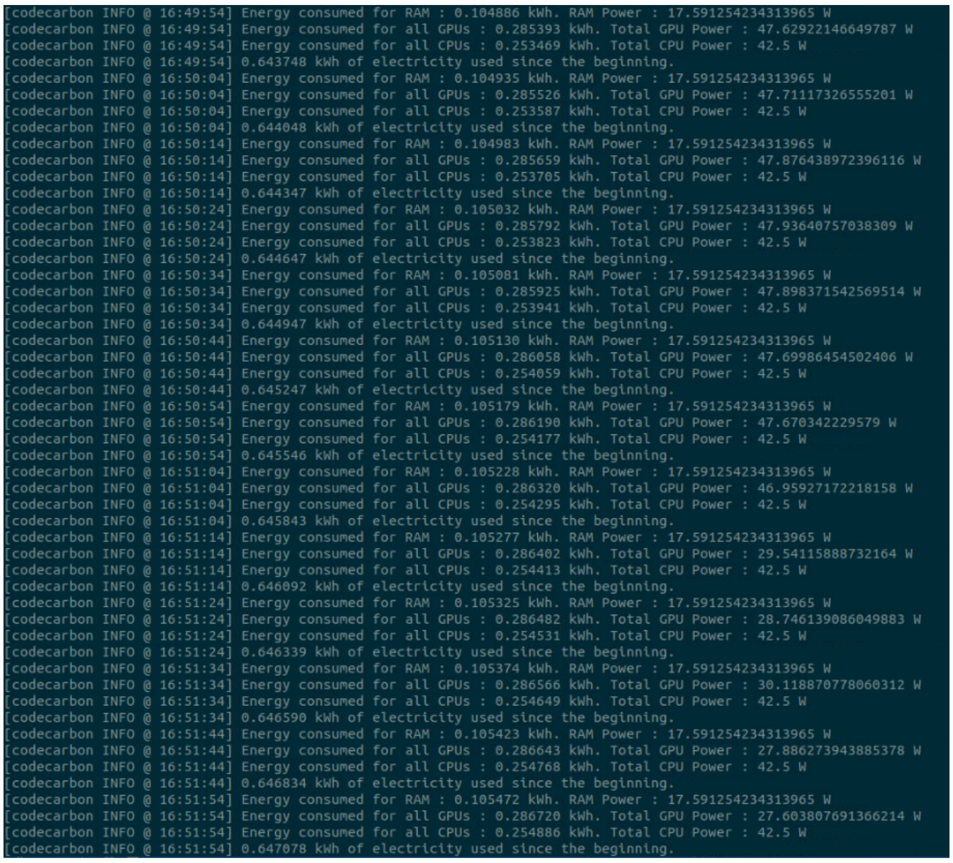


```
CodeCarbon is going in an infinite loop to monitor this machine.
[codecarbon INFO @ 10:51:59] [setup] RAM Tracking...
[codecarbon INFO @ 10:51:59] [setup] GPU Tracking...
[codecarbon INFO @ 10:51:59] Tracking Nvidia GPU via pynvml
[codecarbon INFO @ 10:51:59] [setup] CPU Tracking...
[codecarbon WARNING @ 10:51:59] No CPU tracking mode found. Falling back on CPU constant mode.
[codecarbon WARNING @ 10:52:00] We saw that you have a Intel(R) Xeon(R) E-2278G CPU @ 3.40GHz but we don't know it. Please contact us.
[codecarbon INFO @ 10:52:00] CPU Model on constant consumption mode: Intel(R) Xeon(R) E-2278G CPU @ 3.40GHz
[codecarbon INFO @ 10:52:00] >>> Tracker's metadata:
[codecarbon INFO @ 10:52:00]   Platform system: Linux-5.15.0-91-generic-x86_64-with-glibc2.29
[codecarbon INFO @ 10:52:00]   Python version: 3.8.10
[codecarbon INFO @ 10:52:00]   CodeCarbon version: 2.3.2
[codecarbon INFO @ 10:52:00]   Available RAM : 46.910 GB
[codecarbon INFO @ 10:52:00]   CPU count: 16
[codecarbon INFO @ 10:52:00]   CPU model: Intel(R) Xeon(R) E-2278G CPU @ 3.40GHz
[codecarbon INFO @ 10:52:00]   GPU count: 1
[codecarbon INFO @ 10:52:00]   GPU model: 1 x NVIDIA GeForce RTX 3070
[codecarbon INFO @ 10:52:03] ApiClient Successfully registered your run on the API.

Run ID: 0177ef73-3715-4046-a6bd-0136ce061d6a
Experiment ID: 571e25cc-480a-42eb-ab87-2d37d3e6a5e1

[codecarbon INFO @ 10:52:13] Energy consumed for RAM : 0.000049 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:52:13] Energy consumed for all GPUs : 0.000108 kWh. Total GPU Power : 38.86491439138051 W
[codecarbon INFO @ 10:52:13] Energy consumed for all CPUs : 0.000118 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 10:52:13] 0.000275 kWh of electricity used since the beginning.
