POLITECNICO DI TORINO

Master of Science in Data Science and Engineering

Master of Science thesis

Model-Based Reinforcement Learning for Driver Action Prediction

An AI-based solution using Model-Based Reinforcement Learning for a time-series forecasting problem in the automotive industry

Supervisor
Prof. Fabrizio LAMBERTI

Company Supervisors:
Dott. Ilario GERLERO
Dott. Nicolò CHIAPELLO

Candidate
Francesco SCORCA

December 2022

Abstract

Advanced Driver Assistance Systems (ADAS) can be significantly improved with effective driver action prediction: predicting driver actions early and accurately can help mitigate the effects of potentially unsafe driving behaviors, avoid possible accidents, and improve vehicle powertrain model predictive control applications. Concerning the interpretation of the term "action", consistent efforts in the literature focus on the vehicle's trajectory, while there exist works on the forecast of the driver's intention (e.g. going straight, turning left, etc.) or pedals pressure. The aim of this project is to develop a system predicting steering wheel angles, accelerator and brake pedals pressures in a fixed time-window, exploiting a sensor-less architecture: no additional sensors beyond those already present in the car are required, nor are any biometric readings necessary. The driver's actions are forecasted through an algorithm based on Artificial Intelligence that combines vehicle dynamics (e.g., lateral/longitudinal acceleration) and the perception of the road and the vehicle's surroundings. For the outlined purpose it is proposed a methodology that approaches the mentioned time-series forecasting problem by exploiting a Model-Based Reinforcement Learning framework. First, an autonomous driving agent is trained in a virtual simulation environment that reproduces the road conditions and driving dynamics of a motor vehicle. The Model-Based paradigm implies that, while interacting with the environment, a model of this is learned through Supervised Learning, creating an imaginary copy of the world. The action prediction system will envision the trained agent that, starting from the current state of the driver and of the environment, will move ahead in time in the learned environment model. The sequences of actions imagined are saved and represent the output of the system: the driver action prediction. The work conducted allows for a two-fold result. First, an analysis of the trade-offs in adopting a model-based approach or a model-free one in an industrial driving simulator, evaluated on the sample and compute efficiency of the Reinforcement Learning framework. Second, it is presented the possibility of obtaining a system capable of a prediction in a time-horizon built upon modules obtained through Model-Based Reinforcement Learning, evaluated on both metrics defined during the work, and metrics common in the literature.

Contents

Chapter 1

Introduction

1.1 Thesis genesis

This work is the result of a collaboration between the GRaphics And INtelligent Systems (GRAINS) group of Politecnico di Torino, Department of Control and Computer Engineering, and SENSOR Reply. SENSOR Reply is a Reply Group company with expertise in IoT applications powered by Artificial Intelligence techniques. The company's mission is to provide customers with data-driven software applications and engineering services for decision support.

The result of the work conducted is a system capable of a prediction in a time horizon built upon modules obtained through Model-Based Reinforcement Learning (MBRL), evaluated on both metrics defined during the work, and metrics common in the literature. Furthermore, even if not strictly required by the prediction system we propose, we perform experiments with Model-Free Reinforcement Learning (MFRL) techniques to show the trade-offs in adopting a model-based approach or a model-free one in an industrial driving simulator. This is evaluated on the sample and compute efficiency of the Reinforcement Learning framework.

The approach developed in this work is based on methodologies proposed in previous research projects [1, 2], whose purpose was different, but whose core idea remains the same: using an RL agent as a digital twin of the driver.

1.2 Objectives

The purpose of this thesis project is to develop a system capable of inferring a sequence of steering wheel angles, throttle and brake pedals pressures in a fixed time window.

In fact, Advanced Driver Assistance Systems (ADAS) can be significantly improved with effective driver action prediction: knowing in advance the future behavior of a human, it is possible to tune the vehicle or launch alerts depending on the characteristics of the driver. For instance, predicting driver actions early and accurately can help mitigate the effects of potentially unsafe driving behaviors, avoid possible accidents, and improve vehicle powertrain model predictive control applications [3, 4].

The aim is to develop a methodology with the prospect of future industrialization, keeping into account the non-availability of complex sources of data. Therefore, the proposed approach addresses the problem under analysis by providing a non-intrusive solution, exploiting the signals coming from the current onboard technology and from the vehicle dynamics. The software-based solution proposed has the advantage of not requiring the implementation of other instrumentation in the vehicle, no cameras or biometric tracking systems, to be strictly adhering to the principles of privacy and data protection.

The actual implementation envisions a system forecasting the driver's actions through an algorithm based on Artificial Intelligence that combines vehicle dynamics and the perception of the road and the vehicle's surroundings.

The development of this project can be summarised in three steps:

- Implementation of Model-Based Reinforcement Learning technique for training an autonomous driving agent, while learning as well a model of the environment into which the driver is inserted;

- Exploitation of the trained modules obtained for developing a prediction system outputting a series of future actions or trajectories by moving the RL agent ahead in time in the learned environment model. This step presents also a comparison with the literature considering a state-of-the-art model modified to work with our data;

- First trials of system enhancement, through the insertion of a driver profiling block, to choose the RL agent most similar to the human whose actions we want to predict.

1.3 Document structure

The present thesis document is characterized by the following structure.

- **Introduction**: description of the problem and the steps needed to achieve the final objective.

- **State of the art**: overview of several interpretations of "action prediction" in the literature, with the relative algorithms and tools commonly exploited.

- **Background**: description of the algorithms, techniques and physical models used in this work.

- **Material and methods**: description of the methodology developed throughout this thesis works, from the implementation of the MBRL framework to the realization of the prediction system. This chapter presents several experiments concerning different parts of the work.

- **Results**: quantitative analysis of the results and comments on their implications.

- **Discussion**: qualitative considerations of the idea leading the development or emerged throughout it.
- **Conclusions**: considerations on the outcomes of this thesis works and discussion of possible future extensions.

Chapter 2

State of the art

In this chapter, various ways the "driving action prediction" problem can be intended in the literature are presented. Having overviewed the methods, it is highlighted highlight how the proposed prediction system positions itself in this field. Finally, it is brought a review the research tools exploited by the state-of-the-art models and by the one developed throughout this thesis work.

2.1 Driving data

Before diving into the analysis of the methods, it is possible to summarize the typical data exploited for the prediction task. In fact, although the literature envisions different branches based on the algorithms' methodologies and outputs, the inputs remain quite the same and can be grouped into the following types:

- **motion data** of the target vehicle and its nearby agents, generally represented as time-series of positions and its derivatives on each ax;
- **context data**, of the surrounding environment, often given through images, maps, etc.;
- **high-level data**, which underwent some feature engineering processes, such as the "time-to-collision" between two vehicles.

2.2 Action Prediction formalizations

The first step required to dive into the challenge of predicting the driver's actions consists of providing a definition of the term "action". The literature envisions mainly two branches, occasionally overlapping, well-suiting such a definition: **maneuver** (or intent) and **trajectory**. Furthermore, it is worth mentioning some works involving the forecast of accelerations/decelerations or even the pressure of a driving pedal, by clustering them into a new branch referred to as **driver action prediction**.

In fact, despite being a less explored field, it better matches the level of detail that we aim to achieve in this work and our interpretation of the problem. This

	Overview
Maneuver	Predicting a future action among a predefined set
Trajectory	Forecasting a coordinates sequence
Action	Forecasting accelerations and pedals pressures sequences

Table 2.1: Literature branches

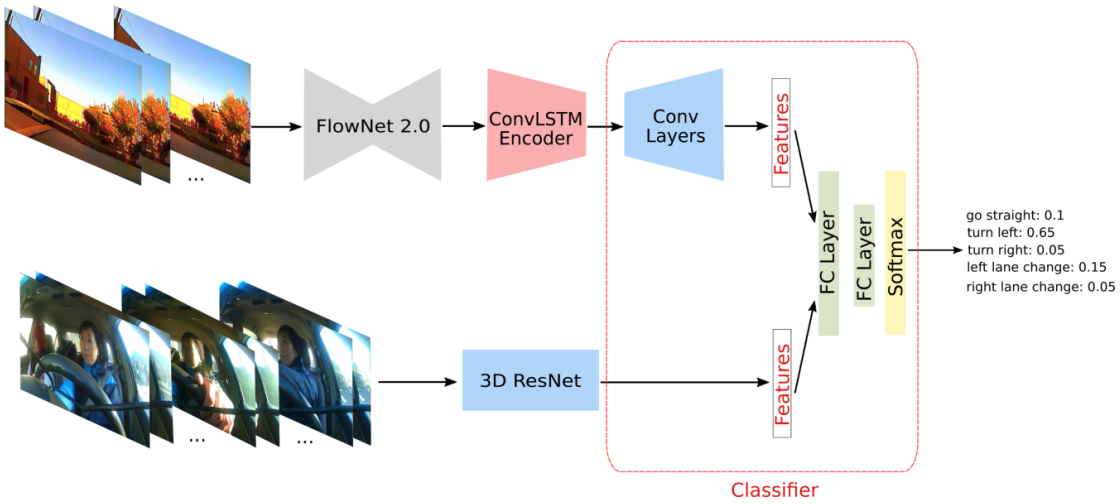

Figure 2.1: An example of architecture employed in the Maneuver Prediction task, presenting as output a discrete set of actions.

section has been structured as an exploration of these ways to intend the problem, summarized in Table 2.1, reporting the major contemporary approaches to tackle it.

2.2.1 Maneuver Prediction

The maneuver prediction problem involves anticipating in a classification fashion an action among a predefined set of driver intentions such as "go straight", "lane change", and so forth. A notable example of a solution is shown in Figure 2.1, with the architecture proposed in [5], which combines the information from the driver monitoring videos with the outside view. In particular, a ConvLSTM [6] based encoder predicts the future outside frame, while a 3D ResNet [7] acquires features from the driver video. Applying convolutional layers to the outside optical flow it is possible to jointly leverage features from both branches for classifying the next maneuver. [8] focuses on the binary problem of predicting a hard brake by exploiting scalar signals from different sensors through a proposed RNN architecture. In particular, signals are divided into categories, and for each one an individual network is trained: the final prediction is taken by the most confident network, namely the one outputting the highest probability score on a label. [3] uses Deep Bidirectional Recurrent Neural Networks (DBRNNs) on sensor data, camera images, and GPS maps, thereby enabling the temporal fusion of both past and future contexts for several

binary problems (Braking Action Prediction, Lane Change Action Prediction, etc.).

2.2.2 Trajectory Prediction

Trajectory prediction consists mainly of the regression problem of forecasting a coordinates sequence. The literature in this field is not limited to the prediction of a vehicle's movement since the subject of the focus often involves pedestrians, more drivers simultaneously, or a mixture of those. This branch collects a major part of the literature efforts related to our research, thereby resulting in a natural field in which to compare the system that we will present throughout this thesis work.

To ease the movement in this line of research we present clusters of similar algorithms. It is important to highlight that the categories into which the works are divided are not orthogonal, in fact, often papers in a paragraph make use of approaches or tools presented in another. Despite its overlapping nature, the structure of this section is aimed to the construction of a narrative in which every paragraph can present several ways in which a key idea or tool can be instantiated.

Model-based methods

Although Deep Learning dominates the literature on trajectory prediction, there exist methods directly exploiting the dynamics of the ego-vehicle. A major approach is **motion model-based** vehicle trajectory prediction, in which the vehicle is simplified as an entity controlled by physics laws, described by the mathematical relationship between the parameters of the movement, such as position, velocity and acceleration [9]. The main motion models [10] include Constant Velocity Model (CV), Constant Acceleration Model (CA), Constant Turn Rate and Velocity Model (CTRV), Constant Turn Rate and Acceleration Model (CTRA), Constant Steering Angle and Velocity Model (CSAV), Constant Steering Angle and Acceleration Model (CSAA). The algorithms in this category can operate in real-time, making them particularly suited for pre-crash warning systems. However, relying on vehicle motion parameters and ignoring the influence of road geometry, motion model-based trajectory prediction methods perform especially badly when the vehicle change its behaviors suddenly: predictions can be accurate and reliable only in the short term, less than one second. To address this, it is possible to first identify a maneuver for then associate a model, an approach referred to as **maneuver-based** vehicle trajectory prediction. Even though the recognition phase of these methods is often associated with Machine Learning methods, it is possible to adopt rule-based methods in place of them, resulting in a whole model-based pipeline. An example is given in [11] in which the similarity between vehicle historic trajectory and lane lines is extracted. A threshold on such a metric establishes if employ models associated with a lane keep maneuver or a lane change one. In fact, this work shows how it is possible to concurrently employ a motion-based model[1] and a maneuver-based model, returning a final estimation combining the ones from the different models.

[1] In this case the CTRA model.

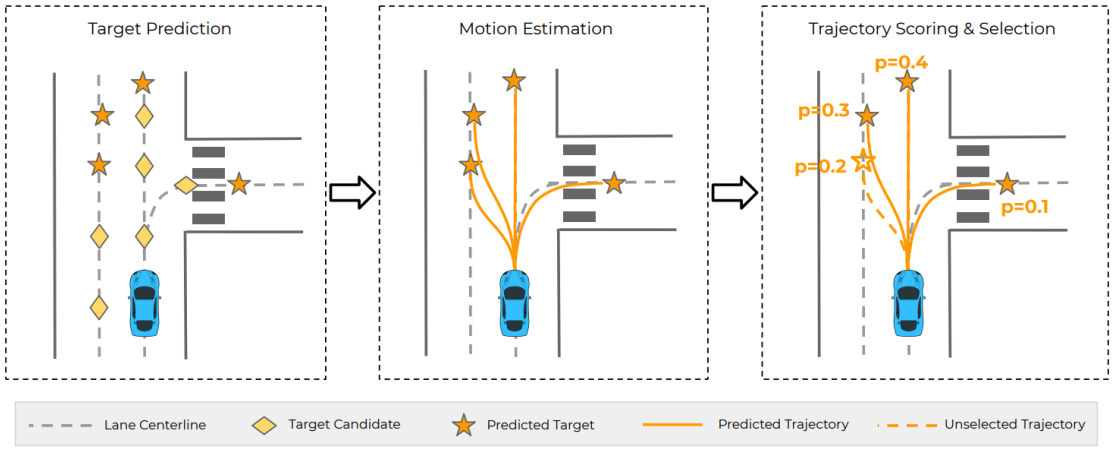

Figure 2.2: The TNT framework.

Direct regression methods

With this paragraph, the review of AI-based methods starts from simple applications that directly tackle the regression problem. [12] trains an LSTM [13] for the future trajectory of a single "target" vehicle using information of ego-vehicle and vehicles immediately around it, consisting of positions, velocities, and estimated times to collisions (TTC). In two similar[2] works [14] and [15] incorporate the maneuver prediction task to predict the successive kinematic states being the execution of the identified maneuver. In particular, the latter system inputs transformed coordinates of the surrounding vehicles to two LSTMs, the first recognizing the high-level lateral intention and providing it to the other, the second outputting future locations in the next 5 seconds. We can envision such a maneuver-oriented framework also in [16], which aims at predicting a future trajectory distribution. There are two output branches, one outputting the maneuvers probabilities, the other outputting time-series composed of the parameters of a Gaussian distribution, conditioned on the input and on the maneuver probability: the product of the parametrized distribution and of the maneuvers distribution returns the overall trajectory distributions conditioned on the input.

Goal-driven methods

A milestone in the literature was introduced in the Target-Driven-Trajectory (TNT) [17] framework in Figure 2.2, suggesting that for prediction within a moderate time horizon, the future modes can be effectively captured by a set of target states.

TNT first predicts an agent's potential target states T steps into the future, by encoding its interactions with the environment and the other agents. Having done that, it generates trajectory state sequences conditioned on targets. A final stage estimates trajectory likelihoods and a final compact set of trajectory predictions is

[2]The main difference being whether to output the anticipated maneuver or just keep it as an intermediate representation.

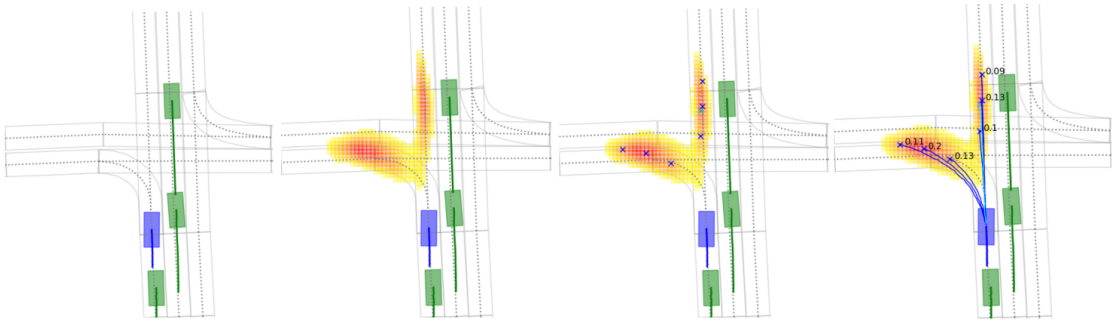

Figure 2.3: Trajectory prediction pipeline of models exploiting probability heatmaps.

selected.

Such focus on trajectory endpoints is increased in HOME [18] and GOHOME [19], two frameworks outputting an image representing the probability distribution of the agent's future location, as exemplified in Figure 2.3. To do this, they employ an output target consisting of an image with a Gaussian centered around the ground truth position. In a second step, they sample from the heatmap a finite number of possible future locations, optimizing specific metrics without retraining the model. Finally, a separate Neural Network builds the full trajectories based on the past history and conditioned on the sampled final points. Furthermore, it is also possible to associate each trajectory a probability computed as the integral of the probability heatmap under the circle of radius 2 m around the endpoint of the trajectory. The differences between HOME and GOHOME derive from the input relative to the environmental context. The former directly employs High Definition maps (HD maps) so that the processing of such information relies on CNNs, while the second extracts a graph from the HD maps, therefore exploiting GNNs) (Graph Neural Networks) to process it. A peculiarity of these frameworks is that probability heatmaps are a great way of representing information coming from different sources or models in a common system of reference and can be averaged together. This allows for these models' concurrent exploitation through model ensembling.
The mentioned exploitation of graph structures in a goal-driven context can lead to state-of-the-art results, as proven by Prediction via Graph-based Policy (PGP) [20]. Here, a graph encoder outputs learned representations for each node of the lane graph, incorporating the HD map and the surrounding agent context. Then a behavior cloning-based module outputs a discrete probability distribution over outgoing edges at each node, allowing for sampling paths in the graph. Such paths are basically sequences of intentions of the driver and represent the longitudinal component of the choice. The longitudinal variability component is inserted through an attention-based trajectory decoder that outputs trajectories conditioned on paths traversed by the policy and a sampled latent variable.

Finally, [21] gives a more comprehensive intention representation with a new recurrent encoder-decoder architecture, Stepwise Goal-Driven Network (SGNet), which predicts goals step-by-step. This work, schematized in Figure 2.4, breaks the simplistic assumption that an agent's intentions are exclusively represented by a single long-term goal, by estimating numerous smaller goals along the way and explicitly including them at each decoder time step. In particular, a stepwise goal

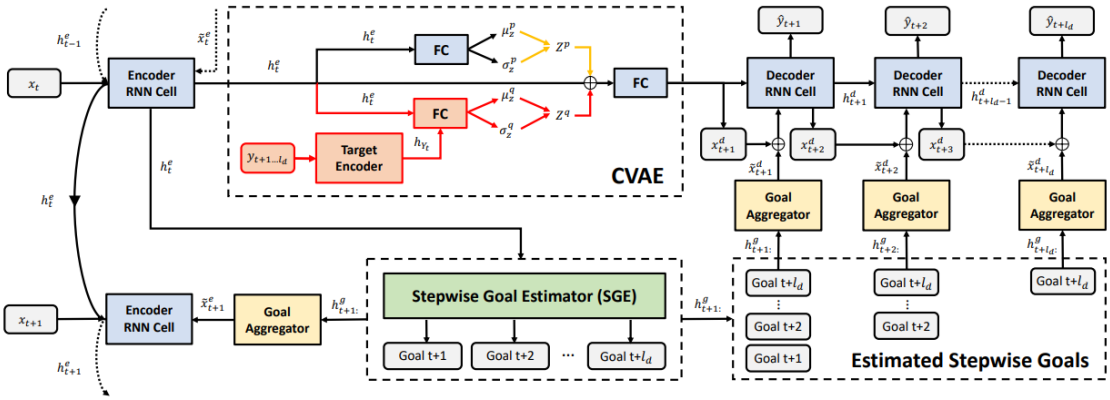

Figure 2.4: SGNet. Arrows in red, yellow, and black respectively indicate connections during training, inference, and both training and inference.

estimator (SGE) predicts an object's future locations as stepwise goals and embeds them as input to the decoder in an incremental manner. SGE also fuses and feeds all predicted goals into the encoder for the next time step, treating historical goals as additional information. To provide a more compact representation, and extract useful information, there are aggregators that adaptively learn the importance of each stepwise goal with an attention mechanism. It is possible to add stochasticity to the framework by sampling a latent variable from a Conditional Variational Autoencoder (CVAE), learning the distribution of the future (output) trajectory conditioned on the observed (input) trajectory.

Inverse Reinforcement Learning methods

Another common tool used for the prediction task under analysis is Inverse Reinforcement Learning (IRL). It is even possible to use it for purposes other than the direct prediction, as [22] does, using it as a regularization method. The proposed model is composed of two modules, trajectory prediction module (TPM) and reward function (RF). TPM is based on an encoder-decoder RNN architecture, where the encoder encodes inputs (past trajectory and scene context information) while the decoder predicts the future trajectory, keeping into account the rewards produced by RF. RF inputs the scene context at time t and the position for time t and outputs a reward. RF does this for both the ground-truth position and the predicted position so that through a loss function comparing the relative rewards it is possible to update the RF module weights.

Differently, [23] exploit IRL for the prediction considering also the interaction element since the distribution over all possible trajectories of the predicted vehicle depends not only on historical information, but also on future plans of other vehicles that interact with it.

To achieve such interaction-aware predictions, first, they explicitly consider the hierarchical trajectory-generation process of human drivers involving both discrete and continuous driving decisions. The discrete driving decisions determine a pattern, whereas the continuous driving decisions influence the details of the trajectory, as exemplified in Figure 2.6. Based on this, the distribution over all future trajectories

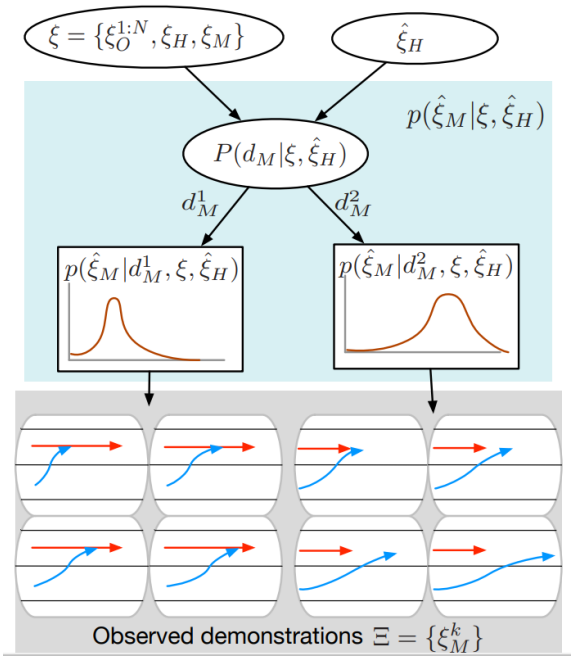

Figure 2.5: The probabilistic and hierarchical trajectory–generation process for a lane-changing scenario. The predicted vehicle (blue) is trying to merge into the lane of the host vehicle (red). With O other vehicles, H host vehicle (the interacting one), M ego-vehicle, given all observed historical trajectories $\xi = \xi_O^{1:N}, \xi_H, \xi_M$ and his belief about the host vehicle's future trajectory $\hat{\xi}_M$, the trajectory distribution of the predicted vehicle over all the trajectory space is partitioned by the discrete decisions: merge behind (d_M^1) and merge front (d_M^2).

of the predicted vehicle is formulated as a mixture of distributions partitioned by discrete decisions. Then they apply IRL hierarchically to learn the distributions from real human demonstrations based on the principle of maximum entropy [24], assuming that all drivers are exponentially more likely to make decisions (both discrete and continuous) that lead to a lower cost. Such a cost function is linearly parametrized by a group of pre-selected features whose weights are optimized so that the given demonstration set is most likely to happen. In a similar work, [25] even tests personalized modeling showing that outperforms the general modeling method.

Finally, P2T is capable of state-of-the-art results by using IRL in one that we could loosely see as a goal-driven approach. In particular, P2T [26] learns rewards and a policy to plan on a 2D grid, in the form of sequences of positions. Then, through an attention-based trajectory generator it outputs continuous valued trajectories[3] conditioned on the sampled plans.