[codecarbon INFO @ 10:52:23] Energy consumed for RAM : 0.000098 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:52:23] Energy consumed for all GPUs : 0.000236 kWh. Total GPU Power : 45.99513210988771 W
[codecarbon INFO @ 10:52:23] Energy consumed for all CPUs : 0.000236 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 10:52:23] 0.000570 kWh of electricity used since the beginning.
[codecarbon INFO @ 10:52:33] Energy consumed for RAM : 0.000147 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:52:33] Energy consumed for all GPUs : 0.000326 kWh. Total GPU Power : 32.491703854176805 W
[codecarbon INFO @ 10:52:33] Energy consumed for all CPUs : 0.000354 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 10:52:33] 0.000827 kWh of electricity used since the beginning.
[codecarbon INFO @ 10:52:43] Energy consumed for RAM : 0.000195 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:52:43] Energy consumed for all GPUs : 0.000455 kWh. Total GPU Power : 46.300313185976776 W
[codecarbon INFO @ 10:52:43] Energy consumed for all CPUs : 0.000472 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 10:52:43] 0.001122 kWh of electricity used since the beginning.
[codecarbon INFO @ 10:52:53] Energy consumed for RAM : 0.000244 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:52:53] Energy consumed for all GPUs : 0.000583 kWh. Total GPU Power : 46.29646429545545 W
[codecarbon INFO @ 10:52:53] Energy consumed for all CPUs : 0.000590 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 10:52:53] 0.001418 kWh of electricity used since the beginning.
[codecarbon INFO @ 10:53:03] Energy consumed for RAM : 0.000293 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:53:03] Energy consumed for all GPUs : 0.000712 kWh. Total GPU Power : 46.35989002016412 W
[codecarbon INFO @ 10:53:03] Energy consumed for all CPUs : 0.000708 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 10:53:03] 0.001713 kWh of electricity used since the beginning.
```