[3]Agent locations without assigned times are referred to as paths, whereas agent locations with assigned times as trajectories.

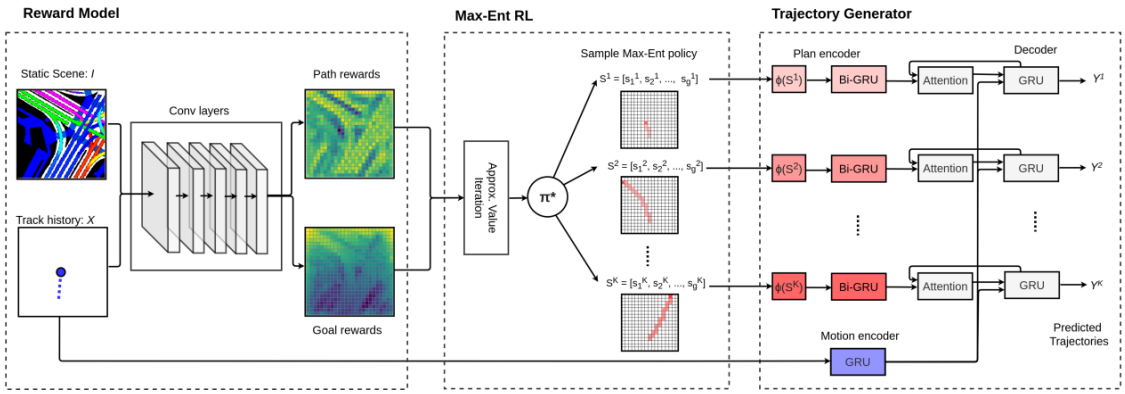

Figure 2.6: P2T: plans to trajectories.

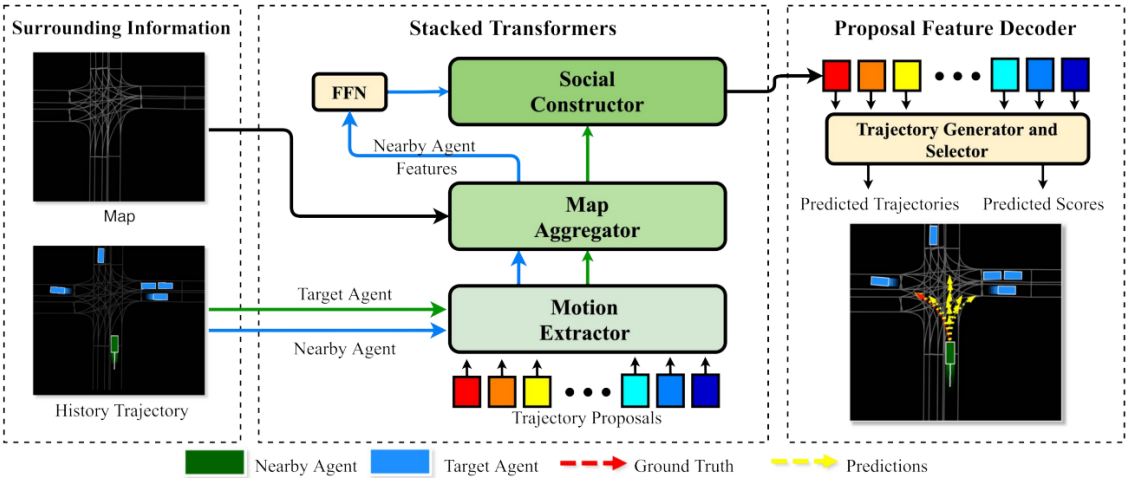

Figure 2.7: MultiModal Transformer framework.

Proposal-based methods

Proposal-based approaches first define candidate points or trajectories as proposals, and then regress or classify these proposals to the ground truth. With predefined proposals, these methods alleviate the optimization burden and narrow down the feasible space of solutions, even though the results depend on heuristic methods applied to sample the candidate points. A recent example is MultiModal Transformer [27], in which the proposals are first randomly initialized and then refined to incorporate contextual information. mmTransformer is designed based on the transformer [28] architecture, particularly effective in modeling sequential data. The whole model (Figure 2.7) can be viewed as stacked transformers in which the past trajectories, the road information, and the social interaction are aggregated hierarchically with several transformer encoder-decoder modules.

2.2.3 Driver Action Prediction

As [4] highlights a driver's pedal behavior model has the potential to be used in various vehicle powertrain model predictive control applications. However, modeling

the driver pedal behavior is challenging because the pedal position is the output of a complicated virtual system, which involves human sensing, decision-making, and body movement processes. One simple alternative to driver pedal behavior models is the vehicle speed model. In these models, the torque demand of the vehicle can only be calculated by taking derivatives of the modeled vehicle speed. As the torque demand is a key factor in the powertrain control systems, it is more desirable to have its direct and accurate prediction that may not be provided by these speed models. The solution proposed by [4] considers the pedal behavior of a given driver on a given vehicle as a sequence of actions, intended as adjustments of the pedal position. Vehicle speed and acceleration, and road information that a driver receives is considered as the inputs to the model, while the probabilities of pedal actions are the outputs of the model. This driver pedals behavior model is depicted under the input-output hidden Markov model (IOHMM) [29] framework, with states standing for the modes of driving behaviors, (e.g., brake for a stop, free acceleration, or cruise, etc), determining the mapping between the input that a driver receives and the probabilities of the action. The proposed IOHMM-based model can only provide the output action probability distribution for one step. To be able to make multistep predictions within one step a pedal position is sampled from the output distribution and then fed into a vehicle-road model, providing the inputs to each time-step of the prediction window.

Moving to AI-based systems, [30] implements LSTMs taking traffic state information (velocities, accelerations, distances, etc.) as input to output a Gaussian mixture that models a distribution of future longitudinal acceleration values in the next timestep. At each timestep, accelerations are sampled from such distributions, and the input state for the next timestep is calculated accordingly.
[31] proposes a framework consisting of undirected graphs representing the interactions between vehicles, a graph convolution neural network [32] used to encode the graph structure directly, and a fully-connected or LSTM mixture density network used to predict future acceleration distributions. More in detail, also in this case the outputs of the ANN are Gaussian mixture model parameters that characterize the future acceleration distribution. Commonalities do not stop here, indeed, here as in all the methods overviewed in this paragraph is envisioned the practice of outputting a time-series of predictions by updating input parameters of the time t+1, basing on a sampled acceleration[4] at time t. Such a technique has the advantage of being particularly flexible, since we do not have to fix the output window length in advance, but we can unroll the prediction until desired at test time. However, it needs the availability of a model to update the inputs according to the outputs at the previous time-step, therefore being limited to simple scenarios.

2.3 Industrial requirements

We defined the cluster of works in Section 2.2.3 as the "driver action prediction" branch, because of its similarity in the way we intended the problem. Indeed, the

[4]Meant as pedal pressure, actual dynamic, and also accounting for a deceleration if negative.

purpose of this work is to develop a system capable of inferring a sequence of driver commands, namely **accelerator and brake pedal pressure and steering angle**. Furthermore, the proposed methodology is born with the prospect of future industrialization. This can imply the **non-availability of complex sources of data**, such as densely annotated maps exploited by some of the best-performing SOTA models. As will be explained in Section 4, the proposed approach addresses the problem under analysis by providing a **non-intrusive solution**, exploiting the signals coming from the current **onboard technology** (e.g. ADAS systems) and from the **vehicle dynamics**. The software-based solution proposed has the advantage of not requiring the implementation of other instrumentation in the vehicle, no cameras or biometric tracking systems. It has a low computational cost and does not require modifications to the passenger compartment. This could be relevant, especially, in the luxury car market, where attention to design is an important element of this type of vehicle. In fact, the objectives that drive the development can be summarized as follows:

- compared to the more pursued objective of trajectory prediction, the granularity of predictions is augmented, and indeed it will be possible to confront on a common field by computing the trajectory derived from the application of the forecast driving commands;
- since a digital twin of the driver is instantiated, in the introduced framework the meaning of "action" is arbitrarily extendable, as long as the action can be simulated and produce an effect on the car;
- the prediction system proposed is meant to be integrated into the vehicle's ECU (Engine Control Unit), it does not require any additional instrumentation that distorts the car's interior design, and to be strictly adhering to the principles of privacy and data protection.

2.4 Materials and Tools

In this section, we present the sources of data utilized or utilizable for developing and/or benchmarking the prediction systems presented in Sections 2.2.1, 2.2.2, 2.2.3, and for the system at the core of this thesis work.

2.4.1 Datasets

The following is a list of datasets commonly used for the research branches introduced. As mentioned before, Trajectory Prediction can focus on objects other than vehicles, and because of that, some pedestrian datasets are mentioned as well.

nuScenes dataset

The nuScenes [33] dataset is a large-scale autonomous driving dataset. The dataset has 3D bounding boxes for 1000 scenes collected in Boston and Singapore. Each scene (Figure 2.8) is 20 seconds long and annotated at 2Hz. This results in a total of

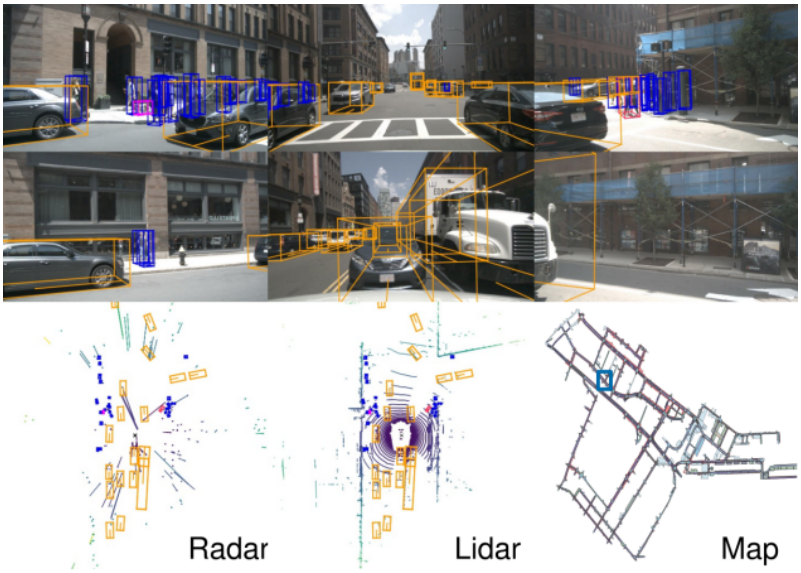

Figure 2.8: An example from the nuScenes dataset

Figure 2.9: An example from the ETH dataset

28130 samples for training, 6019 samples for validation and 6008 samples for testing. The dataset has the full autonomous vehicle data suite: 32-beam LiDAR, 6 cameras and radars with complete 360° coverage.

ETH dataset

ETH [34] is a dataset for pedestrian detection. The testing set contains 1,804 images (Figure 2.9) in three video clips. The dataset is captured from a stereo rig mounted on car, with a resolution of 640 x 480, and a framerate of 13-14 FPS.

UCY dataset

The UCY [35] dataset consists of real pedestrians' trajectories with rich multi-human interaction scenarios captured at 2.5 Hz. It is composed of sequences taken in public spaces from top view, as shown in Figure 2.10.

Figure 2.10: An example from the UCY dataset

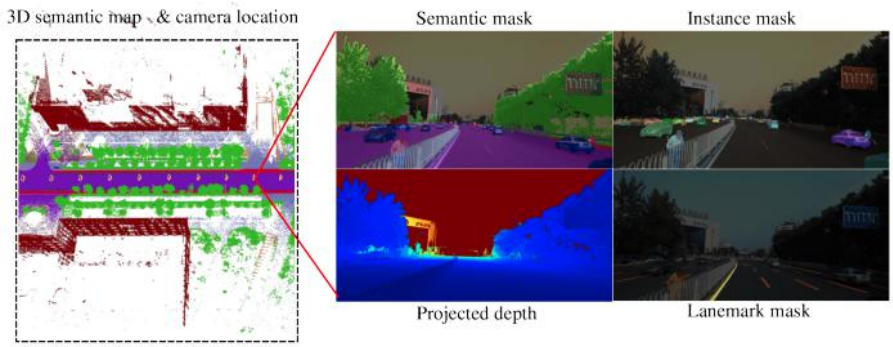

Figure 2.11: An example from the ApolloScape dataset

ApolloScape dataset

ApolloScape [36] is a large dataset consisting of over 140,000 video frames (73 street scene videos) from various locations in China under varying weather conditions. Pixel-wise semantic annotation of the recorded data is provided in 2D, with point-wise semantic annotation in 3D for 28 classes. In addition, the dataset contains lane marking annotations in 2D. An example from this dataset is reported in Figure 2.11.

INTERACTION dataset

The INTERACTION dataset contains naturalistic motions of various traffic participants in a variety of highly interactive driving scenarios from different countries. Examples from this dataset are reported in Figure 2.12

HDD dataset

Honda Research Institute Driving Dataset (HDD) [37] is a dataset to enable research on learning driver behavior in real-life environments. The dataset, exemplified in Figure 2.13, includes 104 hours of real human driving in the San Francisco Bay Area collected using an instrumented vehicle equipped with different sensors.

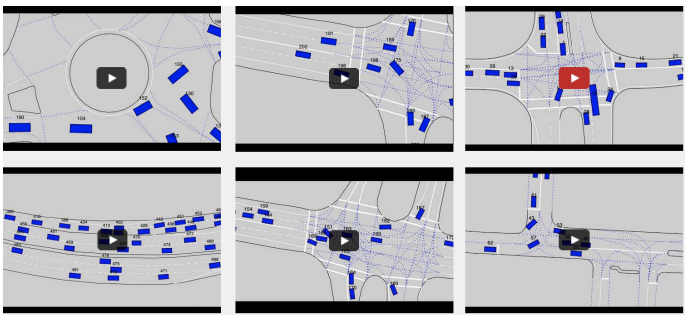

Figure 2.12: Examples from the INTERACTION dataset

Figure 2.13: Examples from the HDD dataset

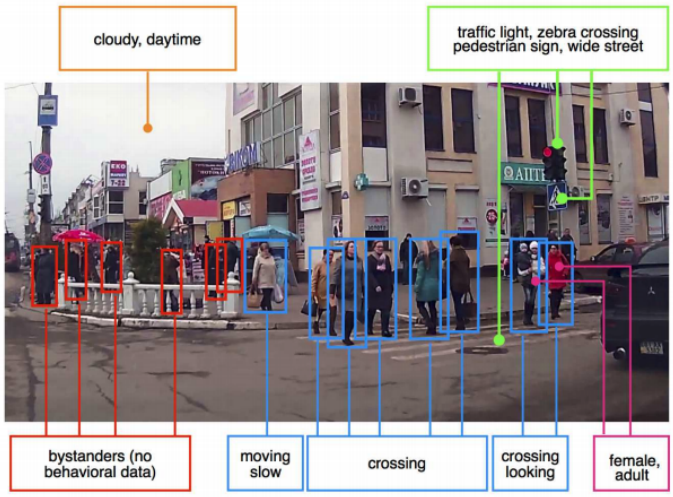

Figure 2.14: An example from the JAAD dataset

JAAD dataset

Joint Attention in Autonomous Driving (JAAD) [38] is a dataset for studying joint attention in the context of autonomous driving. The focus is on pedestrian and driver behaviors at the point of crossing and on the factors that influence them. To this end, JAAD dataset provides a richly annotated collection of 346 short video clips (5-10 sec long) extracted from over 240 hours of driving footage, as shown in Figure 2.14.

Behavior annotations specify behaviors for pedestrians that interact with or require the attention of the driver. For each video, there are several tags (weather, locations, etc.) and timestamped behavior labels from a fixed list (e.g. stopped, walking, looking, etc.). In addition, each frame presents a list of demographic attributes is provided for each pedestrian (e.g. age, gender, direction of motion, etc.) as well as a list of visible traffic scene elements (e.g. stop sign, traffic signal, etc.).

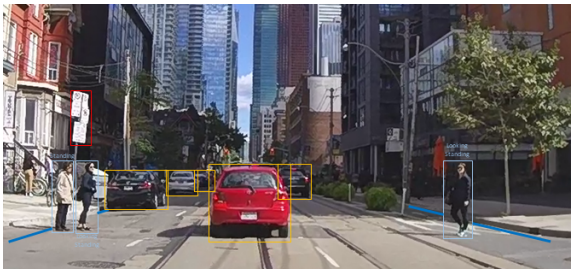

Figure 2.15: An example from the PIE dataset

TITAN dataset

TITAN [39] consists of 700 labeled video clips captured from a moving vehicle on highly interactive urban traffic scenes in Tokyo. The dataset includes 50 labels including vehicle states and actions, pedestrian age groups, and targeted pedestrian action attributes.

PIE dataset

PIE [40] is a new dataset for studying pedestrian behavior in traffic. PIE contains over 6 hours of footage recorded in typical traffic scenes with on-board camera. It also provides accurate vehicle information from OBD sensor (vehicle speed, heading direction and GPS coordinates) synchronized with video footage. Rich spatial and behavioral annotations are available for pedestrians and vehicles that potentially interact with the ego-vehicle as well as for the relevant elements of infrastructure. There are over 300K labeled video frames (Figure 2.15) with 1842 pedestrian samples making this the largest publicly available dataset for studying pedestrian behavior in traffic.

2.4.2 Simulation Environments

Different driving simulation environments and models are examined below, whose exploitation can allow for research in the branches reviewed.

Complete Vehicle Model

The Complete Vehicle Model is a Simulink model for emulating a complete vehicle, namely its powertrain, driveline, and dynamics [41].

The main subsystems and transmission components, shown in Figure 2.16, are:

- Driver inputs: throttle/brake profiles;
- Engine: system-level model of spark-ignition and diesel engine;
- Torque Converter: three-part torque converter consisting of an impeller, turbine, and stator;
- Transmission subsystem: CR-CR 4-speed transmission;

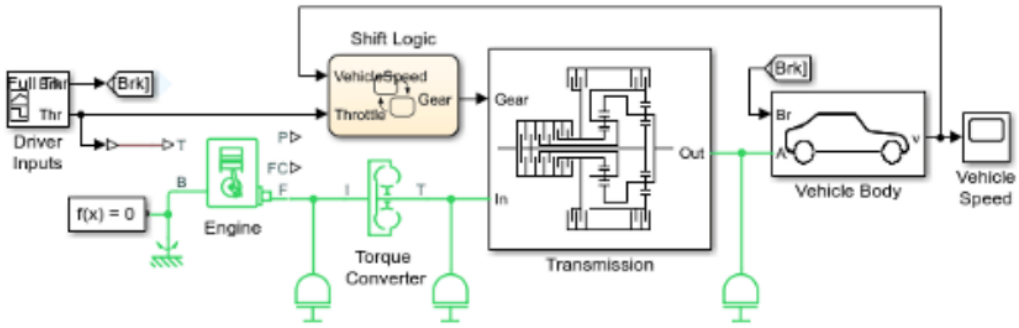

Figure 2.16: Complete Vehicle Model

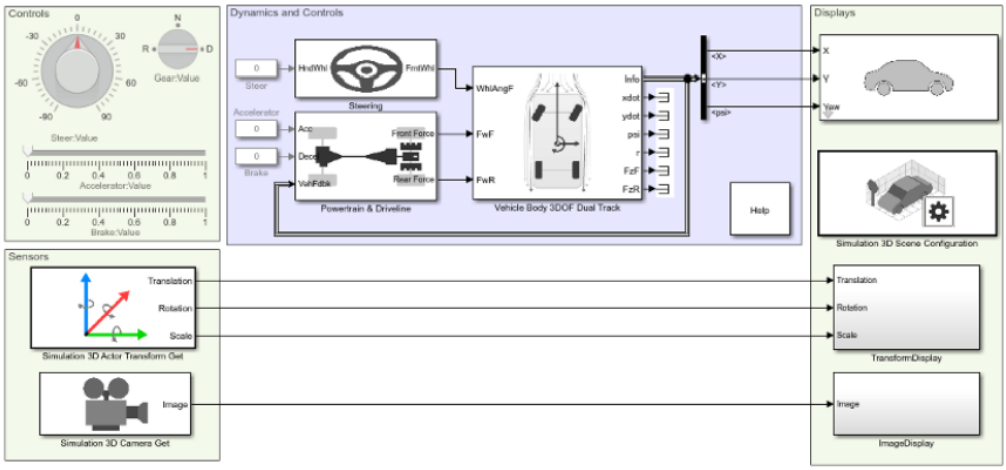

Figure 2.17: Scene Interrogation in 3D Environment

- Shift Logic: Stateflow® implemented transmission controller;
- Bodywork: Vehicle, tire, and brake dynamics.

Although it is open for customization and well-documented, it does not accept steering movement as input but only throttle and brake signals to the engine and transmission control system.

Scene Interrogation in 3D Environment

The scene interrogation in 3D environment is a Simulink model for emulating a complete vehicle (powertrain, driveline, and dynamics) linked with a 3D visual representation with the Unreal Engine [42]. As shown in Figure 2.17, the scene interrogation with camera and ray tracing reference application contains:

- A passenger vehicle with a simple driveline, combined slip wheel, and 3DOF vehicle dynamics model;
- A camera mounted on the rear-view mirror of the passenger vehicle;
- Steering, acceleration, gear, and braking controls;

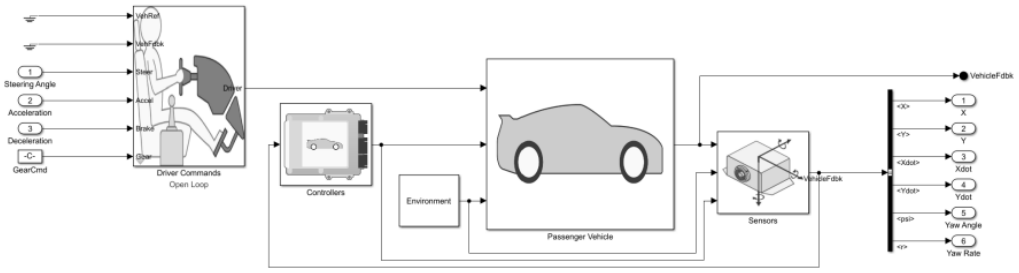

Figure 2.18: Conventional Vehicle Reference Application

- Vehicle light controls;
- 3D viewing environment configured for the Virtual Mcity scene.

The downside of this simulator is that it requires the use of a GPU for 3D rendering.

Conventional Vehicle Reference Application

The Conventional Vehicle Reference Application [43] is an open-source Simulink model that emulates a complete vehicle model with internal combustion engine, transmission, and associated control algorithms. The block set includes a library of components for modeling propulsion, steering, suspension, vehicle body, brakes, and tires. It is often used for powertrain matching and component selection analysis, control algorithm design and diagnostics, and hardware-in-the-loop (HIL) testing.

As shown in Figure 2.18, the main subsystems are:

- Driver commands – Steering, throttle, brake, and gear (optional);
- Passenger Vehicle - It implements a vehicle that contains transmission and engine subsystems;
- Controllers' subsystem - It implements a powertrain control module (PCM) containing a transmission control module (TCM) and an engine control module (ECM);
- Environment subsystem - It creates environmental variables, including road slope, wind speed, temperature, and ambient pressure;
- Sensor's subsystem - An inertial measurement unit (IMU), a sensor assembly consisting of a three-axis accelerometer that measures acceleration and a three-axis gyroscope that measures angular velocity.

This model allows testing the vehicle with standard driving maneuvers, such as a double lane change, or as part of customized scenarios. It is open for customization and well-documented. The downside is that it does not have a comprehensive and useful 2D visualization of vehicle behavior.

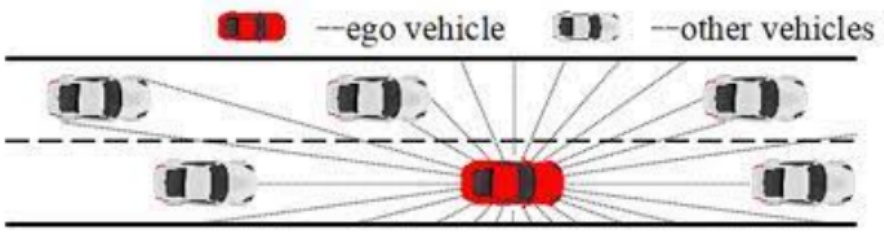

Figure 2.19: Automated Lane-Change Manoeuvring simulation environment

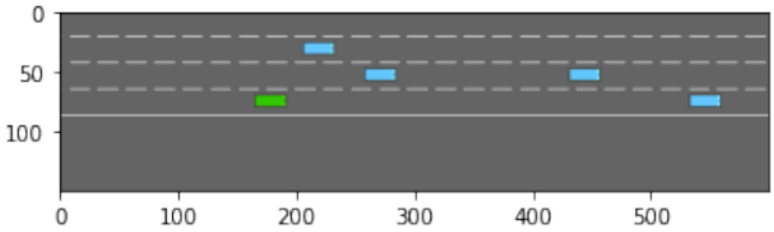

Figure 2.20: Highway Gym simulation environment

Automated Lane-Change Manoeuvring

The Automated Lane-Change Manoeuvring is a 2D visual simulation environment for a problem of Lane-Changing [44]. It is developed with Pyglet, a powerful and easy-to-use Python library for game development. The simulation is a simplified traffic flow scene, exemplified through Figure 2.19. It consists of the lane line, the lane boundary line, and the vehicles in each lane. The agent can observe its own relative position through the distance state information between itself and the other vehicles. It can control the steering wheel angle, throttle opening, and brake pedal pressure. The positive aspects of this simulator are that it has easy and intuitive graphics and does not require the use of a GPU. The downside of this tool is the impossibility to access its source code.