Figure 7.10: First training session. Monitoring start. Source: Author's elaboration



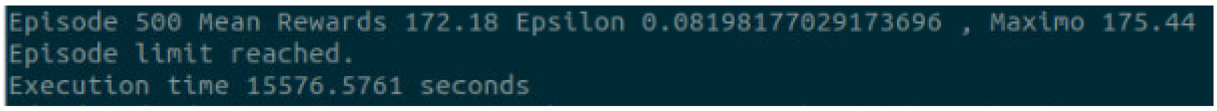
```
[codecarbon INFO @ 16:49:54] Energy consumed for RAM : 0.104886 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:49:54] Energy consumed for all GPUs : 0.285393 kWh. Total GPU Power : 47.62922146649787 W
[codecarbon INFO @ 16:49:54] Energy consumed for all CPUs : 0.253469 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:49:54] 0.643748 kWh of electricity used since the beginning.
[codecarbon INFO @ 16:50:04] Energy consumed for RAM : 0.104935 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:50:04] Energy consumed for all GPUs : 0.285526 kWh. Total GPU Power : 47.71117326555201 W
[codecarbon INFO @ 16:50:04] Energy consumed for all CPUs : 0.253587 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:50:04] 0.644048 kWh of electricity used since the beginning.
[codecarbon INFO @ 16:50:14] Energy consumed for RAM : 0.104983 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:50:14] Energy consumed for all GPUs : 0.285659 kWh. Total GPU Power : 47.876438972396116 W
[codecarbon INFO @ 16:50:14] Energy consumed for all CPUs : 0.253705 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:50:14] 0.644347 kWh of electricity used since the beginning.
[codecarbon INFO @ 16:50:24] Energy consumed for RAM : 0.105032 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:50:24] Energy consumed for all GPUs : 0.285792 kWh. Total GPU Power : 47.93640757038309 W
[codecarbon INFO @ 16:50:24] Energy consumed for all CPUs : 0.253823 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:50:24] 0.644647 kWh of electricity used since the beginning.
[codecarbon INFO @ 16:50:34] Energy consumed for RAM : 0.105081 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:50:34] Energy consumed for all GPUs : 0.285925 kWh. Total GPU Power : 47.898371542569514 W
[codecarbon INFO @ 16:50:34] Energy consumed for all CPUs : 0.253941 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:50:34] 0.644947 kWh of electricity used since the beginning.
[codecarbon INFO @ 16:50:44] Energy consumed for RAM : 0.105130 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:50:44] Energy consumed for all GPUs : 0.286058 kWh. Total GPU Power : 47.69986454502406 W
[codecarbon INFO @ 16:50:44] Energy consumed for all CPUs : 0.254059 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:50:44] 0.645247 kWh of electricity used since the beginning.
[codecarbon INFO @ 16:50:54] Energy consumed for RAM : 0.105179 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:50:54] Energy consumed for all GPUs : 0.286190 kWh. Total GPU Power : 47.670342229579 W
[codecarbon INFO @ 16:50:54] Energy consumed for all CPUs : 0.254177 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:50:54] 0.645546 kWh of electricity used since the beginning.
[codecarbon INFO @ 16:51:04] Energy consumed for RAM : 0.105228 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:51:04] Energy consumed for all GPUs : 0.286320 kWh. Total GPU Power : 46.95927172218158 W
[codecarbon INFO @ 16:51:04] Energy consumed for all CPUs : 0.254295 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:51:04] 0.645843 kWh of electricity used since the beginning.
[codecarbon INFO @ 16:51:14] Energy consumed for RAM : 0.105277 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:51:14] Energy consumed for all GPUs : 0.286402 kWh. Total GPU Power : 29.54115888732164 W
[codecarbon INFO @ 16:51:14] Energy consumed for all CPUs : 0.254413 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:51:14] 0.646092 kWh of electricity used since the beginning.
[codecarbon INFO @ 16:51:24] Energy consumed for RAM : 0.105325 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:51:24] Energy consumed for all GPUs : 0.286482 kWh. Total GPU Power : 28.746139086049883 W
[codecarbon INFO @ 16:51:24] Energy consumed for all CPUs : 0.254531 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:51:24] 0.646339 kWh of electricity used since the beginning.
[codecarbon INFO @ 16:51:34] Energy consumed for RAM : 0.105374 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:51:34] Energy consumed for all GPUs : 0.286566 kWh. Total GPU Power : 30.118870778060312 W
[codecarbon INFO @ 16:51:34] Energy consumed for all CPUs : 0.254649 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:51:34] 0.646590 kWh of electricity used since the beginning.
[codecarbon INFO @ 16:51:44] Energy consumed for RAM : 0.105423 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:51:44] Energy consumed for all GPUs : 0.286643 kWh. Total GPU Power : 27.886273943885378 W
[codecarbon INFO @ 16:51:44] Energy consumed for all CPUs : 0.254768 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:51:44] 0.646834 kWh of electricity used since the beginning.
[codecarbon INFO @ 16:51:54] Energy consumed for RAM : 0.105472 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 16:51:54] Energy consumed for all GPUs : 0.286720 kWh. Total GPU Power : 27.603807691366214 W
[codecarbon INFO @ 16:51:54] Energy consumed for all CPUs : 0.254886 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 16:51:54] 0.647078 kWh of electricity used since the beginning.
```

Figure 7.11: First training session. Monitoring end. Source: Author's elaboration

On the Figures 7.10 and 7.11 are shown the start and the end of the energy consumption monitoring. On the Figure 7.11 can be seen when the training sessions ended, just by checking the energy power demanded on the GPU. During the training, the power on the GPU was around 47 W, and once it finished, the power went down to the range of 27-30 W. The total energy consumption during the training session was 0.646 kWh.

The execution has taken place in Tarragona. The most recent data available from the Catalan government on the equivalence of CO_2 and kWh is from the 3rd of May 2023, which gives a ratio of 273 g CO2eq/kWh. Considering such value, the total carbon footprint of the training session is 0.173 Kg CO_2 eq.