Highway Gym

The Highway Gym is an open-source 2D visual simulation environment [45]. It is developed with the gym library for problems of Lane-Keeping or Lane-Changing. The agent drives on a multilane highway populated with other vehicles, as exemplified in figure 2.20. Its goal is to reach a high speed while avoiding collisions with neighboring vehicles. The agent can perform longitudinal (speed changes) and lateral (lane changes) actions. It is possible to add road junctions with oncoming vehicles, roundabouts with flowing traffic and intersections with intense traffic.

Udacity's Self-Driving Car Simulator

The Udacity's Self-Driving Car (Figure 2.21) is an open-source 3D visual simulation environment developed on the Unity game platform [46]. It allows to manually drive a car to generate training data (image data, steering angles, and acceleration

Figure 2.21: Udacity's Self-Driving Car Simulator

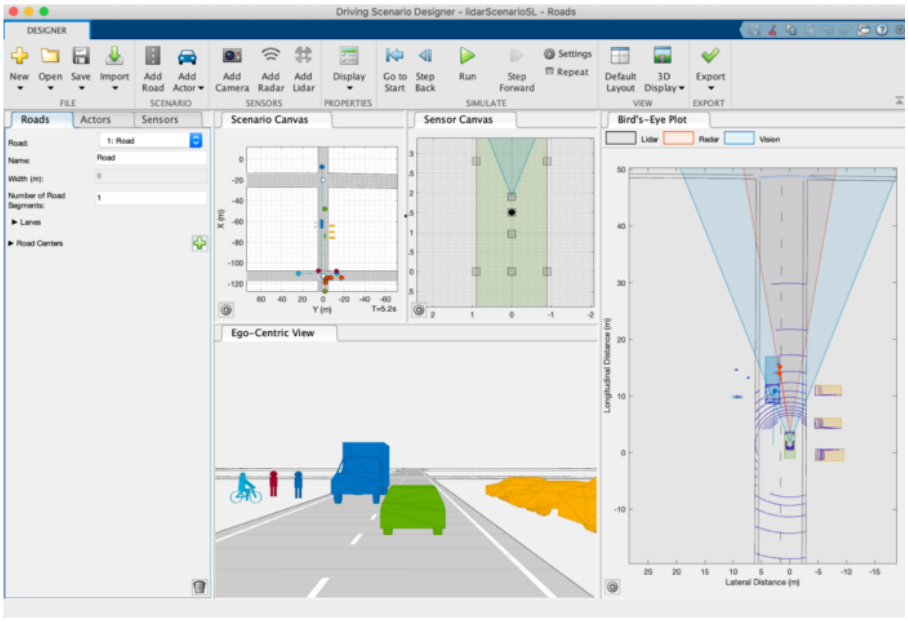

Figure 2.22: Driving Scenario Designer

throttle) or to test a machine learning model. The disadvantage of this simulator is that it offers only basic vehicle dynamics.

Driving Scenario Designer

The Driving Scenario Designer application [47] is a MathWorks tool and enables the design of synthetic driving scenarios for testing autonomous driving systems. With this tool, of which we show a visualization in Figure 2.22, it is possible to:

- create models of roads and actors (vehicles and pedestrians) using a drag-anddrop interface;
- configure vision, radar, lidar, INS and ultrasonic sensors mounted on the ego vehicle.

Figure 2.23: AVSimulation

- import road data from OpenStreetMap into a driving scenario;
- export synthetic sensor readings into MATLAB;
- generate MATLAB code of the scenario and sensors, then programmatically edit the scenario and import it back into the application for further simulations;
- generate a Simulink model from the scenario and sensors and use the generated models to test the sensor fusion or vehicle control algorithms.

It does not require high hardware resources, is easily configurable and can be connected to a vehicle model. It also features 3D and 2D visualization of the vehicle and sensors and an integrated Unreal Engine.

AVSimulation

AVSimulation [48] is a complete 3D driving simulation software allowing scenario modification and sensors (RADAR, LIDAR, cameras) and vehicle modeling at the physical level. It makes it possible to develop and validate ADAS vehicles, both autonomous and connected on any type of platform (real-time, cloud, PC) or to perform human behavior studies on simulators. The simulator, whose Figure 2.23 is an example, provides digital resources (urban plans, buildings, vehicles) and a flexible API python that allows controlling all aspects related to the simulation, such as static and dynamic actors, traffic scenarios and environmental properties (weather, conditions, illumination etc.). The positive aspects of AVSimulation are that it offers many ADAS and Sensor systems and an excellent personalization of the environment, but the downside is that it is not open source.

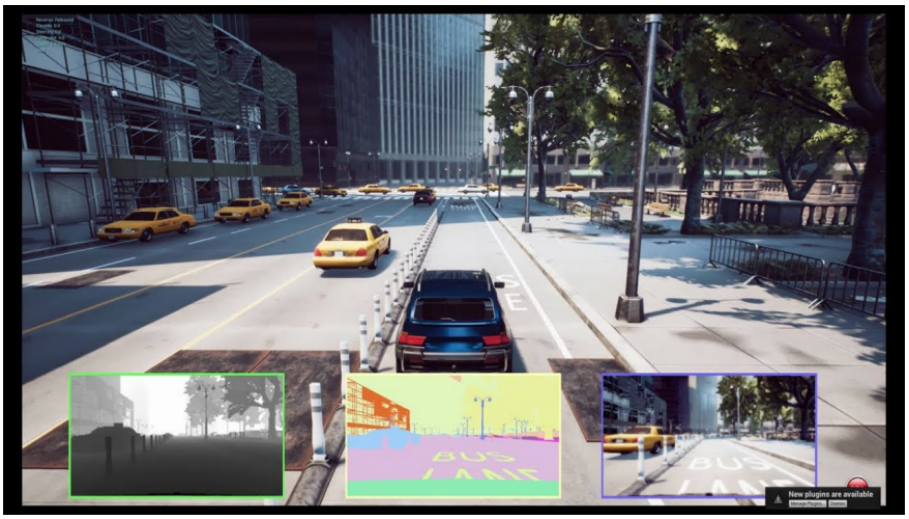

Figure 2.24: AirSim Simulator

AirSim

The AirSim (Figure 2.24) is an open-source 3D visual simulation environment developed by Microsoft [49]. It is a simulator for drones, cars and more, built on Unreal Engine and on Unity game platform. The goal is to develop a platform for AI analysis for autonomous vehicles to experiment with deep learning and reinforcement learning algorithms. AirSim's APIs allow to interact with the vehicle in simulation, retrieve useful images and vehicle variables as data to train models for deep learning, control the vehicle, get state, and so on. AirSim allows to set throttle, steering, handbrake, and gear, and to retrieve the state vehicle information, such as speed and the 6 kinematics quantities (position, orientation, linear and angular velocity, linear and angular acceleration). It also supports some vehicle sensors: camera, barometer, IMU (inertial measurement unit), GPS, magnetometer, distance sensor and LiDAR.

TORCS

The Open Racing Car Simulator (TORCS) is an open-source 3D driving simulator based on OpenGL [50]. It is a highly portable multi-platform car racing simulation with a sophisticated physical model easy to modify. It is used as an ordinary car racing game and as a research platform. The purpose of the simulator is to develop artificial intelligence for vehicle control. TORCS simulates the main components of the vehicle dynamic, such as mass, rotational inertia, links and differentials, friction, collision, suspensions, and aerodynamics. The 3D rendering, shown in Figure 2.25, is lightweight, and it can be turned off for faster training. The downside of this simulator is that it lacks in the complexity of urban driving, such as pedestrians, intersections, cross traffic, and other characteristics of urban environment.

Figure 2.25: TORCS Simulator

Figure 2.26: CARLA Simulator

CARLA Simulator

CARLA is an acronym of Car Learning to Act [51, 52]. It is an open-source 3D visual simulation environment for urban driving, as shown in Figure 2.26, based on Unreal Engine. It is developed by European Commission to support autonomous driving research and ADAS testing. The simulator provides open digital resources (urban plans, buildings, vehicles) and a flexible API python that allows controlling all aspects related to the simulation, such as static and dynamic actors, traffic scenarios and environmental properties (weather, conditions, illumination etc.). The commands that control the vehicle are steering, acceleration and braking. CARLA also supports a flexible configuration of sensor suites (LIDARs, multiple cameras, depth sensors, GPS etc.). The positive aspects of CARLA are that it offers many ADAS systems and excellent customization of the environment, but the downside is that it provides few variables on the dynamics of the vehicle.

Parameters	Var	Units	Value
Vehicle mass	m	[kg]	1181
Longitudinal distance from center of mass to front axle	a	[m]	1.515
Longitudinal distance from center of mass to rear axle	b	[m]	1.504
Vertical distance from center of mass to axle plane	h	[m]	0.134
Track width	w	[m]	1.563
Gears		[adims]	6
Driveline Model			Rear Wheel Drive Active Differential

Table 2.2: Vehicle Specifics

2.4.3 SENSOR Driveline environment

The actual simulation environment choice consists of an asset of SENSOR, which is built upon some of the previously mentioned simulators and has the following key features:

- it is developed by implementing various tools from MATLAB and Simulink;
- it has interfaces with the python library OpenAI gym [53], commonly exploited for Reinforcement Learning;
- it reproduces vehicle behavior and driving conditions and is shaped to allow training on a lane-keeping task.

Vehicle model

The MATLAB implemented vehicle model, deepened in Section 3.1.3, is a 14 degrees-of-freedom (DOF) model and provides a series of signals on the dynamics of its components. The vehicle model is configured according to the parameters defined in Table 2.2, and represented in Figure 2.27.

In addition, it is simulated a sensor on the vehicle, configured according to the specifications defined in Table 2.3. It is a camera that provides information on road signs, such as the distance of the vehicle to the lane lines or the curvature of the road.

Road scenario

The road scenario is developed with Driving Scenario Designer, reviewed in Section 2.4.2. It allows for randomly generating different types of road scenarios, by employing an algorithm designing the layout of the road varying:

- the lane width;
- the number of curves;

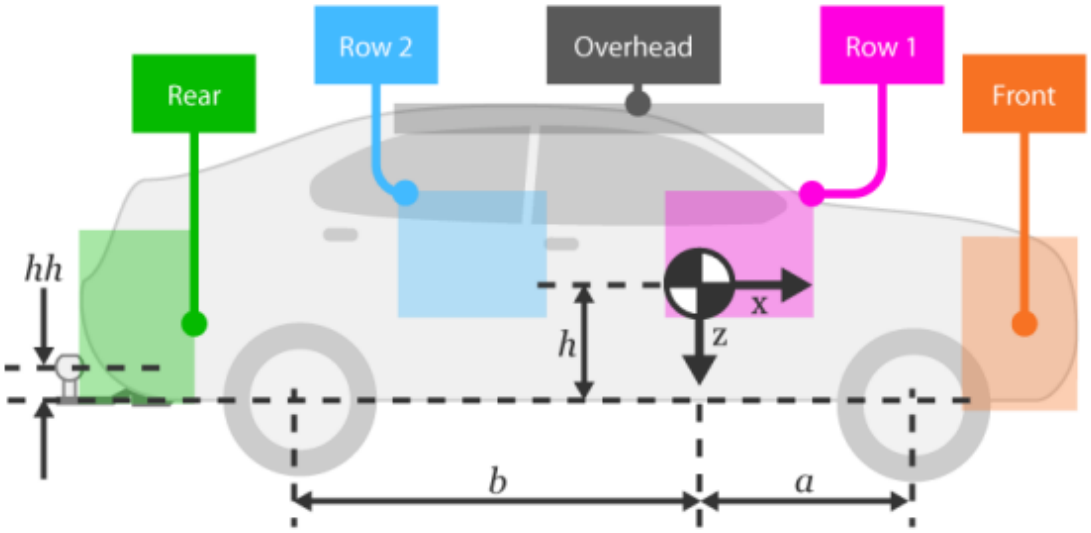

Figure 2.27: The load locations and vehicle parameter dimensions

Parameters	**Units**	**Value**
Sensor's (x, y) position	[m]	[1.9, 0]
Sensor's height	[m]	1.1
Pitch angle of sensor mounted	[deg]	1
Maximum detection range	[m]	150
Focal length	[pixel]	[800, 800]
Image size produced by the camera	[pixel]	[480, 640]

Table 2.3: Camera specifics

- the radius of curvature.

Based on these three variables, the difficulty of the scenario is defined. If the random generation is active, the road generated are always single-lane (Figure 2.28) and do not include any intersections or other road users, such as pedestrians or other vehicles.

Road difficulty level

In the road scenario, the complexity factor is the route layout. The random generation algorithm follows a track rating scale, defining 10 levels of difficulty based on the lane width, the number of curves, and the radius of the curvature. Each of these factors is given a weight, whose maximum value is reported in Table 2.4, according to the importance this variable has in defining the level of difficulty, and the sum of these weights give the difficulty level of the track.

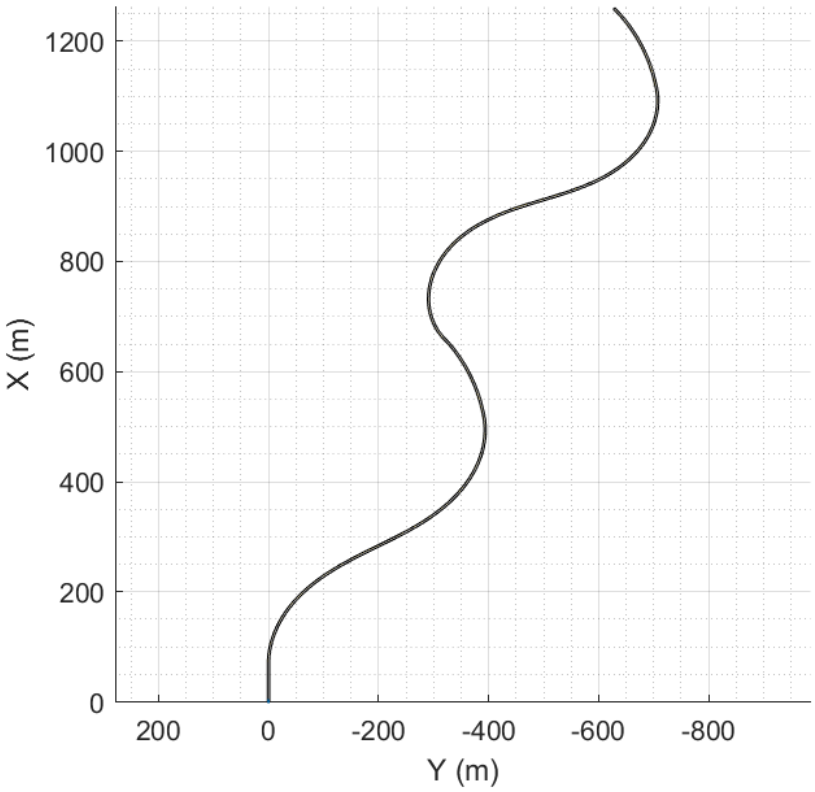

Figure 2.28: An example of road track randomly generated.

Factors	**Units**	**Maximum value**
Width Lane	[m]	4
Number of curves	[adims]	2
Radius of curvature	[m]	4

Table 2.4: Maximum weight attributed to each factor

Graphic User Interface

Considering that the outlined simulator is meant to be utilized for Reinforcement Learning frameworks, we highlight that for this learning paradigm, unlike in other Machine Learning techniques, it is also important to have a visualization of the newly trained agent to evaluate its behavior in a dimension that is not only numerical but also qualitative. Such an intuition of its goodness is provided by the **Graphic User Interface (GUI)** shown in Figure 2.29.

The visualization can be divided into three parts:

- the top left is the ego view of the vehicle and provides information on the actions taken and the behavior of the vehicle, as well as the instant reward and total reward earned;
- on the right, the first half shows the physical variables of the vehicle and the

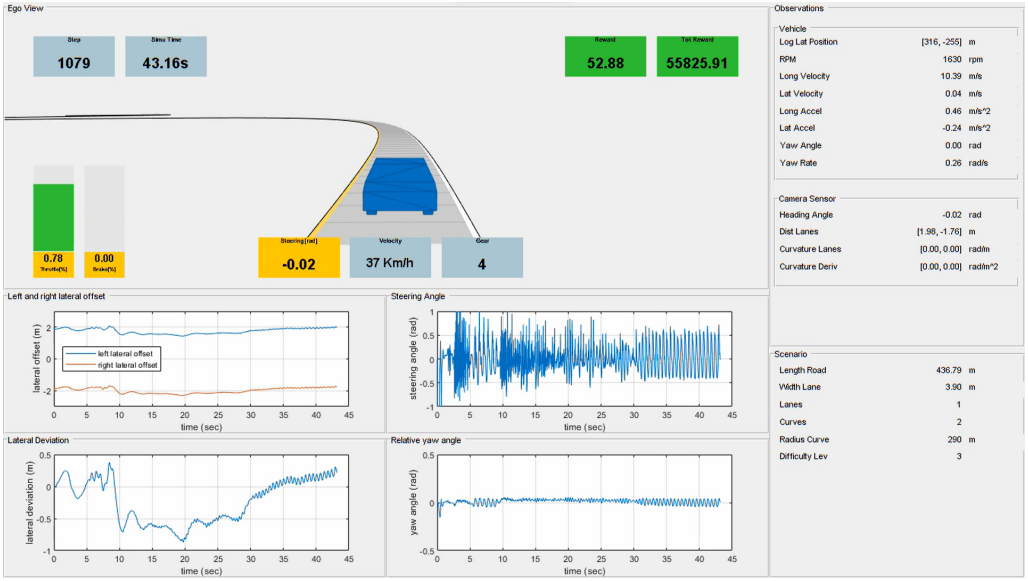

Figure 2.29: Graphical interface of agent behavior.

sensors, the input of the neural network. While in the second half we have information about the scenario on which the agent is validating or testing;

- at the bottom, there is the time evolution of some physical signals useful for the general understanding of the vehicle behavior, such as the distance to the lane line or the steering progression.

Agent-Environment interaction

The MATLAB environment is used for creating an OpenAI gym environment, envisioning a **lane-keeping** task. The agent perceives its surroundings through its dynamics and sensors and modifies its state by performing certain actions. As shown in Figure 2.30, a Python "Orchestrator" component handles communication between the MATLAB and Python sides of the environment. The decision-making and all the Reinforcement Learning part is left to Python as well, where the training can be performed through stable-baselines [54] library, or, after this thesis work, with MBRL-lib [54] library.

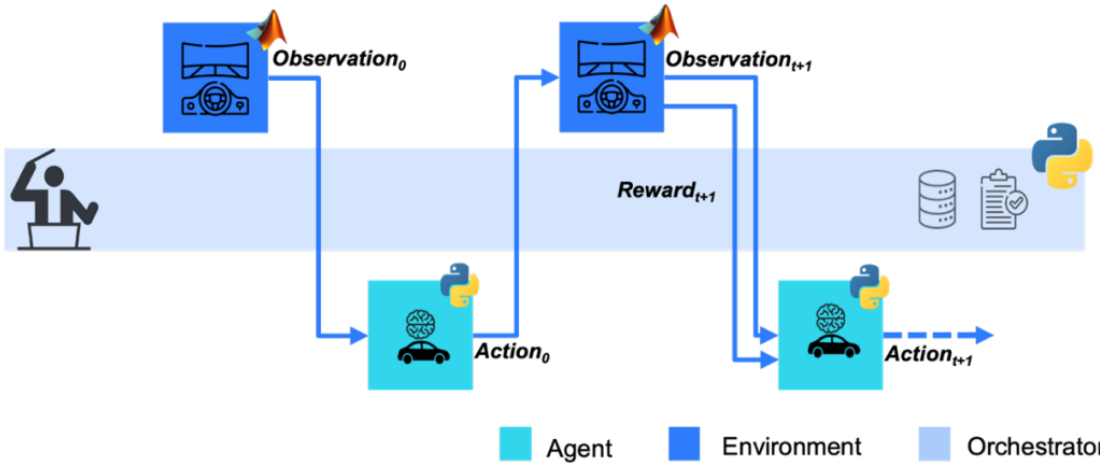

Figure 2.30: The interaction between the MATLAB environment, and its wrapped Python version, where Reinforcement Learning is performed.

Chapter 3

Background

This chapter presents the previous knowledge, tools, and algorithms exploited in this study.

3.1 Vehicles variables

The behavior of a vehicle is the result of the action of interactions (Figure 3.1) that have different origins and points of application. Some, such as weight and inertia, are proper to the body itself, while others are induced by the driver through the controls, and others come from outside.

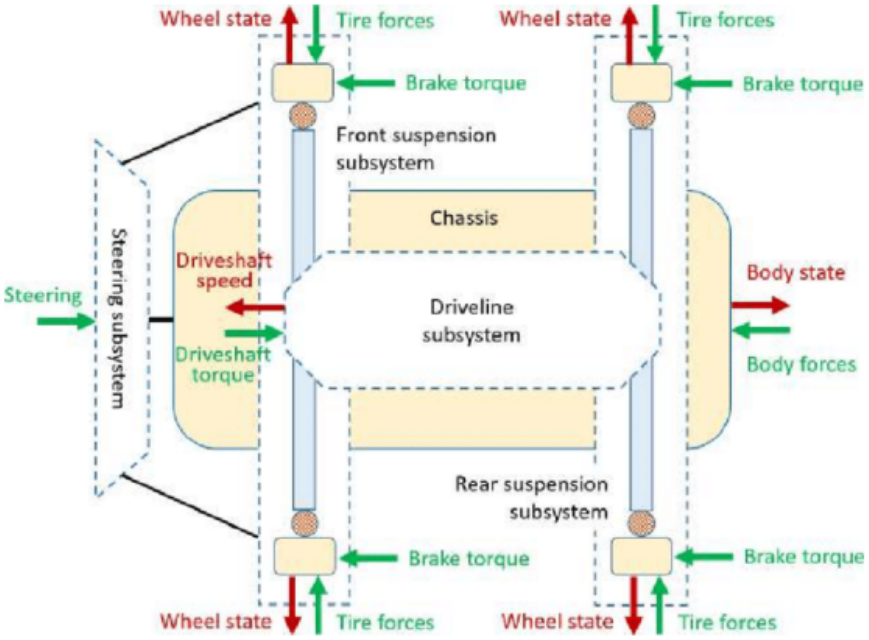

Figure 3.1: Decomposition of a wheeled vehicle into subsystems

3.1.1 Driving actions

The driver is in charge of observing the external environment and conducting the vehicle according to needs and demands. The interface between the driver and the vehicle is fixed and it comes down to a few main elements:

- steering wheel, determining the vehicle steering angle, according to the transmission ratio, and thus responsible for the lateral control;
- throttle pedal, that through pressure determines a positive longitudinal acceleration of the vehicle;
- brake pedal, that through pressure determines a negative longitudinal acceleration of the vehicle;
- gear lever, not always present, (e.g. for vehicles equipped with automatic transmission the selected gear is not controllable by the driver).

3.1.2 Powertrain and drivetrain

The **powertrain** consists of the engine and all the components that generate a propulsive drive in the vehicle. On the other hand, the **drivetrain** includes the transmission and the differential.
Those components are not under the direct control of the driver, but they respond to its action and can provide useful information about the way the vehicle is used. Further insightful signals to consider may be engine RPM (Rotation Per Minute), engine instant torque, generated power, and selected gear.

3.1.3 Vehicle dynamics

The dynamics of a body consists of the forces that act on it and all the resulting accelerations (i.e. along the 3-dimensional components).

Vehicle model

To study vehicle behavior in the longitudinal, lateral, and vertical directions we used the 14 degrees-of-freedom (DOF) vehicle model shown in Figure 3.2. It is composed of an elastic vehicle body mass and four non-elastic wheel masses:

- the suspended mass has 6 DOF and it includes longitudinal, lateral, vertical, roll, pitch, and yaw motion;
- each of the wheels has 2 DOF, which consist of vertical movement of the wheel and rotation of the wheel.

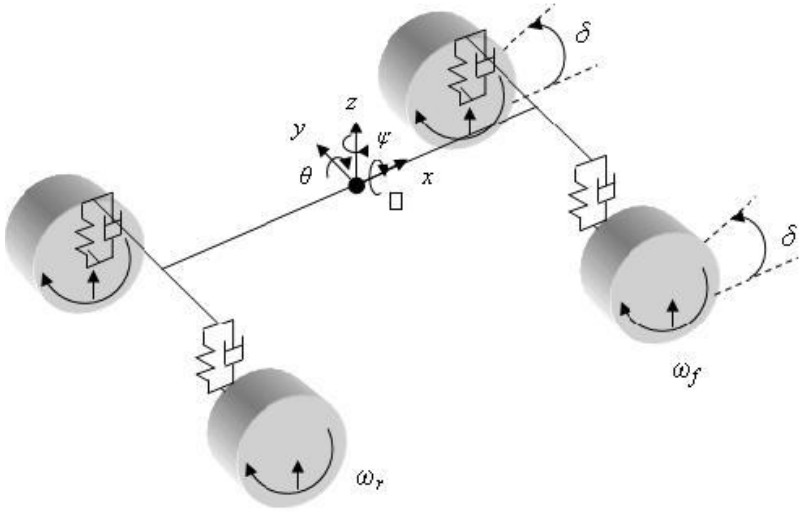

Figure 3.2: 14 DOF Vehicle Mode

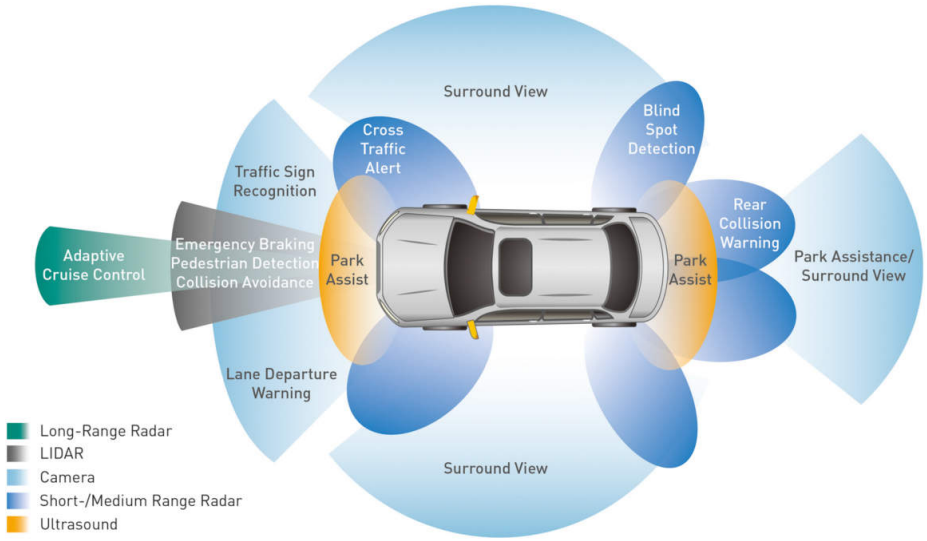

Figure 3.3: An overview of the sensing system for vehicles ADAS

3.1.4 ADAS Sensors

ADAS stands for the new Advanced Driver Assistance Systems, devices that can ensure safer cars, especially in terms of active protection. These new technologies (Figure 3.3) consist of a series of sensors that are mounted on cars in different locations and with different functions. Radar sensors, ultrasonic sensors, cameras, and in the future even lidar allow for the detection of the vehicle's environment and consequently for automated driving and parking capabilities. Cameras, in particular, permit to easily recover information such as the status of a traffic light or traffic signals can be integrated. The sphere of camera use is broad, for instance it

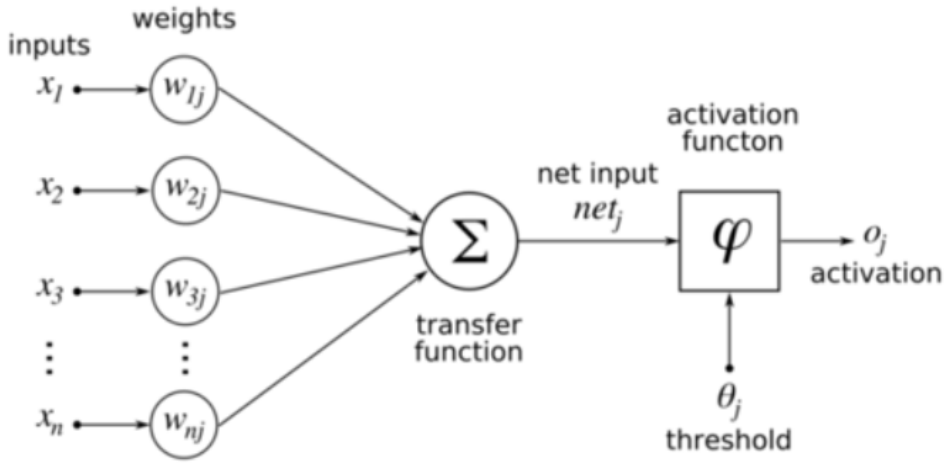

Figure 3.4: Schematical representation of the Rosenblatt perceptron.

recognizes lane markings and supports driver assistance features such as lane keeping or emergency braking, which react to the presence of vehicles, pedestrians, and bicycles.

3.2 Deep Learning

Deep Learning (DL) is the branch of Machine Learning (ML) that involves artificial neural networks (ANN) as models able to incorporate both the feature extraction and classification or regression steps.

3.2.1 Feed-Forward Neural Network

Multi-Layer Perceptron (MLP) represents an architecture of an artificial neural network, composed of fully connected layers. This means that every neuron of a layer receives as input the weighted output of all the neurons of the previous layer. Every neuron sums the weighted values of the input and possibly applies a non-linear function (Figure 3.4), intuitively representing whether or not the outcome is meaningful to be forward propagated.

3.2.2 Convolutional Neural Network

Convolutional Neural Network (CNN) is a more complex ANN architecture that exploits locally-connected layers. It involves a kernel that slides over the input tensor and provides a single-value result by means of a weighted sum (Figure 3.5).

3.2.3 Recurrent Neural Network

The **Recurrent Neural Network (RNN)** is an architecture whose connections form a direct graph along a temporal sequence (Figure 3.6). The RNN creates an

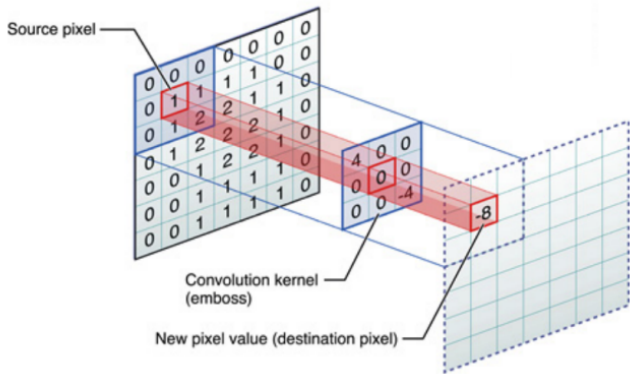

Figure 3.5: Schematic representation of a convolution operation.

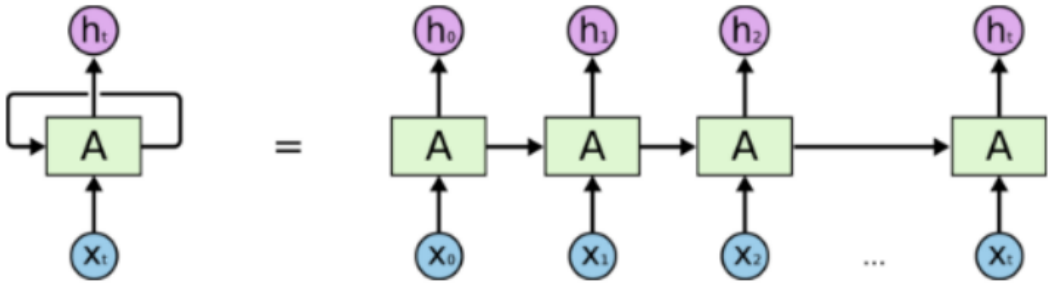

Figure 3.6: An unrolled recurrent neural network.

internal state h_t that encodes all the information it considers relevant to keep from the previous history.

This architecture provides a memory feature that is of paramount importance in time series and in all applications where the dynamic evolution of the data is highly informative.

3.3 Reinforcement Learning

Reinforcement Learning (RL) is a learning paradigm that envisions an agent capable of dealing with dynamic environments. This decision process is carried out by observing the environment state and selecting the preferable action to reach the final goal (Figure 3.7).

The two main entities involved in this learning process are:

- the **agent**: intelligent module capable of perceiving the environment and selecting the action to perform;
- the **environment**: context with which the agent interacts and that evolves according to its actions.

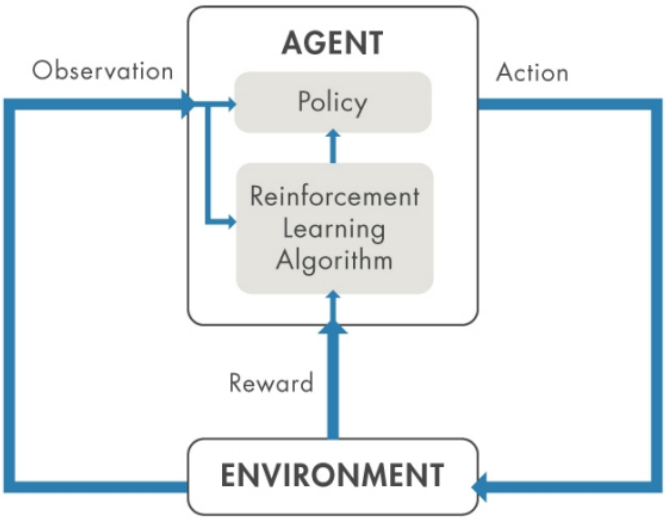

Figure 3.7: Reinforcement Learning loop.

3.3.1 Sim2Real gap

The environment is usually implemented as a virtual simulation for ease of manipulation, the possibility of handling a big amount of iterations, and speeding up the simulation time.

The drawback of this approach is the deployment of the trained agent in a real context that is different from the simulated one. This can correspond to a degradation of the model performance.

Therefore, the goal in designing the environment is reducing the so-called **simulation to reality gap**, hence creating an environment as realistic as possible.

3.3.2 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a discipline applying Deep Learning tools to Reinforcement Learning algorithms. In fact, it employs ANNs used as function approximators to learn value functions and/or policies, depending on the specific algorithm.

3.3.3 Model-Free vs Model-Based RL

One of the most important branching points in an RL algorithm is the question of whether the agent has access to (or learns) a model of the environment. Here, the model of the environment is a function that predicts state transitions and rewards.

The main upside to having a model is that it allows the agent to plan by thinking ahead or to train in the imagined environment. When this works, it can result in a substantial improvement in sample efficiency over methods that do not have a model.

The main downside is that a ground-truth model of the environment is usually

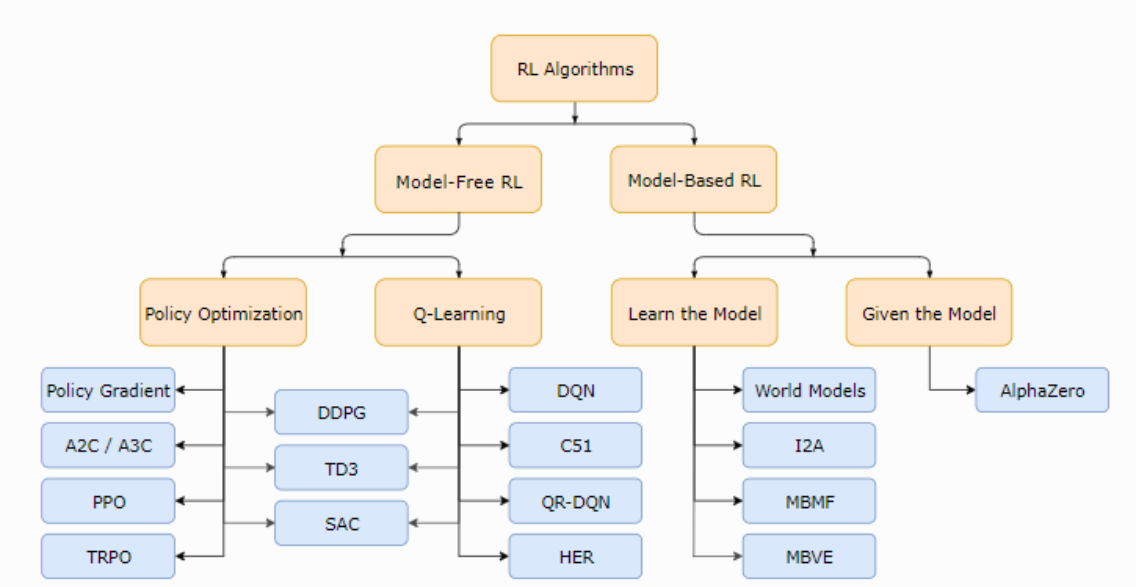

Figure 3.8: A non-exhaustive taxonomy of algorithms in modern RL

not available to the agent. If an agent wants to use a model in this case, it has to learn the model purely from experience, which creates several challenges. The biggest challenge is that bias in the model can be exploited by the agent, resulting in an agent which underperforms in the real environment.

Algorithms using an environment model are called **model-based**, and those that do not are called **model-free**, and depending on this there are different possibilities of what to learn [55].

Model-Free methods

The algorithm in this learning paradigm, in general, assumes the following non-exclusive approaches:

- **Value Learning,** aiming at learning functions accounting for the value of taking a specific action in a specific state (e.g. Q-function), then use them to infer a deterministic signal of what action to take;
- **Policy Learning,** inferring directly the policy that the agent should adopt. The model takes as input the state and predicts a probability distribution over its actions.

The actual methods can go beyond one or the other by adopting trade-offs between them or interpolating from one to the other.

Model-Based methods

In this case, there are even more orthogonal clusters of approaches, where the model may either be given or learned. Therefore, we propose a list far from being exhaustive:

- **Pure Planning,** never explicitly representing the policy, but employing pure planning techniques like model-predictive control (MPC) to select actions. In MPC, each time the agent observes the environment, it computes a plan which is optimal with respect to the model, where the plan describes all actions to take over some fixed window of time after the present. The agent then executes the first planned action, discards the rest of the plan, and computes a new one;
- **Expert Iteration,** a straightforward follow-on to pure planning that learns an explicit representation of the policy. The agent uses a planning algorithm in the model, generating candidate actions for the plan by sampling from its current policy. The planning algorithm produces an action that is better than what the policy alone would have produced, hence it is an "expert" relative to the policy. The policy is afterward updated to produce an action more like the planning algorithm's output;
- **Data Augmentation for Model-Free Methods,** using a model-free RL algorithm to train a policy or value function, but augmenting real experiences with imagined ones for updating the agent;
- **Embedding Planning Loops into Policies,** envisioning a subroutine, so that complete plans become side information for the policy, while training the output of the policy with any standard model-free algorithm.

3.3.4 RL Algorithms employed

In this section, we deepen the Reinforcement Learning algorithms characterizing this thesis, starting from **SAC**, the model-free baseline for training our agent, to **MBPO**, its model-based version[1], at the core of the work.

Soft-Actor-Critic

Soft Actor Critic (SAC) [56] is an algorithm that optimizes a stochastic policy in an off-policy way. A central feature of SAC is **entropy regularization**. The policy is trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy. This has a close connection to the exploration-exploitation trade-off: increasing entropy results in more exploration, which can accelerate learning later on. It can also prevent the policy from prematurely converging to a bad local optimum.
More in-depth, let x be a random variable with probability mass or density function P. The entropy H of x is computed from its distribution P according to

$$H(P) = \underset{x\sim P}{E}\big[-\log P(x)\big] \tag{3.1}$$

In entropy-regularized reinforcement learning, the agent gets a bonus reward at each time step proportional to the entropy of the policy at that timestep. The

[1]Actually, MBPO describes a methodology at whose core there can be a generic model-free algorithm. We considered MBPO as the authors of the original paper did.

resulting RL problem is

$$\pi^* = \arg\max_{\pi} E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \Big(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \Big)\right], \tag{3.2}$$

where

- s_t is the state of the environment at time t;
- a_t is the action performed by the agent at time t;
- $R(\cdot)$ is the reward function;
- $\pi(\cdot)$ is a policy;
- $\alpha > 0$ is the explore-exploit trade-off coefficient, with higher values corresponding to more exploration, and lower ones corresponding to more exploitation.

At test time, to see how well the policy exploits what it has learned, stochasticity is removed and it is used the mean action instead of a sample from the distribution. Having overviewed the policy, it is possible to the value functions in the specific form they assume in this algorithm.

- the V-function, representing the value of being in a determinate state, here considering also the entropy term:

$$V^{\pi}(s) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \Big(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \Big) \,\middle|\, s_0 = s\right]; \tag{3.3}$$

- the Q-function, representing the value of taking a specific action in a determinate state, and then behaving according to the policy:

$$Q^{\pi}(s, a) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t H\left(\pi(\cdot|s_t)\right) \,\middle|\, s_0 = s, a_0 = a\right], \tag{3.4}$$

  from which, after some manipulation, we obtain the resulting Bellman equation:

$$Q^{\pi}(s, a) = \underset{s' \sim P}{E}\Big[R(s_t, a_t, s_{t+1}) + \gamma V^{\pi}(s')\Big] \tag{3.5}$$

SAC concurrently learns a policy π_θ, two Q-functions Q_{ϕ_1}, Q_{ϕ_2}, and in some implementations a value function V_ψ, where the subscript represents a parametrization. Such a parametrization involves ANNs and the learning process is characterized by the following items:

- the loss functions for the Q-networks in SAC are

$$L(\phi_i, \mathcal{D}) = \underset{(s,a,r,s',d) \sim \mathcal{D}}{\mathrm{E}}\left[\Big(Q_{\phi_i}(s, a) - y(r, s', d)\Big)^2\right], \tag{3.6}$$

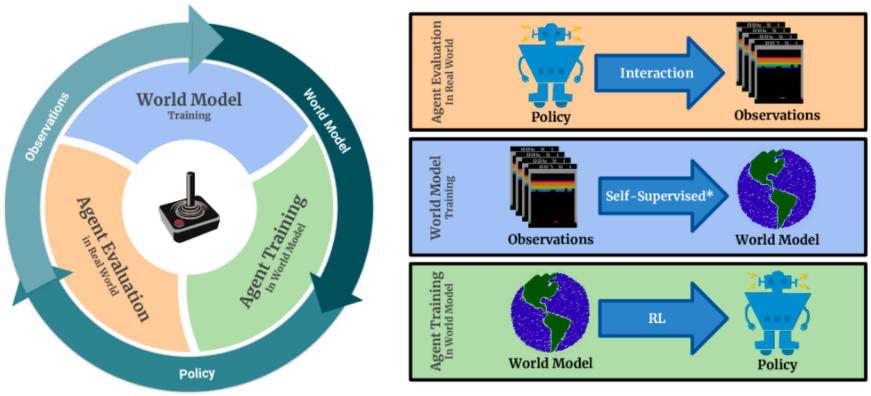

Figure 3.9: MBPO high-level representation.

where the target is given by

$$y(r, s', d) = r + \gamma(1-d)\left(\min_{j=1,2} Q_{\phi_{\text{targ},j}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \qquad \tilde{a}' \sim \pi_\theta(\cdot|s'). \tag{3.7}$$

- the policy is optimized by gradient ascent on the following objective function

$$\max_\theta \underset{s\sim\mathcal{D},\xi\sim\mathcal{N}}{E} \min_{j=1,2} Q_{\phi_j}(s, \tilde{a}_\theta(s,\xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s,\xi)|s), \tag{3.8}$$

where it is possible to notice the use of the reparameterization trick, in which a sample from $\pi_\theta(\cdot|s)$ is drawn by computing a deterministic function of state, policy parameters, and independent noise. Practically, it is used a squashed Gaussian policy to obtain the actions as follows:

$$\tilde{a}_\theta(s,\xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s) \odot \xi\right), \qquad \xi \sim \mathcal{N}(0, I). \tag{3.9}$$

Model-Based Policy Optimization

Model-Based Policy Optimization (MBPO) is a model-based RL algorithm using the environment model with the purpose of "Data Augmentation for Model-Free Methods", introduced in section 3.3.3. The high-level idea behind this algorithm is represented in Figure 3.9 and consists of a three phases loop:

1. the agent interacts with the real environment;
2. a model of the environment through Supervised Learning is learned, creating the "fictional" copy of the environment;
3. the agent is trained on the learned environment model.

The description of such an algorithm is deepened while exemplifying how it is employed in the Simulink environment introduced in Section 2.4.3.

In the outer loop, schematized in Figure 3.10 the agent interacts with the simulator, the real environment, choosing the best action, and leaving the stochasticity to

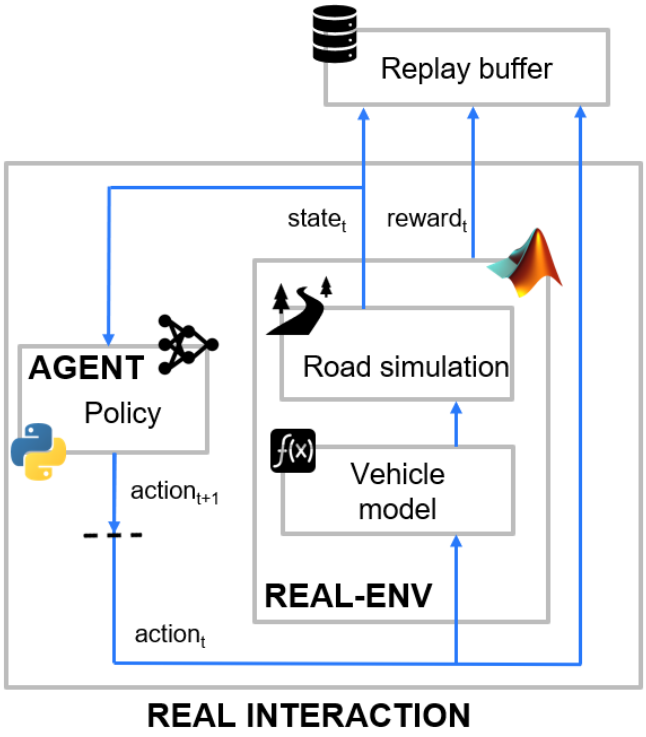

Figure 3.10: Interaction with the real environment.

Figure 3.11: Environment-model learning.

the phase of the interaction with the environment model. The transitions of State-Action-NextState are stored in a buffer, which will be employed for environment learning.

Then there is the phase of the environment modeling, depicted in Figure 3.11 left to probabilistic neural networks, networks whose output neurons simply parameterize a probability distribution function. The objective function responsible for the training of such ANNs is the negative log-prediction probability:

$$loss_P(\theta) = \sum_{n=1}^{N} -log f_\theta(s_{n+1}|s_n, a_n), \quad (3.10)$$

where $f_\theta(s_{n+1}|s_n, a_n)$ is the approximator of the conditional distribution of the next state given the current state and action, parametrized by the ANN, and N is the

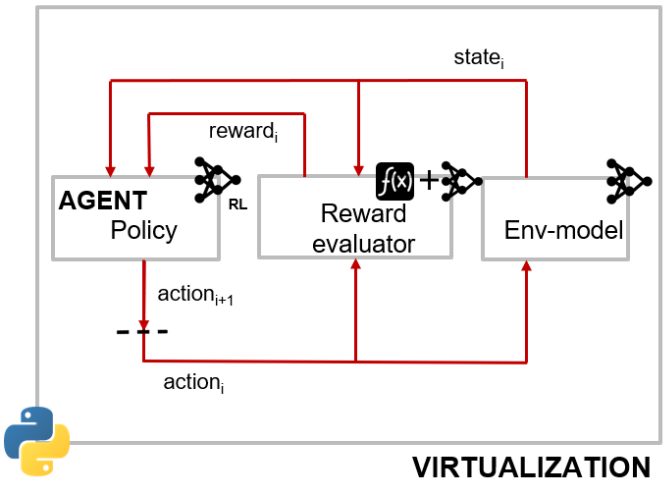

Figure 3.12: Interaction with the imagined environment.

number of samples over which the sum is computed. In this particular case the environment model outputs a Gaussian distribution with diagonal covariances:

$$f_\theta(s_{n+1}|s_n, a_n) = \mathcal{N}\big(\mu_\theta(s_t, a_t), \sigma_\theta(s_t, a_t)\big), \tag{3.11}$$

and the learning is performed through gradient descent. A probabilistic ANN captures aleatoric uncertainty or the noise in the outputs with respect to the inputs. However, the usage of an ensemble of probabilistic ANNs allows for a bootstrapping procedure accounting for epistemic uncertainty, or uncertainty in the model parameters. This technique is introduced in [57] and is crucial in regions when data is scarce and the model can be exploited by policy optimization.