7.4.2 Training session using CARLA



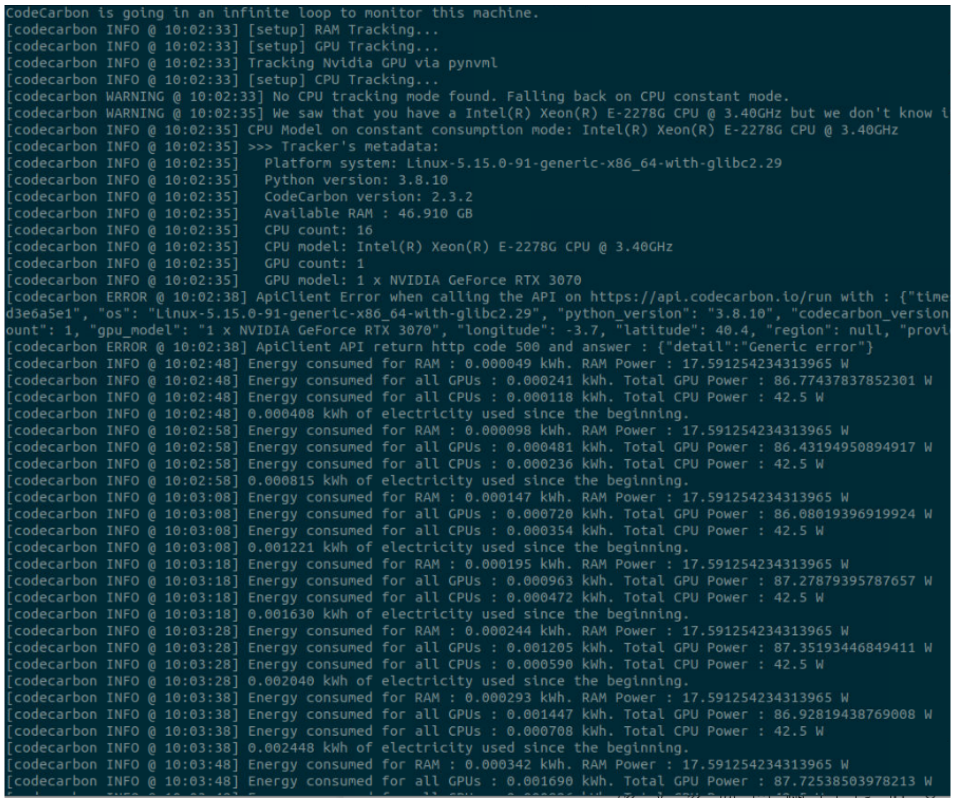
```
Episode 500 Mean Rewards 172.18 Epsilon 0.08198177029173696 , Maximo 175.44
Episode limit reached.
Execution time 15576.5761 seconds
```

Figure 7.12: Execution time for the second training session. Source: Author's elaboration

As shown on Figure 7.12, the second training session was faster, finishing the 500 episodes in 15576 seconds, or 4 hours and 20 minutes.

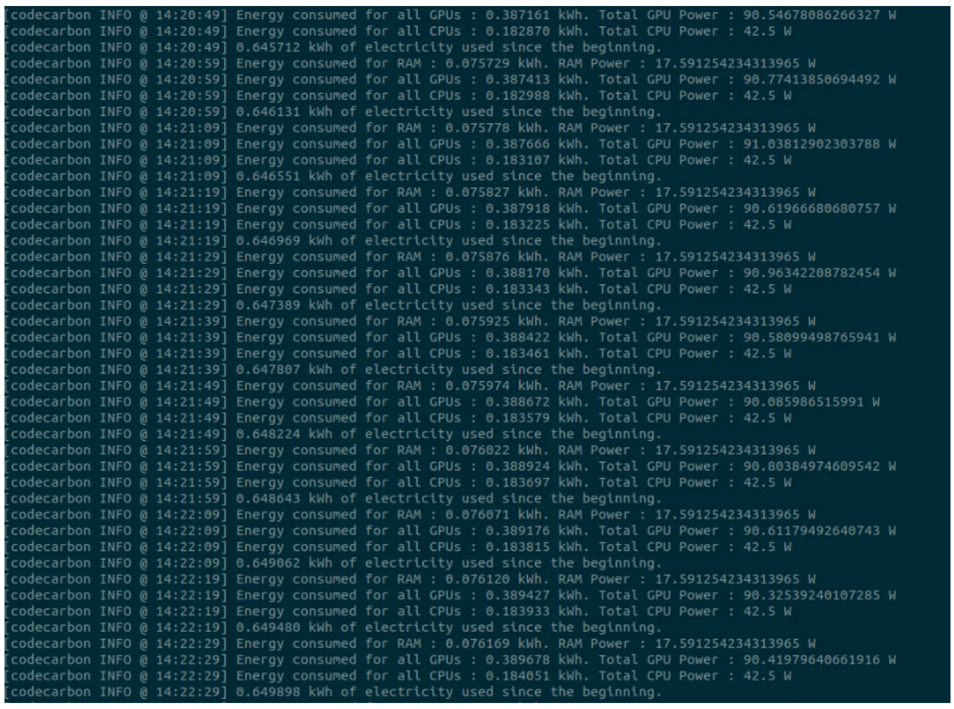
It is important to note that the duration of the training sessions are different because they depend on the total time each episode takes. For example, one the first training session, maybe the agent was able to not go out of the track while driving at a slow speed, making the episodes taking so long, or maybe the reason for such difference is that in the second training session, the agent decided to go out of the track fast in many episodes, making the total time significantly smaller.