Finally, there is the inner loop (Figure 3.12), in which the agent interacts with the imagined model, in a loop that can be defined as "virtualization". The policy optimization algorithm adopted is SAC, and once the agent takes an action, the environment-model ensemble generates a new state by simply selecting a model uniformly at random[2]. In Figure 3.12, the "reward evaluator" is an entity out of the environment model, to highlight how the reward can be learned as well, or how it is possible to imagine the insertion of a separately trained block ANN [3]. A final key feature of MBPO consists of its model usage: instead of having the agent perform a few long rollouts from the initial state distribution[4], many short rollouts are performed starting from some replay buffer states.

The complete loop previously reviewed is shown in Figure 3.13.

[2]This has the further advantage of allowing for different transitions along a single model rollout to be sampled from different dynamics models.

[3]This second possibility, the insertion of a separately trained block is not in the original paper, and could be the focus of future works.

[4]The state in which it arrived in the real environment.

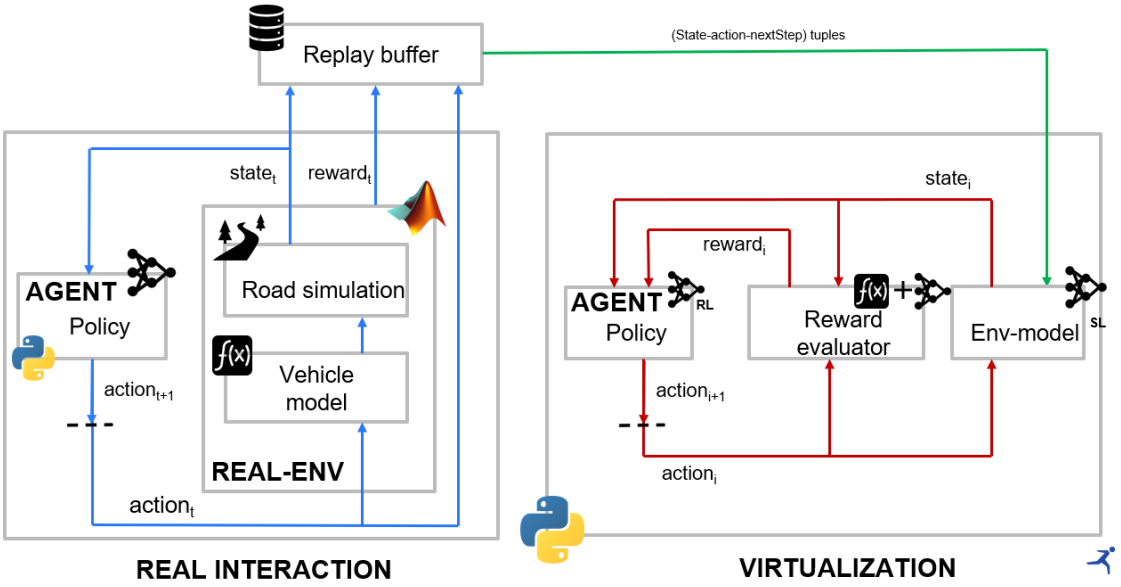

Figure 3.13: MBPO complete loop.

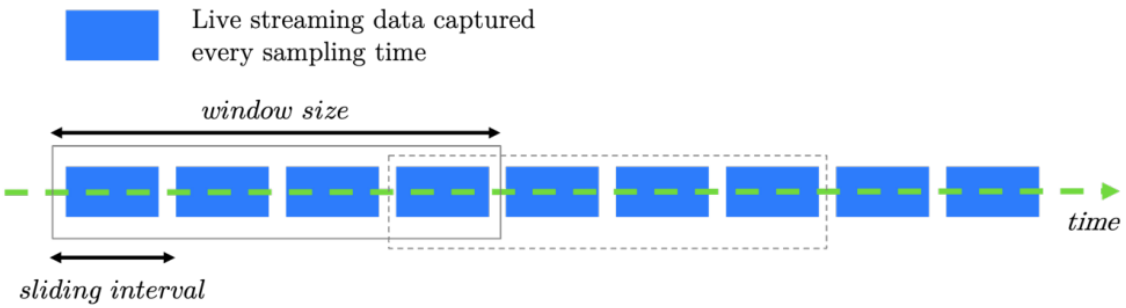

Figure 3.14: Sliding windows on streaming data

3.4 Data structures

A data structure defines the format for processing, organizing, and storing data. There are different types of structures, all designed to organize data for a specific purpose. Here it is introduced the only advanced data structure employed by the developed predictor: the sliding window.

3.4.1 Sliding window

In a sliding window (Figure 3.14) data are grouped within a window that slides over the data stream according to a specified interval. The employed time-based sliding windows have a fixed prediction horizon, namely the number of time-steps within the window, and a fixed sampling time.

Chapter 4

Material and methods

The current chapter presents the formalizations, methodology, environments, and tools characterizing the work presented. First, it is given an overview of the developed prediction system. Follows a step-by-step deepening of the workflow needed to obtain it with its key techniques. After this, there is a description of the dataset onto which perform the evaluation of the proposed system, as well as of the state-of-the-art model exploited to make a comparison. Finally, a first enhancement of the developed system is proposed.

4.1 System overview

This work focus on a prediction system consisting of an RL agent running in parallel to a human driver in a learned environment model. The high-level idea is represented in Figure 4.1.

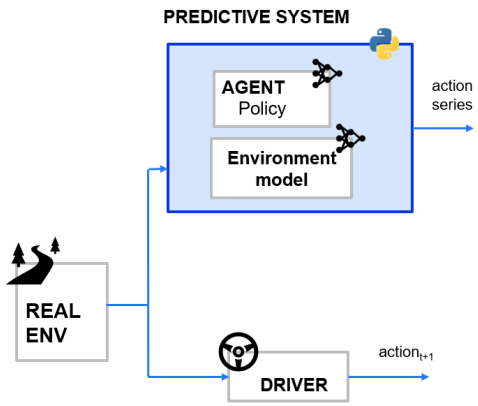

Figure 4.1: High-level representation of the proposed prediction system

In particular, the development workflow of the prediction system at the core of this thesis requires first the training of a Model-Based Reinforcement Learning agent, through the MBPO algorithm as described in Section 3.3.4. After that, two key modules are available:

- the **agent**, trained through Reinforcement Learning;

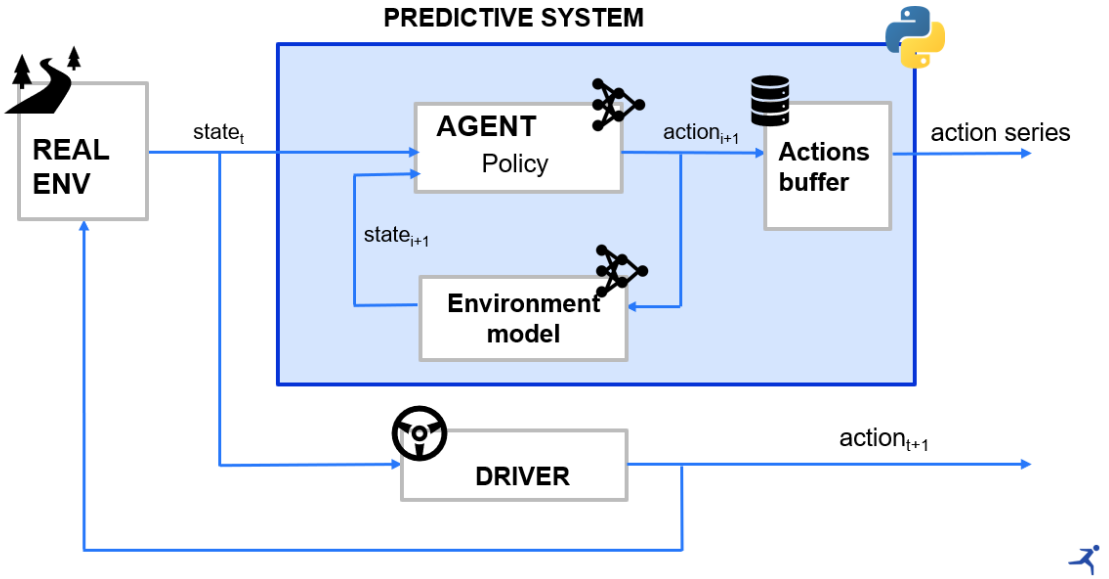

Figure 4.2: The final prediction system, composed of modules trained through Model-Based Reinforcement Learning.

- the **environment model**, trained through Supervised Learning.

The prediction system is built upon these modules as deepened in Figure 4.2 and the way it works can be summarized by the following pipeline.

1. The system inputs **Driving real-time data:**
   - **Vehicle variables**, CAN bus signals from the driveline and vehicle dynamics, such as IMU acceleration, lateral and longitudinal speed, etc.;
   - **Environmental conditions**, information from the road environment, e.g. the distance of the vehicle from the lane lines.
2. The input undergoes a series of transformations to be compliant with the trained Reinforcement Learning agent inputs, so as to provide an **initial state**;
3. The predictive system **loops in the learned environment model** for a fixed number of steps, depending on the desired prediction horizon. Specifically, the RL agent
   - chooses the best action to take in the current state and saves it in a buffer;
   - moves one step ahead in time in the learned environment model, reaching a new state;
4. The series of actions taken represents the **action prediction** and can be evaluated against the ground-truth actions for further applications.

In this work, one of the greatest challenges is to deal with the indeterminacy of subjective human behavior: the difficulty of creating a model that generalizes

sufficiently to the specifics of each person's driving style. The operational workflow described above will be supposed to be performed in the vehicle, to minimize response latency and operate in real-time. On the other hand, this has the disadvantage of harsher computational constraints on the complexity of the Machine Learning model, due to hardware limitations.

4.2 Model-Based Reinforcement Learning framework

In the following paragraphs, it is presented a review of the specific shape taken by the environment, the learning workflow key details, and the principal parameters and structures involved in the MBRL framework. In fact, a major effort in this work consisted in ensuring the compatibility between the MATLAB-Python environment introduced in Section 2.4.3, and the still under-development library mbrl-lib [58], resulting in several modifications of its source code[1].

4.2.1 Real environment

Before deepening the discussion, it is to highlight how the term "real environment" refers to the simulation environment with whom the agent interfaces, in opposition to the learned environment model, recalling a virtualization.
For the agent to be successful in the lane-keeping task first it was necessary to reshape the way it interacts with the environment, namely its observation space, action space, and the rewards it receives. Although previous works already solved the environment [2], this was done by learning algorithms other than the ones we employ, in particular by using the PPO [59] implementation of the stable-baselines library [54]. This algorithm does not allow for a Model-Based extension, so the presented work needs to start back from the development of an agent able to solve the environment, and this implies engineering the way it perceives it. No modifications are performed on the MATLAB simulator, so there is no difference in the states characterizing the real environment.

Observation space

The observation space is a fixed-size vector of continuous floating-point values representing the physical information collected in the driving environment and provided as input to the neural network. There are two sources of information:

- **vehicle dynamics**, consisting of
  - **x and y absolute traveled distance**, measured in the absolute reference system;

[1] In fact, it was necessary also to ensure compatibility with stable-baselines [54] library. Although no algorithms from this model-free reinforcement learning library were exploited, some modules were involved in the python environment itself.

Field	Units
x absolute traveled distance	[m]
y absolute traveled distance	[m]
longitudinal velocity	[m/s]
lateral velocity	[m/s]
longitudinal acceleration	[m/s^2]
lateral acceleration	[m/s^2]
yaw angle	[rad]
yaw rate	[rad/s]
lateral offset	[m]
heading angle	[rad]
lane boundary curvature	[rad/m]
lane boundary curvature derivative	[rad/m^2]

Table 4.1: Observation space

  - **longitudinal and lateral velocity**, measured with reference to the vehicle;
  - **longitudinal and lateral acceleration**, measured with reference to the vehicle;
  - **yaw motion**.

- **camera information**, focusing on the lane on which the vehicle is traveling:
  - **lateral offset** of the vehicle's position relative to the lane boundary, specified as an actual offset. An offset from the lane boundary to the left of the ego vehicle is positive. An offset to the right of the ego vehicle is negative;
  - **heading angle**, the initial lane boundary direction angle. The direction angle of the lane boundary is relative to the direction of the vehicle;
  - **lane boundary curvature**;
  - **lane boundary curvature derivative**.

This results in a 15-dimensional observation space, better shown in Table 4.1.

Action space

The action space is a fixed-dimensional vector of continuous floating-point values representing the decision made by the agent, namely output by the neural network that represents it, and delivered to the vehicle to act it on the environment. Although the three driving commands that are performed in the MATLAB environment are given by:

- **Steering wheel angle**;
- **Throttle pedal pressure** percentage;

Field	Operational Range	Units
Steering Wheel Angle	[-$\pi/2, +\pi/2$]	[rad]
Throttle-Brake pedals pressure	[-1, 1]	[adims]

Table 4.2: Action space

- **Brake pedal pressure** percentage;

the agent showed better behavior with a 2-dimensional action space, obtained by merging the signals concerning the pedals pressures. This leads to the **Throttle-Brake pedals pressure** signal, from which the single pedals pressures are obtained as follows:

- a positive value is interpreted as throttle pedal pressure, and the brake pedal pressure is set to zero in the MATLAB environment;
- a negative value is interpreted as brake pedal pressure, and the throttle pedal pressure is set to zero in the MATLAB environment.

The action space, along with the values that its fields can assume, is detailed in Table 4.2. To account for how often the agent can perform a new action there is the **decision interval** parameter. In the decision interval the agent performs the same action vector, and no observations are collected. In this work, the decision interval is fixed as the **observation sampling rate of 25 Hz**, implying that the agent performs a decision at every sampling time.

Reward function

Defining rewards is one of the key elements when approaching a given problem through Reinforcement Learning, having a major impact on the final behavior of the agent, sometimes resulting in completely unexpected behavior [1]. The aspects taken into account in the final reward function exploited during training are the following.

- **Time.** To avoid a stalemate situation and incentivize the agent to move, the time spent at each step is penalized according to

$$R_{time} = -k_{time} \tag{4.1}$$

  where k_{time} is the time reward weight coefficient;

- **Travel.** The "Direction Guided" reward function, defined in [60], allows the agent to learn to drive faster. The heading angle and the traveled distances (Section 4.2.1) are used at the center of this reward function component. If the vehicle goes in the direction of the road, a high reward is received. Conversely, if the vehicle goes in the opposite direction, it will receive a low reward or penalty, as exemplified in Figure 4.3. Mathematically, the travel reward is defined as

$$R_{travel} = direction\_guided \cdot k_{travel} \tag{4.2}$$

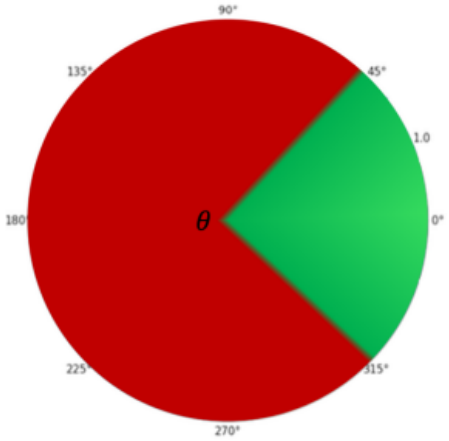

Figure 4.3: Visualization of travel reward acting as an incentive or penalty.

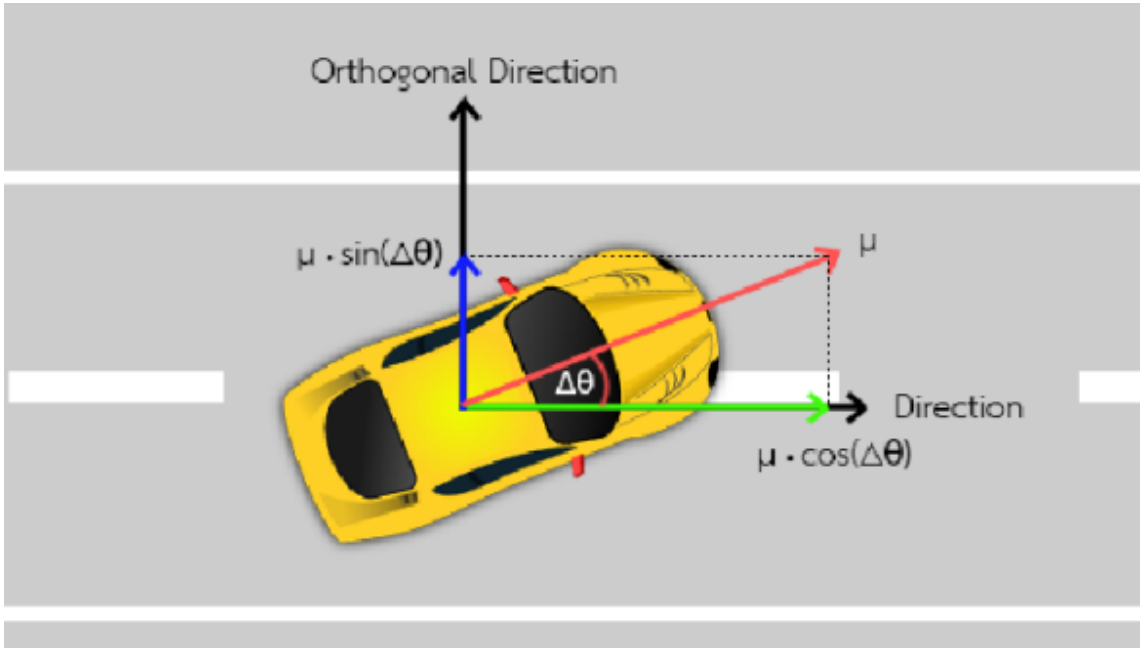

Figure 4.4: Components of the direction guided reward.

$$direction\_guided = \frac{\Delta r \cos \Delta\theta - |\Delta r \sin \Delta\theta|}{decision\_interval} \tag{4.3}$$

where k_{travel} is the travel reward weight coefficient, $\Delta\theta$ is the heading angle and Δr the distance traveled in a *decision_interval*, as can be visualized in Figure 4.4.

- **Lane-keeping task.** The agent must be able to not go off the road and we shall give a negative reward in this case:

$$R_{offroad} = L_{flag} \cdot k_{offroad} \tag{4.4}$$

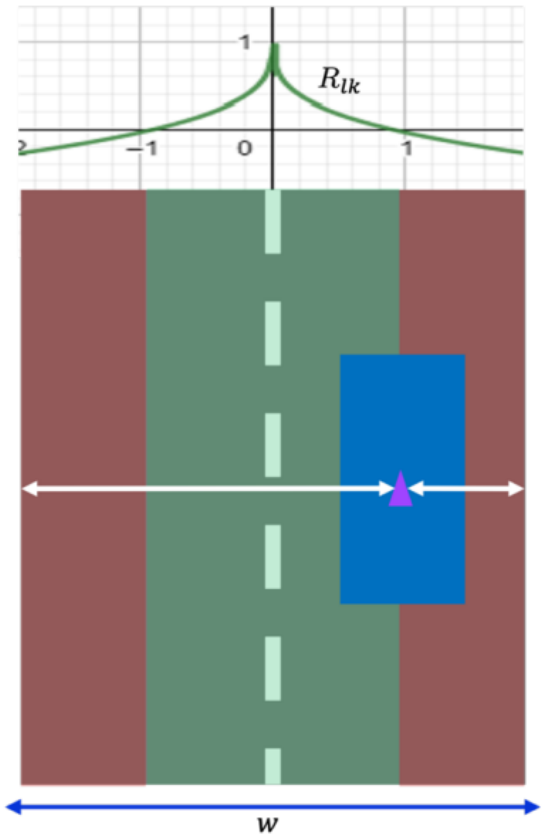

Figure 4.5: Correlation between vehicle CM position relative to lane centre and reward function.

$$L_{flag} = \begin{cases} 1 & \text{if offroad} \\ 0 & \text{else} \end{cases}$$

where $k_{offroad}$ is the reward weight coefficient and L_{flag} is the detection mark vehicle off the road. In the lane-keeping task, the position of the vehicle relative to the center of the lane must also be considered to assess compliance with the task. To assess the position of the vehicle within the lane, the distance of the center of mass (CM) of the vehicle from the center of the lane is used as an index.

As shown in Figure 4.5, the farther the vehicle gets from the lane center the more the reward decreases until it is no more than a quarter of the lane width away from the center, following an inverse cubic function:

$$R_{lk} = \left(\sqrt[3]{\frac{w}{4}} - \left| \sqrt[3]{l_{left} + l_{right}} \right| \right) \cdot k_{lk} \tag{4.5}$$

where k_{lk} is the reward weight coefficient of lane keeping task, w is the lane

Reward weight	Value
k_{time}	1
k_{travel}	5
$k_{offroad}$	200
k_{lk}	5

Table 4.3: Reward weights.

width and l_{left} and l_{right} are the lateral offsets[2].

The total reward function is given by the sum of the introduced components:

$$R_{tot} = R_{time} + R_{travel} + R_{offroad} + R_{lk}, \tag{4.6}$$

with the specific weights being reported in Table 4.3.

4.2.2 Experimental set-up

The workflow for the creation of the presented prediction system allows for experiments already at this stage of development, which is focusing on an autonomous driving agent, rather than on a predictor. However, the objective of the evaluation shall not be the quality of the driving agent, already assessed by previous works [2]. In fact, the focus is on one of the key elements in the presented methodology, the **model of the environment**, in particular, showing the trade-offs in adopting a **model-based or a model-free** approach in an industrial driving simulator. The algorithms compared in this experimental setup are treated in Section 3.3.4 and are:

- **SAC**, a model-free reinforcement learning algorithm, in which the agent is trained on the real environment, namely directly on samples collected during the interaction with the MATLAB environment;

- **MBPO**, the model-based version of SAC, in which a copy of the real environment will be learned and the agent will be trained on this, namely on samples collected on imaginary rollouts.

From a practical standpoint, this is a comparison involving the same learning algorithm, making use of the same hyperparameters when possible, to isolate and assess the impact of the practice of environment-model learning and exploitation.

The experiment is characterized by 3 trainings with different initial random seeds for both the algorithms presented, allowing the computation of **average** and **standard deviation** of the evaluation metrics introduced in Section 4.2.3.

[2]Recall from Section 4.2.1 that the an offset to the right of the ego vehicle is negative. Consequently, when the vehicle is at the center of the lane the offsets have sum equal to zero and the reward is maximum.

Parameter	Value
Learned rewards	False
Rollout length	1
Difficulty level	3
Number of runs	3
Maximum episode duration	2000 steps
Agent evaluation frequency	5000 steps
Environment-model training frequency	200 steps
Validation ratio	10 %
Patience	5 epochs
Log-likelihood improvement threshold	0.01
SAC gradient descent steps per environment step	20

Table 4.4: Parameters characterizing the training and evaluation phases of the algorithms.

4.2.3 Metrics

Following the Model-Based Reinforcement Learning literature [57, 61], the evaluation of the environment-model introduction is based upon metrics concerning the efficiency of the Reinforcement Learning framework. Although there does not exist an official mathematical formulation, the most common are:

- **Sample efficiency**: the reward collected in evaluation episodes after a fixed number of steps in the real environment;
- **Compute efficiency**: the average time for computing a fixed number of steps, measured on the same working machine. To directly compare the algorithms against each other it is introduced the **MBPO-SAC compute time ratio**, given by the ratio between the average computing time of MBPO and the average computing time of SAC;

4.2.4 Learning

In the following paragraphs, it is reported an overview of some design choices regarding the training and testing of the agent, as well as some key hyperparameters characterizing them, summarized in Table 4.4. A theoretical overview of the algorithms employed is given in Section 3.3.4, and to ease the reading of this section a visualization of the MBPO loop is reproposed in Figure 4.6.

Episode

During the training phase, numerous episodes are necessary, and it is necessary to restore the initial state whenever the episode ends (Section 3.3). The episode ends if at least one of three conditions occurs:

- the agent has completed the track;

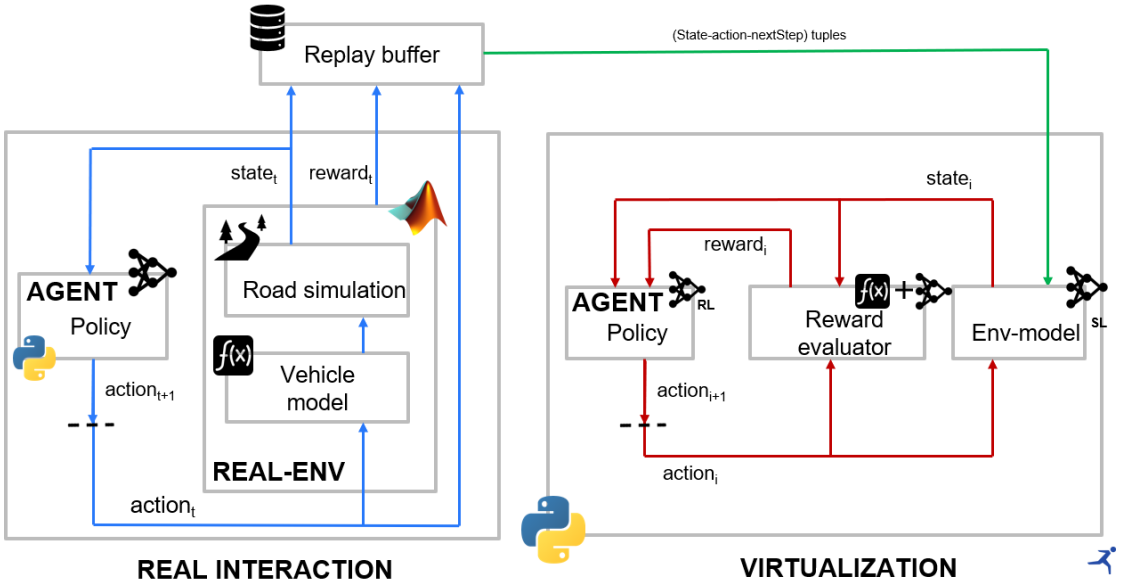

Figure 4.6: MBPO complete loop.

- the agent has gone offroad;
- the agent has reached the maximum number of steps, i.e., the maximum length of the episode even if still on the road. During all the trainings this parameter was fixed to **2000 steps**.

Afterward, the environment is reset to its initial state. The reset repositions the driver to the beginning of the track and regenerates the vehicle model.

Environment model

Although the environment model is learned in parallel to the interaction with the real environment, there are several elements characterizing its design.

- **Object of learning**. In this case, the inference is limited to the state transitions, namely, the object of learning is a function specifying what will be the next state starting from the current one and the action taken, as deepened in Section 3.3.4. This implies that no model of the reward will be learned, and thus, that it is necessary to pass a reward function to the environment model to evaluate the actions in the "virtualized" world. As it is reasonable, it is used the same reward function mentioned in Section 4.2.1, but this is not a requirement.
- **Termination function**. It is necessary to pass a function allowing the environment to understand that it is in a terminal state. In this case, it is not used the same exact one defined in Section 4.2.4. Indeed, for the specific MATLAB simulator exploited, the track final point has no difference from the other points in the road, it is needed only for resetting the environment. This is a practical need that has nothing to do with learning how to deal with an environment. Furthermore, it would make no sense to put a "maximum number of

steps" in the learned environment model. Consequently, in the environment model the termination function labels a state as **terminal only if the agent has gone offroad.**

- **Rollout length.** It is necessary to specify for how long to interrogate the environment model. To minimize the hyperparameters tuning phase duration the **rollout length is set to 1**, given that the MBPO paper states that, although this value can be sub-optimal, the baseline it sets is difficult to beat. Consequently, after each step in the real environment:

  1. a set of states is randomly sampled from a **replay buffer** containing the real states encountered by the agent during the interaction with the real environment;
  2. the agent acts on these starting states and goes to the next ones. These imagined single-transitions are stored in a second buffer, referred to as **SAC buffer**[3], on which the agent will be trained.[4]

Training

At the beginning of each training, and after each validation phase, a track is randomly generated from the MATLAB simulator according to a difficulty level (Section 2.4.3), which determines some key characteristics. Having conducted some tests on a straight road environment (with difficulty 0 or 1), both SAC and MBPO agents were able to beat it, so the final **difficulty level is set to 3** to have a direct comparison between the two. Indeed, this level already presents fairly complex curves, resembling extra-urban roads, and it was not needed to perform an evaluation on extremely harsh environments for the purpose of this thesis project.

After **5000** steps the agent's training is stopped and an **evaluation episode**[5] is conducted, in which a GUI (Graphic User Interface) allows to see how the agent is performing, as shown in Figure 4.7. Furthermore every **200** steps in the real environment **the environment-model is trained**:

1. it is created a training-set with the 90% of the samples in the replay buffer and a validation-set with the remaining 10%;
2. the environment model is trained to minimize the negative prediction log-likelihood, as mentioned in Section 3.3.4, exploiting the **early stopping** technique: if after a number of epochs referred to as **patience** no improvement higher than an **improvement threshold** is observed on the validation-set, training stops.

[3]Recall that the learning agent is SAC both in the model-free and model-based (MBPO) frameworks.

[4]A difference between the replay buffer and the SAC buffer is that the first is continuously filled, while the second gets discarded and re-filled at each interaction. The reason behind this is that the samples in the SAC buffer are sampled from the learned environment model, which gets updated during training.

[5]Subject to the termination conditions outlined in Section 4.2.4.

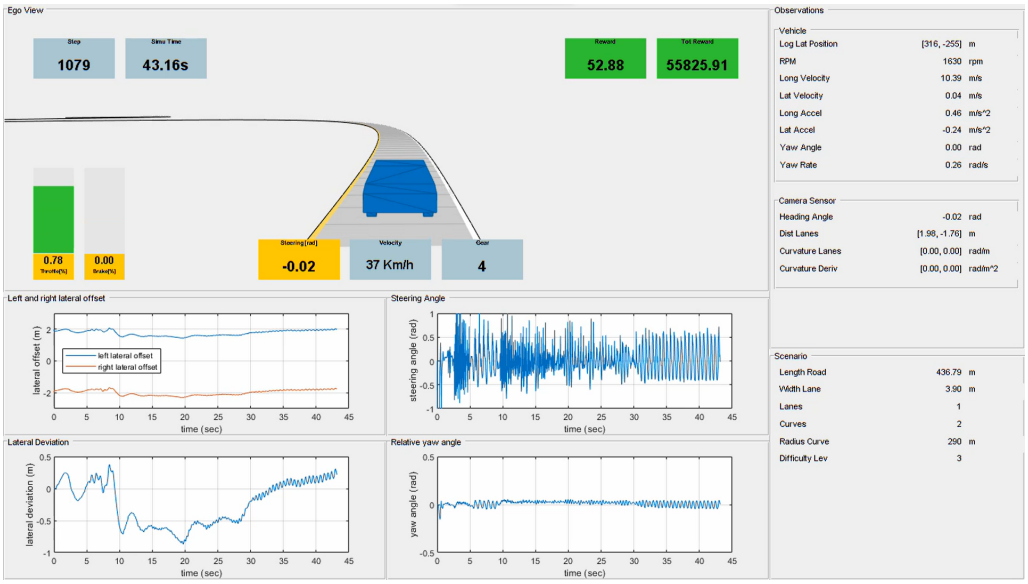

Figure 4.7: Graphical interface of agent behaviour.

The SAC agents in both the model-free and model-based frameworks (MBPO) are trained after each step in the real environment, with the parameters of their ANNs being updated with **20 gradient descent steps**. This high ratio between ANNs parameters updating and real environment steps is quite common in model-based frameworks, and following the MBPO paper, it is kept also in the model-free framework.[6]

4.2.5 Architectures

Although the last years in Deep Learning, as the name suggests, have been characterized by particularly deep architectures, with reference to the number of layers, Reinforcement Learning backbones do not present this peculiarity yet. Furthermore, for the work developed, the state perceived by the agent is characterized by engineered tabular features, so there is no need for the employment of complex architectures for feature extraction[7]. Consequently, for both the agent and the environment model the ANNs structures consist of simple MLPs (Section 3.2.1).

Agent backbones

There are **3 MLPs**, for approximating the 3 functions characterizing the SAC agent (Section 3.3.4):

- **Policy function**, reported in Table 4.5. It inputs the current and outputs a

[6]Actually, such a test in the proposed framework seems to be fair not only for the literature referral. Indeed, in some fast experiments, the update frequency was lowered in the model-free framework, to not introduce any disadvantage, but no improvement was observed. Consequently, tests were performed as the literature suggests.

[7]For instance CNNs 3.2.2 would have been considered in presence of image inputs.

Layer	Number of neurons	Annotation
Input	15	State space dim
Fully-connected layer 1	256	ReLU activation function
Fully-connected layer 2	256	ReLU activation function
Output	4	Action space means and vars

Table 4.5: Policy function ANN structure.

Layer	Number of neurons	Annotation
Input	17	State space + Actions space dim
Fully-connected layer 1	256	ReLU activation function
Fully-connected layer 2	256	ReLU activation function
Output	1	Q value

Table 4.6: Q-functions ANNs structures.

Layer	Number of neurons	Annotation
Input	17	State space + Actions space dim
Fully-connected layer 1	200	Leaky-ReLU activation function
Fully-connected layer 2	200	Leaky-ReLU activation function
Output	30	State space means and vars

Table 4.7: Environment-model ANNs structures.

Gaussian mean and standard deviations, from which the current action will be sampled;

- Two different **Q-functions**, reported in Table 4.6. They input the current state and current action and output the Q-value, accounting for the goodness of taking such an action in that state.

All these feed-forward ANNs present 2 hidden layers with the ReLU non-linearity as activation function:

$$ReLU(x) = \begin{cases} x & if \quad x \geq 0 \\ 0 & if \quad x < 0 \end{cases} \tag{4.7}$$

Environment-model backbones

The environment model is represented by an **ensemble of 8 MLPs** whose structure is presented in Table 4.7. It takes a 17-dimensional input given by the concatenated 15-dimensional current state and 2-dimensional current action. The signal is propagated through 2 hidden layers having a Leaky ReLU as activation function, where

Parameter	Value
Prediction horizon	2 sec
Window length	50 samples
Prediction interval	0.04 sec

Table 4.8: Prediction window summary.

$$LeakyReLU(x) = \begin{cases} x & if \quad x \geq 0 \\ 0.01 \cdot x & if \quad x < 0 \end{cases} \tag{4.8}$$

Finally, the ANNs return the means and variances of a Gaussian representing the distribution of the 15-dimensional next state, resulting in a 30-dimensional output.

4.3 Prediction system

Having explored the details behind the design and training of the MBPO agent, it is possible to dive into deepening how its modules can provide the **action prediction** system introduced in Section 4.1, and how it can be used for **trajectory prediction** as well.

4.3.1 Implementation

Prediction window

A key design feature of the developed system is how long it will predict in the future, a time referred to as **prediction horizon**, set to **2 seconds** when performing inference[8], being this a common value in the literature.
Given that the MBPO agent has been trained to interact with the environment at 25 Hz (Section 2.4.3), when it will move ahead in time, it will interact with the environment model with the same frequency. Consequently, each action (and imagined observation) will present an **interval of 0.04 seconds** with the next one, and the 2-seconds prediction window will present **50 entries**. These features are summarized in Table 4.8.

MBRL Modules exploitation

It is now possible to deepen the interaction between the trained agent, which will be the digital twin of the driver, and the environment model, mentioned at point 3 of Section 4.1.

1. The agent perceives the state of the environment which is:

[8]Notice how the prediction system presented is used only in inference, there was not a training phase of the predictor, being it composed of modules obtained through the MBRL framework.

- **real** only in the first step, derived from the real situation that the driver is facing. In this case, a buffer is initialized;
- **imagined** in further steps.

2. The agent **acts deterministically** outputting the **mean value** returned by the policy network, that represents the best action[9]. For the **action prediction** task the action is saved into an **action buffer**;

3. The learned environment model is stepped: the ensemble of backbones of the environment model inputs the current state and the action taken by the agent and **outputs deterministically the next state**[10]. For the **trajectory prediction** task the first two entries of the next state, representing the traveled distances along the x and y axis, are saved into an **offsets buffer**. These are in absolute value, so they are given the sign of the third and fourth next-state entries, representing the longitudinal and lateral velocity of the vehicle.

   Having an ensemble of ANNs and needing to store just one value in the offsets buffer, among the output next states is chosen the one whose absolute sum of the entries is minimum. Regarding action prediction, it is stored the action associated with this state in the actions buffer. The rationale behind this choosing criterion is to prevent potential errors of the ANN representing the environment model: if an entry in the next state vector diverges, this minimization criterion will not choose that state as a prediction. Practically, this constraint acts as a regularization method;

4. If the prediction horizon has not been reached return to step 1 of this loop.

For the **trajectory prediction** task to obtain **the final trajectory**:

- it is performed a **cumulative sum** in the offsets buffer;
- each entry is added the **initial coordinates** of the vehicle, information available to the system.

4.3.2 Dataset

To test the proposed predictor, as well as to perform a comparison with a state-of-the-art model, a human demonstrations dataset is needed. The data collected were

- encoded in the form seen by the RL agent, transforming them appropriately, and discarding non-used features (e.g. RPMs);
- used to extract a dataset with the following shape:

[9] Recall that, to act stochastically, the policy network outputs the mean and variance of the action vector, for then sampling the actual action.

[10] Also these ANNs, to inject stochasticity, output the mean and variance of the state vector, for then sampling the next state. To perform a deterministic step they simply output the mean value.

Figure 4.8: Steering wheels and pedals employed during human data collection.

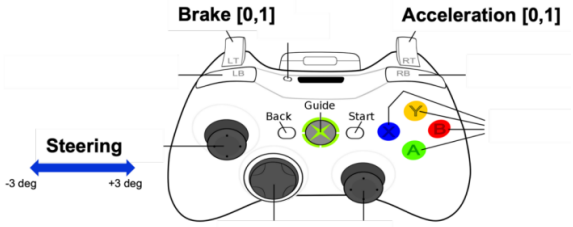

Figure 4.9: Controller employed during previous human data collection.

- **input features:** the input of the system, practically these will be the initial states of the real environment;
- **output time-series:** the ground truth windows, composed of actions or positions, depending on the task.

Collection

The collection was a supervised process to observe external driver behavior and to evaluate future improvements in this procedure. In particular, each person was explained the objective of the simulation, namely to complete the track without ever leaving their lane, and asked to maintain as consistent a driving style as possible throughout the collected demonstrations. Drivers were given the opportunity to familiarise themselves with the controls with a few test laps. Most importantly, the collection was performed in the same simulator exploited by the agent, to which the human interfaced through Logitech gaming steering wheels and pedals, shown in Figure 4.8. The obtained data were integrated with data from previous collections performed through an Xbox Joystick as the driving controller, using the two potentiometers (LT and RT, in Figure 4.9) to simulate the pressure on the throttle and brake pedals.

To even ease the human interface with the simulator the 3D rendering of the environment was enabled, and to ensure realism no data was displayed on the GUI, as shown in Figure 4.10.

Summarizing, there are three types of tracks:

- a simple straight road, with 30 seconds lasting tracks;
- a closed curved road, lasting 120 seconds. The map of this is shown in Figure

Figure 4.10: 3D rendering of the simulation environment.

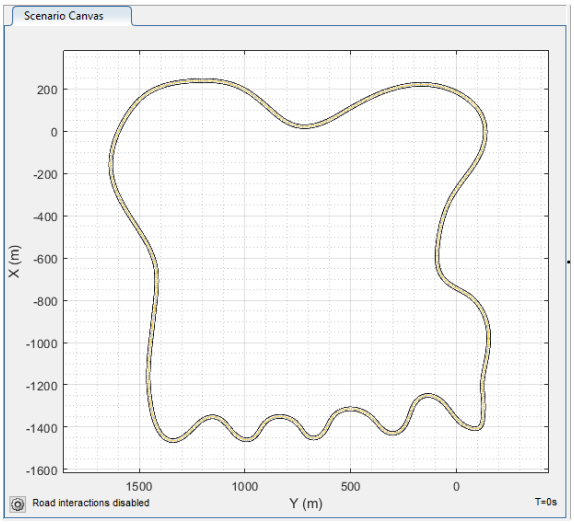

Figure 4.11: Map of the curved road onto which human data have been collected.

4.11, but no human completed the whole track. The collection time was not increased since the driver, after that threshold time, started to consistently lower its performance. Since analyzing variations in driving style, such as tiredness, distraction, etc. is out of the scope of this thesis project, the 120 seconds lasting laps were kept, allowing to define the driver state as **nominal**;

- several increasing difficulties driving tracks, randomly generated as the ones in which our agent was trained (Figure 2.28).

Metadata and required additional labels

The collection of further data, in the form of metadata and labels, is not thought to be relevant. Indeed, both the prediction tasks considered just need a transposition of the data collected: the construction of timeseries of positions and actions. In fact, the prediction system developed is nothing but a virtual twin of the driver, so it is not possible to think of additional data that would increase the quality of the predictor without these having been seen by the agent: the agent and the predictor are the very same thing.

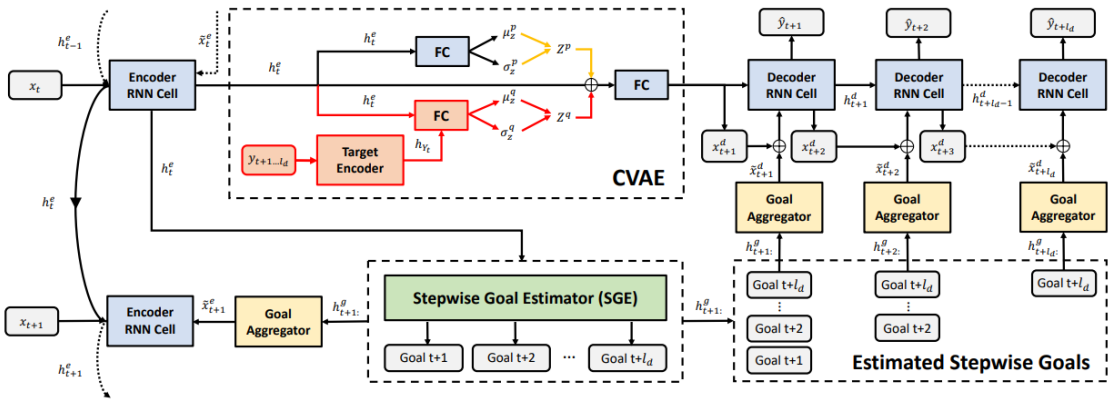

Figure 4.12: SGNet

4.3.3 Comparison model: SGNet

To compare against the state-of-the-art a **deterministic SGNet** is instantiated, a model overviewed in Section 2.2.2. This is shown in Figure 4.12, which presents also a Conditional Variational Autoencoder (CVAE), needed to provide stochastic outputs, that we will not use in our model. This model is chosen because in its paper [21] it is described as a methodology prone to working on different data sources[11] as long as the historical positions are available: additional data are just concatenated to these inputs.
To test in the most similar conditions in which the proposed predictor works it is necessary to:

- provide as **input** the **vehicle initial coordinates**, plus the set of features characterizing the **observation of a RL agent**[12], reported in Table 4.1. Consequently, the input will be a degenerated time-series composed of only one sample;

- require the **same outputs**, a prediction window of 50 samples, with 2 seconds as prediction horizon.

However, the described model, as will be deepened in Section 5.1, was not able to return an efficient system, so the predictor developed in this thesis project, not having any memory of the past, is compared with two further SGNets, receiving in input time-series of:

- **10 samples**, namely 0.4 sec, a fifth of the future prediction window;

- **25 samples**, namely 1 sec, half of the future prediction window.

[11] In fact, the authors trained and tested on different types of datasets for different benchmarks.

[12] Except for the absolute traveled distances, to avoid redundant information. In fact, this model also sees its position, in terms of coordinates, differently from the MBPO agent.

As is usual in the data-driven approaches not all of the data are employed for testing the models. In this case, Driver Trajectory Prediction task evaluation envisions a comparison with a state-of-the-art model, belonging to the Supervised Learning paradigm. Because of this, there is a need for data to train and validate this model before testing it. Consequently, available data are split as follows:

- **Training split,** composed of 2 tracks for each driver in the curved road and 6 tracks in the increasing difficulty scenario. This split counts 12 400 samples and it is used for training the SGNets;
- **Validation split,** composed of 1 track for each driver in the curved road and 2 tracks in the increasing difficulty scenario. This split counts 8 300 samples and it is used for validating the SGNets and stopping the training when no improvements are observed;
- **Testing split,** composed of 1 track for each driver in the curved road, 1 track in the straight road for each driver, and 2 tracks in the increasing difficulty scenario. This split counts 9 700 samples and it is used for testing both the proposed prediction system and the SGNets.

4.3.4 Experimental set-up

The object of the evaluation will be the quality of the prediction in:

- the **action prediction** task, at the core of the thesis project;
- the **trajectory prediction** task, central in the literature, from which the **SGNet** is taken, observing how a state-of-the-art model compares with our **predictor** in a realistic scenario, with scarce sources of data.

Furthermore, it is proposed a comparison centered on **memory requirements** of the proposed model and the SGNet. The metrics allowing for this are defined in Section 4.3.5.

Finally, although it should be obvious, it is to highlight that such a test on a preprocessed dataset constitutes an offline procedure. Bringing the developed system to work in real-time to test in online frameworks will be an idea for future works.

4.3.5 Metrics

Driver Action Prediction

The predictor is evaluated on the following metrics, designed throughout this project:

- **Window MSE** and **Window MAE**, where the difference is computed for each signal at each time-step and averaged over the prediction window:

$$MSE = \frac{1}{H}\sum_{t=1}^{H}\big(u(t) - \hat{u}(t)\big)^2, \tag{4.9}$$

$$MAE = \frac{1}{H}\sum_{t=1}^{H}\big|u(t) - \hat{u}(t)\big|, \tag{4.10}$$

where

- t is the time-step;
- H is number of steps in the prediction window;
- $\hat{u}$ is the predicted signal;
- u is signal ground truth;

- **WAE** (Window Absolute Error), where the absolute difference is computed for each signal at each time-step:

$$\mathbf{WAE} = \big|u(t) - \hat{u}(t)\big|, \tag{4.11}$$

returning not a scalar but a vector with each entry corresponding to a time-step. This allows for analyzing how the prediction quality degrades when the prediction horizon increases;

- **Window Sign Accuracy**, averaging over the prediction window the number of times the sign of the signal is correct. This happens when the system predicts the right turning direction, or it does not mistake an acceleration for a braking and vice-versa:

$$Sign\_accuracy = \frac{1}{H}\sum_{t=1}^{H} sign\_\delta\Big((u(t), \hat{u}(t)\Big), \tag{4.12}$$

$$sign\_\delta(x, y) = \begin{cases} 1 & \text{if} \quad sign(x) = sign(y) \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

where $sign(\cdot)$ is the sign function.

Trajectory Prediction

The comparison against the state-of-the-art model trained on our data is conducted on the most common evaluation metrics in the literature:

- **ADE** (Average Displacement Error), the Euclidean distance between the real and the predicted trajectories:

$$ADE = \frac{1}{H}\sum_{t=1}^{H} dist\big(\mathbf{x}(t), \hat{\mathbf{x}}(t)\big), \tag{4.14}$$

where

- $dist(\cdot)$ is the Euclidean distance;
- $\hat{\mathbf{x}}$ are the predicted coordinates of the vehicle;

  - $\mathbf{x}$ are the ground-truth coordinates of the vehicle;

- **FDE** (Final Displacement Error), the Euclidean distance between the endpoints of the real and the predicted trajectories:

$$FDE = dist\big(\mathbf{x}(H), \hat{\mathbf{x}}(H)\big), \tag{4.15}$$

To perform the memory requirements comparison between the proposed prediction system and the SGNet it is just employed the function "summary" of the "pytorch-summary" library. This returns the number of parameters and the size of the ANN onto which the function is applied.

4.4 Enhancement

The system presented in Section 4.1 basically aims at instantiating a digital twin of the driver able to move ahead in time to predict the human behavior. Having said this, it is intuitive that the introduction of a method to adapt to different drivers will consistently benefit the predictor.

Several ways of doing this have been considered, and a first implementation is proposed in Figure 4.13, envisioning an enhancement of the proposed system involving a profiling block. More in detail, the final system should work as follows:

- the driver senses the **state of the real environment** and performs **actions on the vehicle**. Data concerning this are collected in a **historic driving dataset**;

- a **Profiler** computes a **similarity metric** between data in the **historic driving dataset** and **driving data of several RL agents**, each one of these trained with different parameters in order to show different behavioral tendencies (e.g. nominal driving, aggressive driving, etc.). This should be done at the end of the **warm-up phase** in which no prediction is performed[13] but the human driver's data are collected to profile him;

- after the similarity metric computation, the **Predictive system** overviewed in Section 4.1 will take in its core the RL agent returned by the profiler, the **most similar to the human** to predict. From this moment on the system will work as previously described, moving the RL agent ahead in the **learned environment model** and saving the future actions.

[13] Alternatively, it is already possible to output predictions, choosing an agent among the candidates, at random or with other predefined criteria, accepting lower performance.