On the following figure, the initial and final power required by the different components involved on the execution, and the total energy consumption of the training session are represented.



```
CodeCarbon is going in an infinite loop to monitor this machine.
[codecarbon INFO @ 10:02:33] [setup] RAM Tracking...
[codecarbon INFO @ 10:02:33] [setup] GPU Tracking...
[codecarbon INFO @ 10:02:33] Tracking Nvidia GPU via pynvml
[codecarbon INFO @ 10:02:33] [setup] CPU Tracking...
[codecarbon WARNING @ 10:02:33] No CPU tracking mode found. Falling back on CPU constant mode.
[codecarbon WARNING @ 10:02:35] We saw that you have a Intel(R) Xeon(R) E-2278G CPU @ 3.40GHz but we don't know i
[codecarbon INFO @ 10:02:35] CPU Model on constant consumption mode: Intel(R) Xeon(R) E-2278G CPU @ 3.40GHz
[codecarbon INFO @ 10:02:35] >>> Tracker's metadata:
[codecarbon INFO @ 10:02:35]   Platform system: Linux-5.15.0-91-generic-x86_64-with-glibc2.29
[codecarbon INFO @ 10:02:35]   Python version: 3.8.10
[codecarbon INFO @ 10:02:35]   CodeCarbon version: 2.3.2
[codecarbon INFO @ 10:02:35]   Available RAM : 46.910 GB
[codecarbon INFO @ 10:02:35]   CPU count: 16
[codecarbon INFO @ 10:02:35]   CPU model: Intel(R) Xeon(R) E-2278G CPU @ 3.40GHz
[codecarbon INFO @ 10:02:35]   GPU count: 1
[codecarbon INFO @ 10:02:35]   GPU model: 1 x NVIDIA GeForce RTX 3070
[codecarbon ERROR @ 10:02:38] ApiClient Error when calling the API on https://api.codecarbon.io/run with : {"time
d3e6a5e1", "os": "Linux-5.15.0-91-generic-x86_64-with-glibc2.29", "python_version": "3.8.10", "codecarbon_version
ount": 1, "gpu_model": "1 x NVIDIA GeForce RTX 3070", "longitude": -3.7, "latitude": 40.4, "region": null, "provi
[codecarbon ERROR @ 10:02:38] ApiClient API return http code 500 and answer : {"detail":"Generic error"}
[codecarbon INFO @ 10:02:48] Energy consumed for RAM : 0.000049 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:02:48] Energy consumed for all GPUs : 0.000241 kWh. Total GPU Power : 86.77437837852301 W
[codecarbon INFO @ 10:02:48] Energy consumed for all CPUs : 0.000118 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 10:02:48] 0.000408 kWh of electricity used since the beginning.
[codecarbon INFO @ 10:02:58] Energy consumed for RAM : 0.000098 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:02:58] Energy consumed for all GPUs : 0.000481 kWh. Total GPU Power : 86.43194950894917 W
[codecarbon INFO @ 10:02:58] Energy consumed for all CPUs : 0.000236 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 10:02:58] 0.000815 kWh of electricity used since the beginning.
[codecarbon INFO @ 10:03:08] Energy consumed for RAM : 0.000147 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:03:08] Energy consumed for all GPUs : 0.000720 kWh. Total GPU Power : 86.08019396919924 W
[codecarbon INFO @ 10:03:08] Energy consumed for all CPUs : 0.000354 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 10:03:08] 0.001221 kWh of electricity used since the beginning.
[codecarbon INFO @ 10:03:18] Energy consumed for RAM : 0.000195 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:03:18] Energy consumed for all GPUs : 0.000963 kWh. Total GPU Power : 87.27879395787657 W
[codecarbon INFO @ 10:03:18] Energy consumed for all CPUs : 0.000472 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 10:03:18] 0.001630 kWh of electricity used since the beginning.
[codecarbon INFO @ 10:03:28] Energy consumed for RAM : 0.000244 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:03:28] Energy consumed for all GPUs : 0.001205 kWh. Total GPU Power : 87.35193446849411 W
[codecarbon INFO @ 10:03:28] Energy consumed for all CPUs : 0.000590 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 10:03:28] 0.002040 kWh of electricity used since the beginning.
[codecarbon INFO @ 10:03:38] Energy consumed for RAM : 0.000293 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:03:38] Energy consumed for all GPUs : 0.001447 kWh. Total GPU Power : 86.92819438769008 W
[codecarbon INFO @ 10:03:38] Energy consumed for all CPUs : 0.000708 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 10:03:38] 0.002448 kWh of electricity used since the beginning.
[codecarbon INFO @ 10:03:48] Energy consumed for RAM : 0.000342 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 10:03:48] Energy consumed for all GPUs : 0.001690 kWh. Total GPU Power : 87.72538503978213 W
```