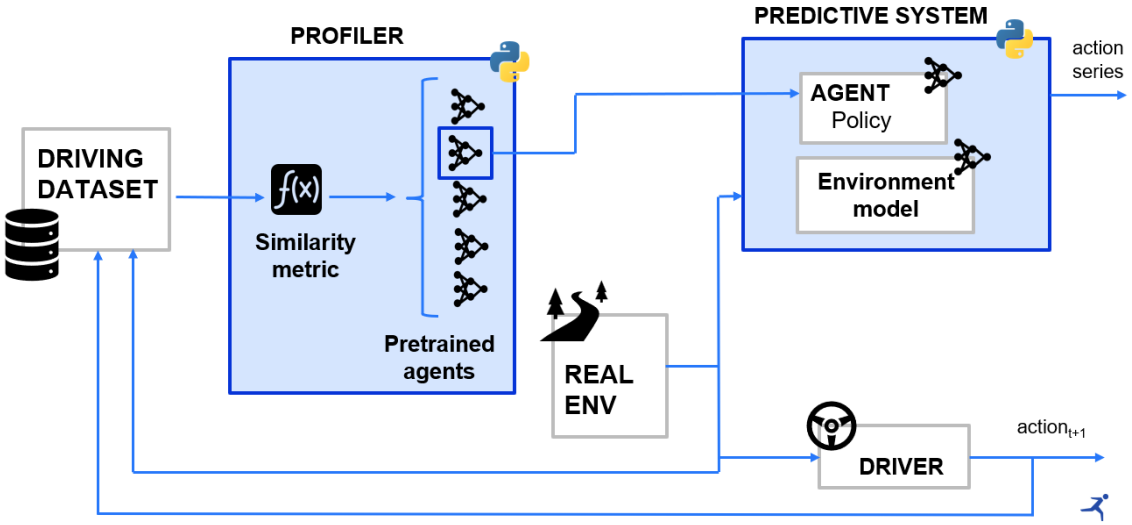

Figure 4.13: Enhanced prediction system, envisioning a profiling block upstream.

4.4.1 Similarity metric

A key design choice in the definition of the Profiler is the metric exploited for profiling the driver. For this first proposal of enhancement, it is employed the **Believability precision**, introduced in [1] and improved in [2][14].

The rationale behind this metric consists of looking at the frequency with which the agent makes the same decision as the human driver in the same driving scenario.

Formally, the believability precision of a driver's behavior is defined as the ratio between the number of events in which the agent performs the same action as the human driver when encountering the same driving state and the number of driving states encountered by both the expert and the agent:

$$Belivability\_precision = \frac{\#(same\_decision\_of\_human\_driver)}{\#(driving\_states\_encountered\_by\_human\_and\_agent)}, \tag{4.16}$$

where the "#" symbol stands for "number of", representing a counting operator.

4.4.2 Experimental set-up

The experiments focusing on the Profiler involve the computation of the Believability precision to choose one among the three agents available after the MBPO training discussed in Section 4.2.2 as the digital twin of the driver. To create the agents' driving datasets, 3 random tracks are generated and each RL agent is tested on these. Having done this, it is necessary the definition of the **human driving dataset** for

[14]In fact, in these works the authors refer to the "believability precision", and to an improved version naming it "believability precision refined". Being a twofold analysis out of the scope of this thesis work, it is used as believability precision its refined version, dropping the "refined" appellative.

profiling through Believability precision.
To understand if part of the data can be representative of all the driving data belonging to a human driver the test involves three sizes for the profiling split for each driver:

- single curved track: **10% of samples**;
- train and validation sets of the SGNet: **68% of samples**;
- all tracks: **100% of samples**. This is the upper bound to compare against. Indeed, the previous split sizes can be defined as sufficient for profiling if the RL agent maximizing the Believability precision with a human is the same independently from the split size.

This representativeness of smaller datasets is linked to the length of the warm-up phase mentioned in Section 4.4. In fact, the lower the data required for profiling, the faster it is possible to start outputting high-quality predictions.

Having done this, the Believability precision will be defined as a successful profiling metric if the agent maximizing it is also the one returning the best actions and trajectory predictions when used at the core of the predictive system.

Chapter 5

Experimental results

This chapter presents the results obtained from the experiments introduced in Sections 4.2.2 and 4.3.4 and summarized in Table 5.1.

Experiment focus	**Summary**
MBPO agent	Analysis of the advantages of a model-based framework, obtained through a model-free algorithm (SAC) and its model-based version (MBPO).
SENSOR Predictor	Analysis in performance for Action Prediction task and comparison with SOTA for Trajectory Prediction task. Comparison on memory requirements.
Profiling block	Analysis of believability precision as profiling metric computed on different profiling split sizes.

Table 5.1: Experiments summary

5.1 MBPO Agent

As mentioned in Section 4.2.3 the evaluation metrics characterizing this section focus on the efficiency of the Reinforcement Learning framework and are summarized in Table 5.2.

The results concerning **sample efficiency** are shown in Figure 5.1, with the **average reward** and its **standard deviation** measured on three runs of 30 000 steps each, with a frequency of evaluation of 5 000 steps.
As it is possible to notice the introduction of the environment model speeds up learning allowing for higher performance. MBPO reaches first its best results, allowing for earlier training stopping at 25 000 steps, which is at the core of the concept of sample efficiency. Consequently, in an industrial context where the collection of data could be demanding the introduction of an environment-model is beneficial from this standpoint. Furthermore, it is possible to draw some conclusions about robustness by looking at the standard deviation of the reward. For both models this is quite

Metric	Summary
Sample efficiency	Reward collected in three evaluation episodes after a fixed number of steps in the real environment.
Compute efficiency	Average time for computing a fixed number of steps, measured on the same working machine.

Table 5.2: Overview of the metrics involved in evaluating the MBPO framework

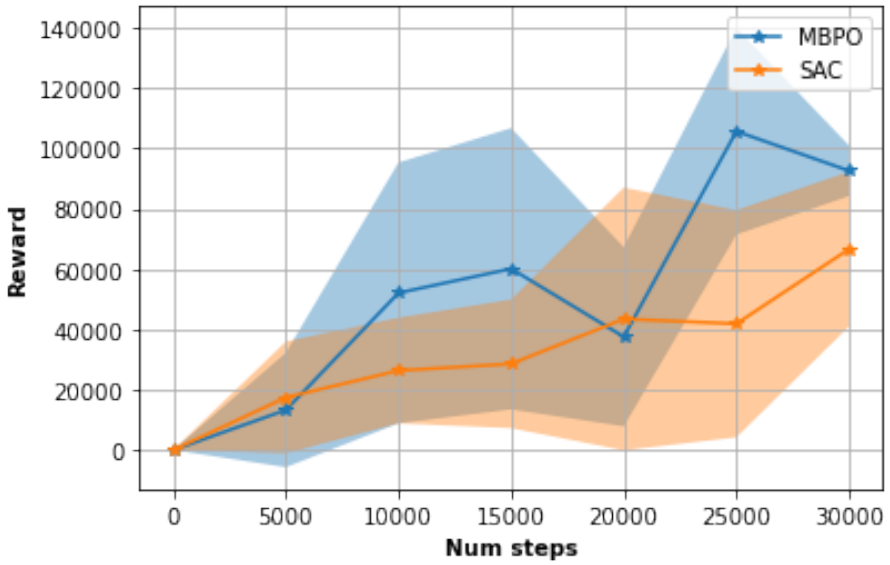

Figure 5.1: Sample efficiency comparison between MBPO and SAC.

high, as it is typical in Reinforcement Learning frameworks: in this paradigm the learning phase is more variable since the agent is trained on data collected by himself, differently than in Supervised Learning contexts. In fact, if an agent happens to end up in bad environment spots, the learning will not benefit from this. It is reasonable to affirm that in a Model-Based context, where the environment model is learned, this agent learning instability will be even harsher because of an additional source of bias: data come from a learned function approximating the environment. This can explain why in the first part of the training there is a larger variance in the performance: the environment model is not yet well approximated and introduces additional noise.

Regarding **compute efficiency**, for computing 5000 steps on the machine summarized in Table 5.3 MBPO needs in average 1h 45 min, while SAC needs 1h 10 min, so the **epoch MBPO-SAC compute time ratio** is 1.5.

Hardware	Summary
CPU	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
GPU	NVIDIA GeForce MX450 @ 2048.0MB

Table 5.3: Overview of the hardware specifications of the working machine employed.

Metric	Summary
MAE	Absolute error of each signal averaged over the prediction window.
MSE	Squared error of each signal averaged over the prediction window.
Sign Accuracy	Accuracy in guessing the sign of each signal averaged over the prediction window.
WAE	Absolute error of each signal at each timestep in the prediction window.

Table 5.4: Overview of the metrics involved in evaluating the Action Prediction task.

Metric	Steering (rad)	Throttle (%)	Brake (%)
Window MAE	0.15 ±0.11	0.22 ±0.18	0.01 ±0.02
Window MSE	0.08 ±0.09	0.14 ±0.16	0.01 ±0.02
Sign accuracy	0.56 ±0.07	0.87 ±0.18	0.98 ±0.03

Table 5.5: Driver Action Prediction overall results.

5.2 Prediction system

As mentioned in Section 4.3.5 the evaluation metrics characterizing this section focus on the quality of the predictions of the proposed framework in the two tasks of Driver Action Prediction and Driver Trajectory Prediction.

5.2.1 Driver Action Prediction

A summary of the metrics involved in the evaluation of this task can be found in Table 5.4.

Table 5.5 shows average and standard deviations of the results computed over all the collected driving tracks.

In particular, looking at the intuitive MAE and combining this information with the one concerning sign accuracy it is possible to draw several conclusions on each driving command forecasting capability. Having done this, it is possible to zoom

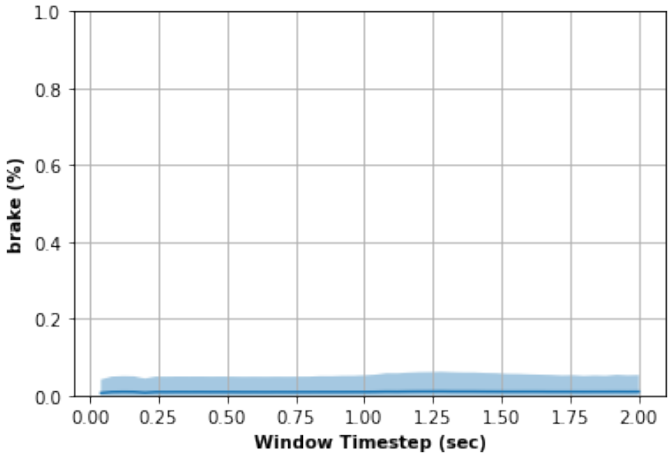

Figure 5.2: Window Absolute Error of brake command.

into the evolution of performance with the evolution of time in the widow prediction by looking at the graphs concerning Window Absolute Errors.

Brake

Brake MAE shows how the system performs practically **null errors when predicting the driver braking action** and Figure 5.2 shows that this is independent of how ahead in the prediction window. This was an expected result, testing in extra-urban scenarios with no other agents to interact with so that the usage of the brake pedal is limited to difficult curves, and it should be easy for the agent the acquire such usage behavior.

Throttle

Concerning the **throttle pedal pressure**, a higher error is observed. A first hypothesis is that this is due to the fact that the **RL agent has a decision frequency much higher than the human one**. In fact, the agent perceives the environment every 0.04 sec and can concurrently act. Conversely, human reaction time to visual stimuli [1] is around 0.19 sec [62]. This implies a smoother driving style of the human, and the RL agent frequent adjustments will result in errors. Also in this case, as in the brake pedal forecasting one, Figure 5.4 shows that our prediction quality is independent of the timestep in the prediction window.

However, while in the braking case it is reasonable to affirm that for the majority of the time both human and RL agent were performing no braking, in the throttle pressure case they both are acting but with different intensities, and hence the errors. The fact that the error is constant can imply that the proposed system is

[1]The focus is on this kind of stimuli only because during data collection the driver can only see how it moves on the track, the simulator does not provide any sound information. However the hypothesis advanced would still hold in the case of sound presence since the same referred source states that the reaction time to this kind of stimuli is 0.16 sec on average.

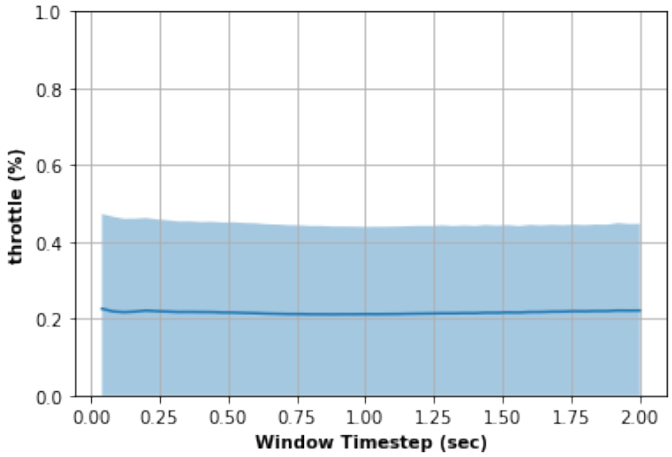

Figure 5.3: Window Absolute Error of throttle command.

good at forecasting future environment conditions so that the error depends only on the different styles between human and RL agent.

Finally, focusing on the throttle and brake sign accuracy in Table 5.5, they are counter-intuitively not equal. In fact, given the way the action space was engineered (Section 4.2.1), these actions can not be performed at the same time. However, it is not true that when the agent is not braking it is necessarily accelerating and vice-versa: the **symmetry is broken by the null action**. Consequently, the 87% capability of our system of predicting an acceleration is not alarming, since **in the majority of the cases that it makes a mistake it is not predicting a brake but a null action**.

Steering

The decision frequency argument can explain the error in forecasting the **steering command** as well. However, observing the evolution in time of the error from Figure 5.3, this presents a counter-intuitive profile: higher errors are committed in forecasting actions nearer in time. This error could be due to the **type of data** available to the proposed system. In fact, considering the state of the environment as perceived by the human, whose sensing system consists basically of his eyes, he has information about the future evolution of the surroundings. Consequently, he will anticipate his maneuvers, even hypercompensating in the opposite directions. To better visualize this, let us consider the example of a curve to the right: a human driver, before undertaking it, will slightly move to the left, in the opposite direction. Conversely, our agent sensing system can provide only information about the immediate surroundings, not allowing the preparatory maneuvers, thus it is possible to observe steering angles in the opposite direction of the ones performed by the human. This hypothesis is strengthened by looking at steering accuracy in Table 5.5, implying that the human and the RL agent turn in the same direction a little more than half of the time.

The conclusion is that it is unthinkable to engineer a highly performing steering

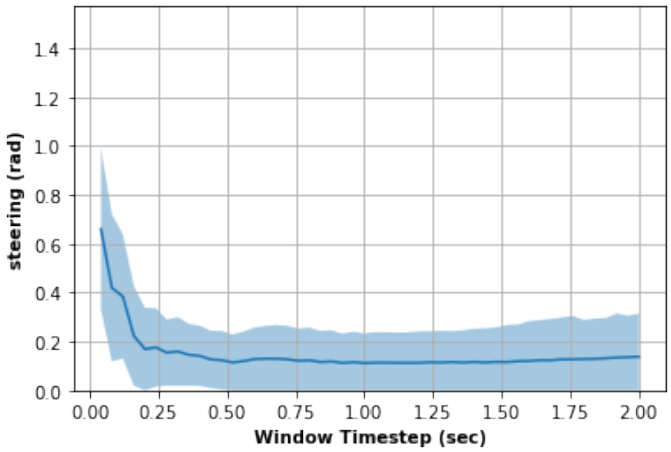

Figure 5.4: Window Absolute Error of steering command.

Metric	Steering (rad)	Throttle (%)	Brake (%)
Window MAE	0.13 ±0.10	0.23 ±0.19	0.01 ±0.02
Window MSE	0.07 ±0.10	0.16 ±0.18	0.01 ±0.01
Sign accuracy	0.56 ±0.06	0.85 ±0.20	0.99 ±0.02

Table 5.6: Driver Action Prediction driver A results.

Metric	Steering (rad)	Throttle (%)	Brake (%)
Window MAE	0.22 ±0.11	0.15 ±0.12	0.01 ±0.01
Window MSE	0.12 ±0.08	0.09 ±0.11	0.01 ±0.01
Sign accuracy	0.58 ±0.01	0.93 ±0.08	0.96 ±0.05

Table 5.7: Driver Action Prediction driver B results.

action predictor without information about the future evolution of the environment, and the introduction of data proving it is reserved for future enhancements of this prediction system.

Considerations on adaptability

It is also possible to deepen this analysis by dividing the results per driver: Table 5.6 reports the results of driver A, Table 5.7 reports the results of driver B, and Figure 5.5 reports their respective analysis in time through the WAE metric.

These tools permit to visualize how it is easier to predict the pedal pressure behavior of driver B, in particular of the throttle one, having already discussed the ease of braking forecasting in the use case scenario. Conversely, there is a higher performance in the steering forecasting of driver A, both from the average error standpoint and the standard deviation one, meaning that predictions precision and robustness can vary from one driver to another. Consequently, although this thesis project does not explicitly encompasses adaptability at this stage of development, it highlights the necessity of this feature, putting the basis for future improvements in this direction.

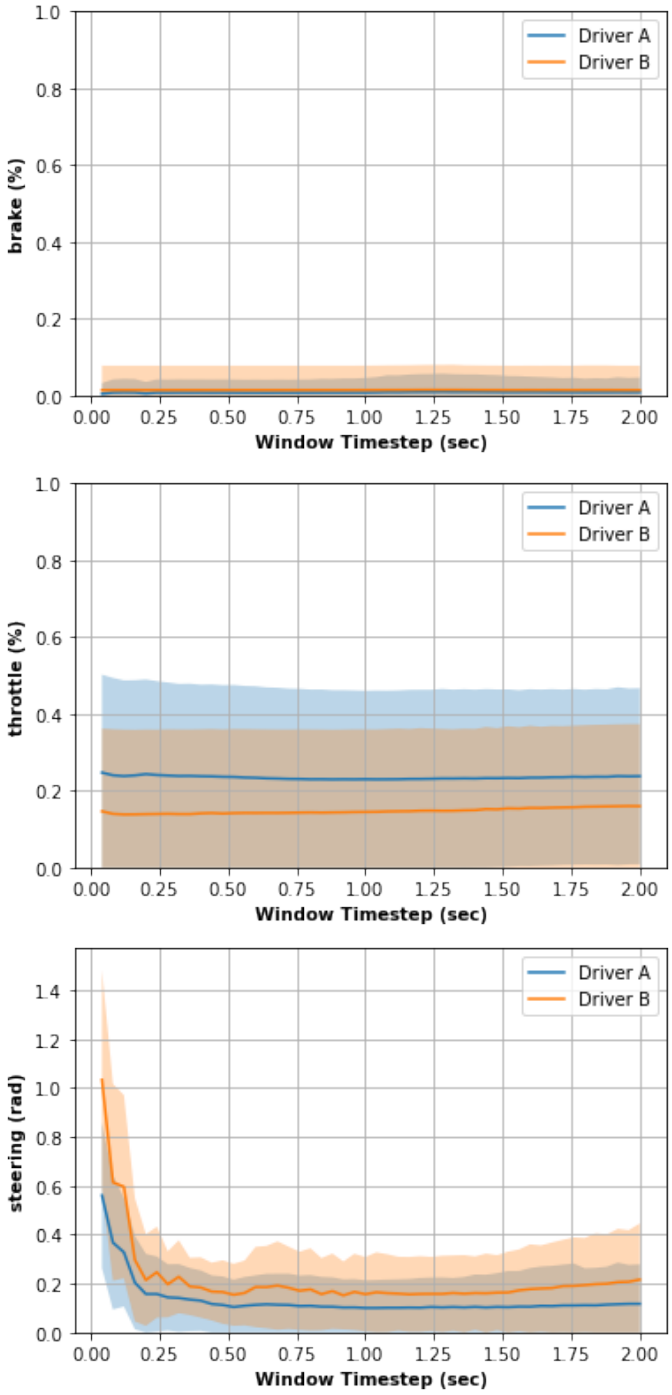

Figure 5.5: Window Absolute Error of the actions focusing on two different drivers.

5.2.2 Driver Trajectory Prediction

A summary of the metrics involved in the evaluation of this task can be found in Table 5.8.

Metric	Summary
ADE	Euclidean distance between the real and the predicted trajectories.
FDE	Euclidean distance between the end-points of the real and the predicted trajectories.

Table 5.8: Overview of the metrics involved in evaluating the Trajectory Prediction task.

Model	ADE (m)	FDE (m)
Ours	7.05	14.55
SGNet, H=1	86.05	101.52
SGNet, H=10	7.73	8.35
SGNet, H=25	4.84	5.54

Table 5.9: Driver Trajectory Prediction overall results.

Prediction quality

Before diving into the final performance of the proposed prediction system and of the SGNets trained for comparison, the curves relative to the training of these last ones are presented. Figure 5.6 involves both ADE and FDE on the validations set used by the SGNets, showing how the training was stopped after each ANN plateau was reached.

In particular, to stop training each model must observe **no improvement for at least 5 epochs**[2] in ADE computed over the validation set.

This graph provides insight into the nature of the predictions as well. Indeed, ADE and FDE are almost superposed, independently from the quality of predictions, which depends on the dimension of the input timeseries. This implies that each prediction in the time window does not keep into account the previous error and the forecasting accuracy is almost independent of how far in the prediction window we are.

Conversely, Figure 5.7 shows that the proposed prediction system's error in forecasting future positions increases almost linearly with the prediction horizon. This is intuitive since as described in Section 4.1 to perform predictions an agent moves ahead in time propagating its actions and states so that the error cumulates in time. However, this does not imply that the overall performance of the system is worse than the one of the SGNets.

Table 5.9 shows a comparison of the prediction system with the SGNets with different input timeseries dimensions, noted as "H".

The presented prediction system average error in the prediction of the position is 7.05 m, while the error in the final position prediction is 14.55 m, almost double,

[2]Actually, the strict threshold of 5 epochs is kept only in the case of the SGNet with input timeseries of 25 samples since each epoch requires hours of training. For the lighter remaining models, training also further this patience parameter was allowed.

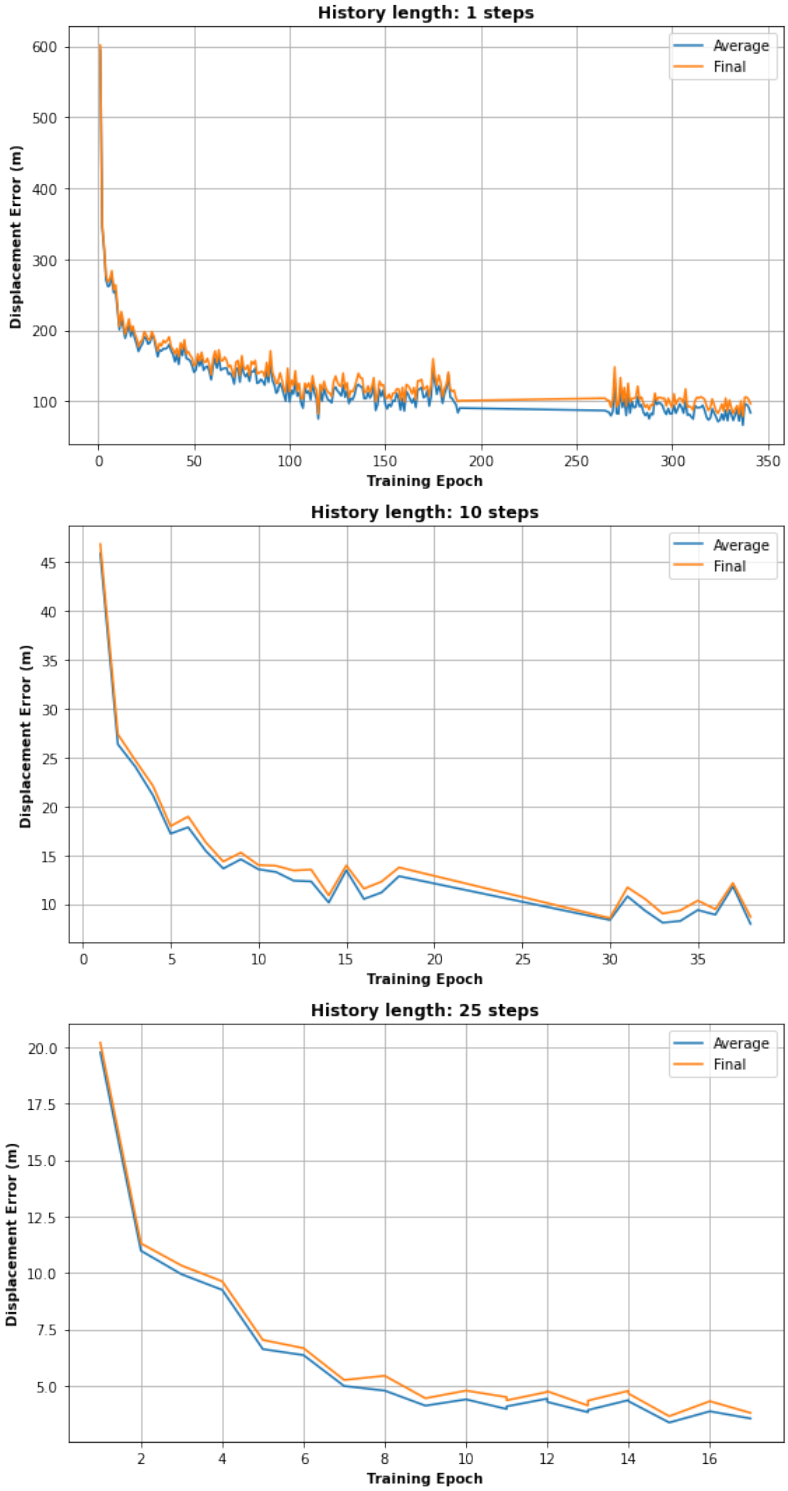

Figure 5.6: SGNet validation ADE and FDE.

as expected for the previous considerations. The SGNet with **no historic information**, namely working with the exact same data of the proposed predictor is basically **incapable of performing Trajectory Prediction**. To reach the same performance in ADE the SOTA model needs in input timeseries of 10 samples in

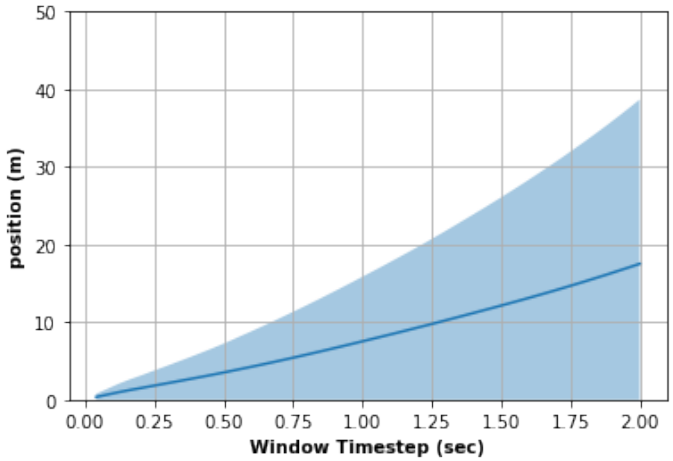

Figure 5.7: Window Absolute Error of vehicle position.