Figure 7.13: Second training session. Monitoring start. Source: Author's elaboration



```
[codecarbon INFO @ 14:20:49] Energy consumed for all GPUs : 0.387161 kWh. Total GPU Power : 90.54678086266327 W
[codecarbon INFO @ 14:20:49] Energy consumed for all CPUs : 0.182870 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 14:20:49] 0.645712 kWh of electricity used since the beginning.
[codecarbon INFO @ 14:20:59] Energy consumed for RAM : 0.075729 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 14:20:59] Energy consumed for all GPUs : 0.387413 kWh. Total GPU Power : 90.77413850694492 W
[codecarbon INFO @ 14:20:59] Energy consumed for all CPUs : 0.182988 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 14:20:59] 0.646131 kWh of electricity used since the beginning.
[codecarbon INFO @ 14:21:09] Energy consumed for RAM : 0.075778 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 14:21:09] Energy consumed for all GPUs : 0.387666 kWh. Total GPU Power : 91.03812902303788 W
[codecarbon INFO @ 14:21:09] Energy consumed for all CPUs : 0.183107 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 14:21:09] 0.646551 kWh of electricity used since the beginning.
[codecarbon INFO @ 14:21:19] Energy consumed for RAM : 0.075827 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 14:21:19] Energy consumed for all GPUs : 0.387918 kWh. Total GPU Power : 90.61966680680757 W
[codecarbon INFO @ 14:21:19] Energy consumed for all CPUs : 0.183225 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 14:21:19] 0.646969 kWh of electricity used since the beginning.
[codecarbon INFO @ 14:21:29] Energy consumed for RAM : 0.075876 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 14:21:29] Energy consumed for all GPUs : 0.388170 kWh. Total GPU Power : 90.96342208782454 W
[codecarbon INFO @ 14:21:29] Energy consumed for all CPUs : 0.183343 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 14:21:29] 0.647389 kWh of electricity used since the beginning.
[codecarbon INFO @ 14:21:39] Energy consumed for RAM : 0.075925 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 14:21:39] Energy consumed for all GPUs : 0.388422 kWh. Total GPU Power : 90.58099498765941 W
[codecarbon INFO @ 14:21:39] Energy consumed for all CPUs : 0.183461 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 14:21:39] 0.647807 kWh of electricity used since the beginning.
[codecarbon INFO @ 14:21:49] Energy consumed for RAM : 0.075974 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 14:21:49] Energy consumed for all GPUs : 0.388672 kWh. Total GPU Power : 90.085986515991 W
[codecarbon INFO @ 14:21:49] Energy consumed for all CPUs : 0.183579 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 14:21:49] 0.648224 kWh of electricity used since the beginning.
[codecarbon INFO @ 14:21:59] Energy consumed for RAM : 0.076022 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 14:21:59] Energy consumed for all GPUs : 0.388924 kWh. Total GPU Power : 90.80384974609542 W
[codecarbon INFO @ 14:21:59] Energy consumed for all CPUs : 0.183697 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 14:21:59] 0.648643 kWh of electricity used since the beginning.
[codecarbon INFO @ 14:22:09] Energy consumed for RAM : 0.076071 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 14:22:09] Energy consumed for all GPUs : 0.389176 kWh. Total GPU Power : 90.61179492640743 W
[codecarbon INFO @ 14:22:09] Energy consumed for all CPUs : 0.183815 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 14:22:09] 0.649062 kWh of electricity used since the beginning.
[codecarbon INFO @ 14:22:19] Energy consumed for RAM : 0.076120 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 14:22:19] Energy consumed for all GPUs : 0.389427 kWh. Total GPU Power : 90.32539240107285 W
[codecarbon INFO @ 14:22:19] Energy consumed for all CPUs : 0.183933 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 14:22:19] 0.649480 kWh of electricity used since the beginning.
[codecarbon INFO @ 14:22:29] Energy consumed for RAM : 0.076169 kWh. RAM Power : 17.591254234313965 W
[codecarbon INFO @ 14:22:29] Energy consumed for all GPUs : 0.389678 kWh. Total GPU Power : 90.41979640661916 W
[codecarbon INFO @ 14:22:29] Energy consumed for all CPUs : 0.184051 kWh. Total CPU Power : 42.5 W
[codecarbon INFO @ 14:22:29] 0.649898 kWh of electricity used since the beginning.
```

Figure 7.14: Second training session. Monitoring end. Source: Author's elaboration

As shown on the Figures 7.13 and 7.14 the power required by the RAM and the CPU are the same as in the training session without using CARLA, on the other hand, the power consumption on the GPU has almost doubled from 47 to 90 W, as it could be expected.

The total training session energy consumption was 0.65 kWh, which is slightly higher than the energy consumption on the session without using CARLA. Applying the same equivalence, the carbon footprint of the training session using CARLA was 0.177 Kg CO_2 eq.

7.4.3 Difference of using CARLA or not

On the previous sections it has been shown the carbon footprint of two different training sessions. The total values of CO_2 cannot be generalized. In order to give an actual ratio, it is better to give the value per time. In that case:

- Without using CARLA: 0.173 Kg CO_2 eq. / 5h59min = 0.4912 g CO_2 eq. / min
- Using CARLA: 0.177 Kg CO_2 eq. / 4h20min = 0.6825 g CO_2 eq. / min

Which leads to a 28% energy savings for the trainings conducted without CARLA.

Chapter 8

Conclusions

8.1 Behavior

It is very important to contextualize the environment in which the agents have been trained. The agent only receives information about its relative position with respect to a map, along with current speed and accelerations. No visual inputs such as images or point clouds are provided to the agent during training. Additionally, CARLA waypoints are not utilized in the training phase.

Considering these limitations, the erratic behavior of the agent becomes clearer. It struggles to correct its course until it is too close to the road's edge, which results in a less smooth performance. Instead of maintaining a consistent, centered trajectory, it often accelerates to its maximum speed (which is limitated) and only corrects its heading when it is about to depart from the track. This behavior is expected due to the absence of information about desired lane positions (waypoints) during training. The reward strategy prioritized greater rewards for better centering by the agent, yet it appears to have no effect during the training phase.

To address this erratic behavior, incorporating visual inputs could be highly beneficial. Introducing a front camera with a lane detector or using raw images could assist the agent in overcoming this challenge.

8.2 Main challenges

Considering that the agent manages to go straight (with the already stated limitations), dealing with curves seems to be a bigger challenge for the agent.

As it has been mentioned on section 6.5.1, on an OpenScenario map, an intersection is defined by different road sectors, depending on the different options that are available for the traffic. For instance, a T intersection, where a vehicle have two options: continue driving straight, or turning. In that case, such intersection would have two different road sectors, or as they have been defined, 'road_id'. When the agent enters into an intersection, an algorithm to decide which 'road_id' is the correct one should be implemented to ensure the correct performance. Such algorithm has not been implemented during this project.

Since the agent is able to turn to fix its trajectory and maintain its trajectory within the road limits inside a road sector, the road sector change is the main challenge found. A possible solution would be to decide beforehand the route the agent has to follow, in which case the correct road sectors would be identified, transforming the 'road_id' selection algorithm to just a checking process: either it is in the correct 'road_id' or not.

Chapter 9

Further work

9.1 Training and implementation

As it has been concluded, additional inputs are necessary to make the agent able to learn. Initially, the project aimed to assess its performance with minimal inputs, which has proven inadequate. Consequently, further efforts suggest integrating visual inputs together alongside the already defined inputs, in order to help the agent to better understand its surroundings.

As it has not been implemented yet, a 'goal' selection is mandatory in an autonomous vehicle. It is crucial to know where the vehicle has to go. As mentioned in the previous section, alongside integrating the 'goal' selection, or destination, it will be necessary to incorporate a 'route' identification. Once the current location and the destination are defined, an algorithm that decides the best route (and identifies the 'road_id' sectors which the vehicle has to follow) should be implemented.

Algorithm 1 Route identification

```
Data: Current location, Destination, OpenScenario map
Result: List of 'road_id', where the vehicle should drive
Best route from current location to destination (List of Coordinates) is identified as 'Route'
for 'road_id' in map do
    if road section with such 'road_id' has coordinates in 'Route' then
        Append 'road_id' to output list
    end
end
```

It is also important to note that the training has been conducted without the presence of any other agent in the environment. Introducing traffic would enable the agent to learn how to navigate unexpected situations and interact with other entities.

In addition, incorporating traffic rules such as signal recognition and interpretation of traffic lights is necessary. Currently, the agent's objective is to drive at its maximum speed within road limits. However, it does not adhere to any traffic signs or lights, which should be addressed and implemented in subsequent iterations.

9.2 Adaptation to the real vehicle

The final step should be the introduction of the trained model into the real vehicle. For doing so, the system is already adapted using the ROS communication. If the model training had been successful using the specified inputs, the adaptation to the real vehicle would have been straightforward. However, since it is possible that achieving optimal performance might require incorporating visual inputs, an extra adaptation process will be needed since the simulation images are not identical to the real-world camera inputs. Therefore, an extra layer will be necessary to adapt the inputs from the real world to the inputs the model was trained on.

Bibliography

[1] J. Fleetwood, "Public health, ethics, and autonomous vehicles.," *Am J Public Health.*, 2017. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5343691/pdf/AJPH.2016.303628.pdf.

[2] L. Ruiz Dern, "Deep q-networks. pid 00275573," *UOC*, 2021.

[3] "Carla ros bridge." https://carla.readthedocs.io/projects/ros-bridge/en/latest/.

[4] J. Choi *et al.*, "Path planning based on bezier curve for autonomous ground vehicles," 2008. https://users.soe.ucsc.edu/~elkaim/Documents/camera_WCECS2008_IEEE_ICIAR_58.pdf.

[5] "Open drive scenario." https://www.asam.net/standards/detail/opendrive/.

[6] M. Bojarski *et al.*, "End to end learning for self-driving cars," 2016. https://arxiv.org/abs/1604.07316.

[7] J. Dinneweth *et al.*, "Multi-agent reinforcement learning for autonomous vehicles: a survey. auton. intell. syst. 2, 27," 2022. https://doi.org/10.1007/s43684-022-00045-z.

[8] A. La Fortelle, "Reinforcement learning for autonomous vehicles: A survey (arxiv, 2019),"

[9] B. Ben Elallid *et al.*, "A comprehensive survey on the application of deep and reinforcement learning approaches in autonomous driving," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 9, pp. 7366–7390, 2022. https://www.sciencedirect.com/science/article/pii/S1319157822000970.

[10] "Self-driving cars with convolutional neural networks (cnn)." https://neptune.ai/blog/self-driving-cars-with-convolutional-neural-networks-cnn. Accessed: 22/10/2023.

[11] A. HongIl and J. Jae-il, "Decision-making system for lane change using deep reinforcement learning in connected and automated driving," *Electronics*, vol. 8, no. 5, 2019. https://www.mdpi.com/2079-9292/8/5/543.

[12] B. Kiran *et al.*, "Deep reinforcement learning for autonomous driving: A survey," 2021. https://arxiv.org/abs/2002.00444.

[13] S. Yan Ho, "Reinforcement learning for self-driving cars," 2018. https://dr.ntu.edu.sg/handle/10356/74098.

[14] W. Jiawei *et al.*, "Multi-agent graph reinforcement learning for connected automated driving," 07 2020. https://www.researchgate.net/publication/342788148_Multi-agent_Graph_Reinforcement_Learning_for_Connected_Automated_Driving.