Model	Num Parameters	Memory requirements
Ours	610 628	2.33 MB
SGNet, H=10	193 503 182	754.98 MB

Table 5.10: Comparison on memory requirements.

the past[3], while it outperforms our model from the FDE standpoint. Finally, the SGNet surpasses the proposed prediction system also on the ADE metric if given 25 samples in the past, namely half of the future prediction window dimension.

Memory requirements

The final comparison between the proposed system and the SGNet is done from the memory requirements standpoint. Considering the backbones described in Section 4.2.5, there are:

- the policy backbone counting 70 916 parameters, requiring around 0.27 MB;
- two Q-functions backbones each counting 70 656 parameters, requiring around 0.27 MB;
- eight environment-model functions backbones each counting 49 800 parameters, requiring around 0.19 MB.

Summing up all the previous items, our prediction system is characterized by **610 628 parameters**, requiring around **2.33 MB** of memory. Conversely, considering the SGNet with H=10, the smaller among the successful ones, it is characterized by **193 503 182 parameters**, requiring **754.98 MB** of memory. To summarize, in the Trajectory Prediction tasks there are comparable results with such a SOTA model, with a system weighting two orders of magnitude less.

[3]Counting the present sample.

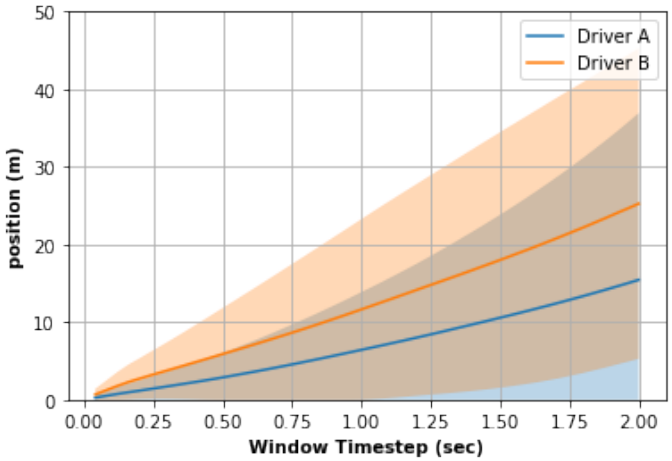

Figure 5.8: Window Absolute Error of vehicle position focusing on two different drivers.

Model	ADE (m)	FDE (m)
Ours	6.55	13.08
SGNet, H=1	73.60	84.00
SGNet, H=10	5.81	6.45
SGNet, H=25	3.51	4.05

Table 5.11: Driver Trajectory Prediction driver A results.

Model	ADE (m)	FDE (m)
Ours	7.85	16.40
SGNet, H=1	86.32	101.57
SGNet, H=10	7.91	8.48
SGNet, H=25	4.82	5.52

Table 5.12: Driver Trajectory Prediction driver B results.

Considerations on adaptability

To conclude this analysis, it is possible to separate the results by the driver as done in the Action Prediction study in Section 5.2.1. Also in this case both the SGNet and the proposed predictor have different performances on different drivers, specifically encountering more difficulty in predicting driver B's position. Recalling from the Action Prediction analysis that this driver was the one whose steering action was more difficult to predict, a hypothesis is that this increased error is correlated to the lateral component of the motion.

It is straightforward to conclude that a form of adaptability to the driver is required independently from the prediction task to perform.

Metric	Summary
Believability precision	Ratio between the number of events in which the agent performs the same action of the human driver when encountering the same driving state and the number of driving states encountered by both the expert and the agent.

Table 5.13: Overview of the metric involved in the Profiler experiments.

5.3 Profiling results

Table 5.13 provides a description of the only metric involved in this experimental section, the Believability precision.

Figures 5.9 and 5.10 show the Believability precision computed on the driving track of the three RL agents and of drivers A and B, with different profiling splits sizes.

First, it is possible to notice that considering all the data, corresponding to the 100% split, the most credible agent for drivers A and B is different. Furthermore, it is possible to notice that **smaller splits are not always representative** of the driver. In fact, looking at driver A graph we observe that the agent more similar to this driver changes if considering all the data or part of these. This can reveal a first flaw of the similarity metric under analysis, consisting of a lack of robustness. However, to confirm this further analysis with larger datasets is needed since the one considered consists only of tracks lasting a few minutes: a real system mounted on a car could have at its disposal data of several hours of drive.

Finally, it is found that the Believability precision is not the metric to put at the center of the Profiler since for driver B it suggests using the candidate agent "mbpoConfig_220_6", but the performance of the predictor using it as the digital twin of the driver is lower with respect to the ones of "mbpoConfig_226_6"[4], both in the action and trajectory prediction tasks. However, we present this result convinced of the methodologic validity of the enhanced system proposed in Section 4.4, envisioning a profiling block, and future works will research a more appropriate profiling metric.

[4]The results reported in Section 5.2 envisioned Predictors using this agent as their core.

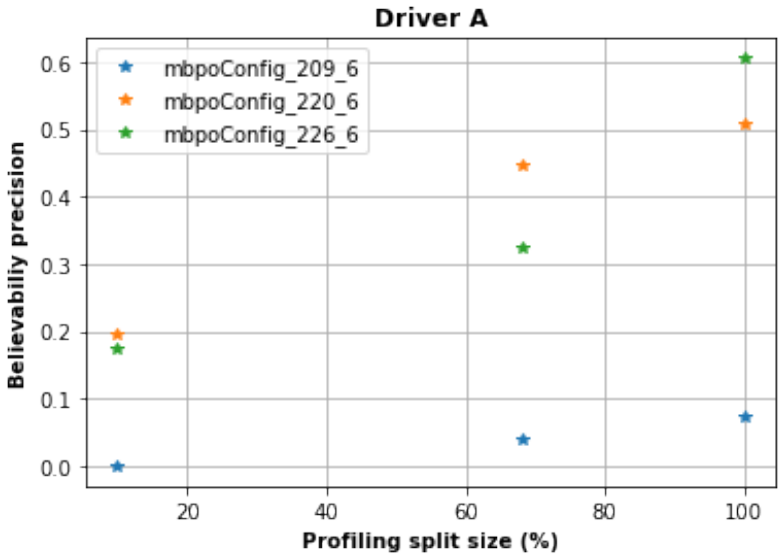

Figure 5.9: Believability precision for driver A computed on different profiling splits.

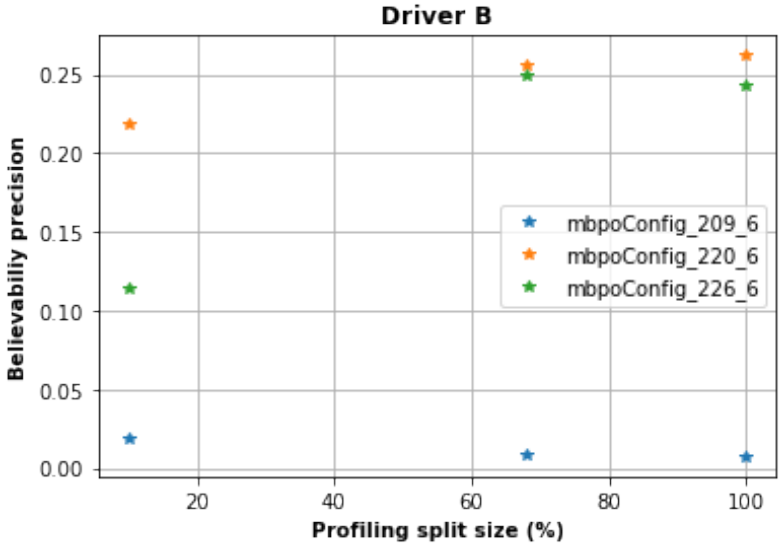

Figure 5.10: Believability precision for driver B computed on different profiling splits.

Chapter 6

Discussion

The present chapter summarizes some high-level considerations drawn throughout the development of this thesis project.

6.1 PPO vs SAC for MBPO

As mentioned, the Lane Keeping task, characterizing the environment that the RL agent needs to beat, was already faced in previous works [2] employing a model-free algorithm, PPO [59]. Considering that MBPO is a model-based algorithm that exploits the environment model for data augmentation for a model-free algorithm, it is virtually possible to use PPO instead of SAC as a learning algorithm for the agent.

Although some developing steps were performed in this direction, these were interrupted, because of a theoretical argument. In fact, PPO is a Policy Gradient algorithm that improves the agent's policy after having performed rollouts of several steps and computed an expectation of the return on these. On the other hand, one of the main points on which the authors of MBPO [61] insisted is that to maximize the advantage of the environment model exploitation, an approach is to minimize the length of the rollouts in the learned environment model. They state that one-step rollouts, consisting of a single state transition, are a difficult baseline to beat. This suggests keeping SAC as model-free algorithm at the core of MBPO given that this algorithm uses a learning approach different from the PPO one. As reported in Section 3.3.4, SAC collects samples to evaluate the quality of being in a state and then optimizes the policy accordingly. Consequently, it is not necessary for these samples to come from the same long rollout: they can come from several randomly initialized starting states and this allows for exploiting one-step rollouts.

6.2 Decision frequency

The RL agent developed throughout this thesis project is characterized by a sample rate of 25 Hz. This means that it receives input from the environment every 0.04 sec and concurrently acts. This has a repercussion on the Predictor as well: when moving the agent ahead in time it is necessary need to feed him with data arriving at the

same frequency. This means that the environment model is frequently interrogated[1] and therefore every error with respect to the real environment will cumulate.

Considering the experiments proposed in Section 4.3.4 a prediction window of 2 sec was characterized by the considerably high number of 50 samples. To avoid this error propagation an idea would be that of reducing the decision frequency, and therefore the sample rate. This is reasonable since the already considered argument that human reaction time to visual stimuli is 0.16 sec [62], so a human will act with a decision frequency of at most 6.25 Hz. Unfortunately, this would come with a drawback. The proposed system, indeed, requires a first RL training and lowering the sample rate would increase the difficulty of the task performed by the agent, decreasing its performance. Practically, there is a trade-off between the performance of the agent and the propagation of the error when exploiting a learned environment model.

6.3 RL outcome variance and human behavior

As already discussed, independently of model-free or model-based, Reinforcement Learning can lead to very different outcomes when training an agent. This is due to the fact that differently from Supervised and Unsupervised Learning, there is no dataset leading the learning process. The agent learns and collects data simultaneously so that bad policies can lead to bad data and so on in a vicious cycle. This problem is softened by several techniques aiming at improving "exploration", namely the collection of data as representative as possible of the state and action space. For instance, SAC, as reported in Section 3.3.4, is characterized at its core by an optimization problem that maximizes the expected reward and the "randomness" (entropy) of the policy. In spite of this, the results shown in Section 5.1 still show considerably varying performances of the trained agents, whose only difference is the random seed.

However, for the purpose of the presented prediction system, this randomness of the outcome can be exploited. In fact, having different agents characterized by different behaviors can be an advantage when wanting to use one of these as a digital twin of the driver. In fact, a future improved version of the predictor can incorporate agents trained in different ways to represent different driving states such as aggressiveness, impaired conditions, etc.. Let us assume for a moment that it does not exist a way to train an RL agent to represent these states directly. In this case, launching a sufficient number of RL trainings with different random seeds will return very different agents, that could be labeled as aggressive, impaired, etc. a posteriori because resembling those driving states even though not explicitly trained for it.

[1] The agent performs an action, the environment model returns a state.

6.4 Learning vs Enforcing

Previous works [2] used a reward function term to prohibit the possibility of accelerating and decelerating at the same time:

$$R_{Unfeasible} = \begin{cases} -1 & \text{if accelerate and brake} \\ 0 & \text{else} \end{cases}$$

However, when exploiting SAC, this approach revealed unsuccessful. A fist hypothesis was that by substituting this term with a derivable one learning would have been easier since performed through gradient descent. One of the attempts is given by the following function:

$$R_{Unfeasible} = -\sqrt[3]{acc \cdot brake},$$

and represented in Figure 6.1.

The undesired behavior, however, was still presented by the agent. Finally, the constraint was directly enforced in the action space (Section 4.2.1), substituting the two vector entries of throttle and brake, with only one so that an action would exclude the other. This experience allows concluding that if there exists a method for enforcing at design time a desired behavior in an RL agent, this should be preferred to a learning technique.

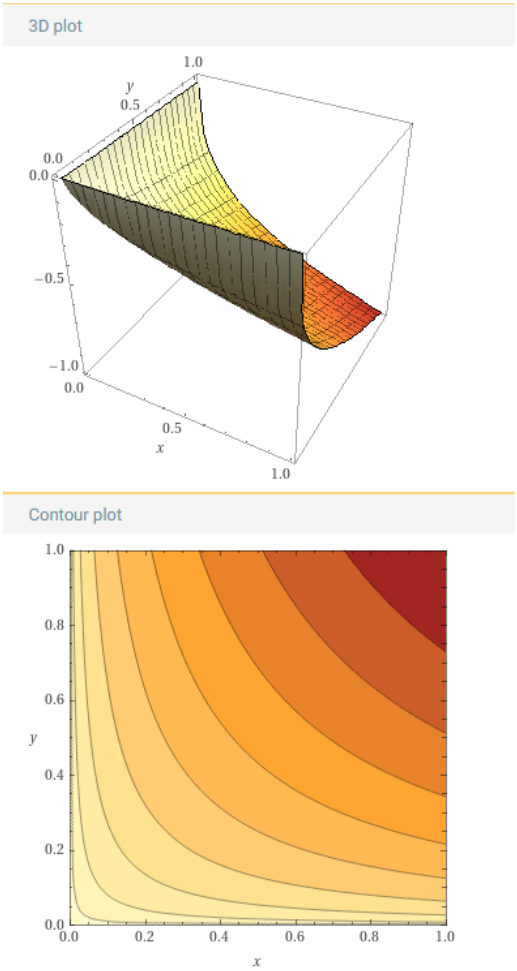

Figure 6.1: Representation of a reward function aiming at prohibit concurrent acceleration (x) and brake (y).

Chapter 7

Conclusions

This final chapter summarises the conclusions of this thesis work and proposes several ways to direct the research concerning the prediction system presented.

7.1 Potential and limits

7.1.1 Environment-model exploitation and design

The work presented allowed first to confirm how a model-based framework in Reinforcement Learning can help speed up training and return higher performance within the context of an industrial application. In the specific case, the exploited algorithm, MBPO, envisions an environment model learned from past experience. This feature can be double-edged and needs careful analysis at the moment of designing a system for future deployment. A learned model has certainly the potential of modeling environments arbitrarily complex since the introduction of new elements[1] depends only on the availability of data concerning them. However, as said, this is a "potentiality", it is just virtually possible: such functions may be not robust and then from the practical standpoint could be limited to simple scenarios. In this last case, the introduction of a learned environment model could be useless, and it could make more sense to exploit mathematical models of the environment (not subject to a learning phase).

Probably, as it often happens, third ways are possible, for instance, deploying a mathematical model of the environment, and learning the residual difference with the real environment.

7.1.2 Prediction quality and requirements

Concerning the forecasting of actions, and of the consequent trajectories, the proposed system can output predictions without requiring naturalistic data to be trained on. Practically, the human behavior to replicate is encoded in the reward function,

[1] Examples of this can be new agents on the streets to interact with, complex road profiles, damaged streets, etc.).

which takes the role of the "labels" in Supervised Learning. Consequently, the performance of the learning task moves from the quality of the data to the quality of the reward function. These items are not exclusive though, existing the possibility of improving the goodness of the reward function through data once they become available, as discussed in Section 7.2.2.

From a practical standpoint, the proposed framework can return decent results concerning the controls over the longitudinal movement, namely of the pedal pressures. At the same time, considering the lateral control left to the steering wheel, it is observed how critical it is the introduction of adaptability tools and of data bringing information about the future evolution of the environment. Concerning this last point, differently from the state-of-the-art model (the SGNet), the proposed system can work also on minimal data, consisting of no historic information, so the mentioned data addition needs to still fulfill this minimality criterion if reasoning from the perspective of future industrialization.

7.1.3 Hardware constraints

The hardware requirements theme is another point deserving some focus. In fact, the presented system sacrificed some precision from the forecasting standpoint but decreased the storage requirements by two orders of magnitude. In spite of this, the models are working in the order of Megabytes of size, so an actual implementation still needs to work on the minimization of this. This leads to a first limitation to tackle, the one of hardware constraints.

7.1.4 MBRL tools for industry

Although there exist many reinforcement learning libraries publicly released that provide high-quality implementations of the complex components involved in training RL agents, the vast majority of these focus on model-free reinforcement learning and lack crucial components of implementing MBRL algorithms [58]. This is a difficult obstacle to overcome to open up the streets to the model-based framework, even though it proved its validity several times. In fact, the exploited (and modified) Python library is still under development, having added new algorithms since the beginning of this thesis project. Consequently, industrial applications wanting to exploit these methods can be hampered by higher difficulty in guaranteeing efficiency and robustness.

7.2 Future works

7.2.1 Data integration

A great but easily solvable vulnerability of the proposed system is the presence of data concerning only the vehicle and the immediate surroundings. As discussed in Section 5.2, this implies an RL agent whose behavior will be characterized by some sort of delay in perception with respect to the human one. In fact, humans, through vision, will have information about the future evolution of the environment

and can keep this into account when executing their maneuvers. Consequently, it is paramount to introduce data regarding the road ahead of the driver, to add to the RL state space the information discussed. Possible choices are the following:

- to work directly on the image of the camera, not extracting the distances from the lane and the other features exploited in this work. However, this comes with the drawback of working with high dimensional states space and requires careful addressing when working with a learned environment model;
- to utilize the same type of engineered features used throughout this thesis work, but not limiting them to the immediate left and right of the vehicle. Specifically, it is possible to input a series of lane distances and curvatures with a fixed distance among each other, e.g. take the lane distances of the road ahead every 2 m, drawing a profile of the near track;
- to exploit information from a map.

Furthermore, when modifying the input data, it is possible to think of the incorporation of historic data, such as the past trajectory, as done by the SGNet. However, this approach, introducing information about the past, is believed to be secondary with respect to the one previously mentioned, introducing information about the future environment: a driver, when performing its decisions, looks only ahead and it is reasonable that a successful RL agent for this task shall recall a human.

7.2.2 GAIL

Concerning the development of a system adaptable to different humans, a technique to introduce is referred to as GAIL, Generative Adversarial Imitation Learning [63]. GAIL is based on Generative Adversarial Networks, a framework characterized by the following items:

- a discriminator that classifies whether a given data point is generated by the generator or by the actual data distribution;
- a generator that generates new data points by learning the distribution of the input data set.

In this specific case, the RL agent would act as the generator and the input data set would be given by human-driving data. Practically, there is a loop in which:

- a discriminator ANN is trained to distinguish between human and RL agents sequences of actions and states;
- a generator, namely the RL agent is trained using the discriminator in its reward function. The RL agent receives rewards when the discriminator is wrong, namely when its behavior is indistinguishable from the human one.

Therefore it is obtained from human data a reward function, encoded by an ANN, representing the driver's behavior. The result will be an RL agent imitating the driver whose data were used for GAIL training.

7.2.3 Profiling

An enhanced version of the developed predictor was proposed in Section 4.4, discussing a system capable of adapting to different humans after a profiling procedure. However, the challenge that this methodology has to overcome consists of finding an appropriate profiling metric for associating a pretrained RL agent with a human driver.

Comparing this technique for adaptability to the previously introduced GAIL, profiling has a more limited capability of replicating a driving style, but it could directly be applicated on edge for inference purposes, not requiring any retraining. Consequently, a reasonable approach consists of a system adopting these techniques in a series:

- after a warm-up phase in which driver data are collected, profiling starts;
- a driver is associated with its most similar (in behavior) RL agent, and this is used to output predictions while keeping collecting data;
- a further RL agent is trained in the cloud through GAIL, obtaining the most similar possible digital twin of the driver. Once it is available, it is used for the driver to predict.

7.2.4 Ensemble exploitation

The MBPO technique learns the environment model through an ensemble of ANNs. It is possible to think of several possibilities to exploit this networks ensemble for the proposed system, assuming mainly two directions:

- improving predictions, still keeping a unimodal output. The system outputs only a series of actions (or trajectory), possible to improve by testing several types of aggregations (mean, median, voting, etc.) on the states returned by the environment model;
- proposing a multimodal system, namely outputting more than a series of actions, propagating independently different states and actions across the ANNs ensemble.

7.2.5 Existing framework improvement

Previous paragraphs referred to several additions to insert into our system. However, of course, it is also possible to think of several improvements to various modules characterizing the system or to the general framework involved in its instantiation.

A first objective is to widen the scope of the simulation environment, currently limited to an extra-urban single-track scenario. Examples are the introduction of other agents on the street to interact with or of more complex road profiles, characterized by signals, varying steep and intersections. A further possibility is to leave MATLAB to move towards different simulators, to participate in official benchmarks for instance in the autonomous driving agent development stage. In fact, some first

research has been conducted focusing on tools for moving the developed framework into the CARLA (Section 2.4.2) simulator.

Furthermore, it is possible to insist on the automation of hyperparameters tuning. The authors of SAC showed how this algorithm is relatively stable with respect to hyperparameters modification, except for the relative scale of the reward term and the entropy term. Nonetheless, it is possible a tuning of the hyperparameters characterizing the environment model in MBPO, for instance the imaginary rollout length, the number of steps performed in the environment model at each interrogation[2].

7.2.6 Resources minimization

As mentioned in Section 7.1.3, since the system is designed from the perspective of deployment, it is necessary to implement techniques for minimizing hardware requirements. Examples of these [64] can be quantization, bringing the parameters and inputs of the ANNs from 32-bit floats to 8-bit integers, and pruning, cutting the uninformative links between neurons of the ANNs.

[2]Although it is reported that single-step rollouts are a difficult baseline to beat, they could be non-optimal.

Bibliography

[1] N. Chiapello, "Machine learning for intelligent avatars in virtual reality simulations," 2020.

[2] S. Santia, "Driving style estimation for impaired driving detection," 2022.

[3] O. Olabiyi, E. Martinson, V. Chintalapudi, and R. Guo, "Driver action prediction using deep (bidirectional) recurrent neural network," 2017.

[4] X. Zeng and J. Wang, "A stochastic driver pedal behavior model incorporating road information," *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 5, pp. 614–624, 2017.

[5] Y. Rong, Z. Akata, and E. Kasneci, "Driver intention anticipation based on in-cabin and driving scene monitoring," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–8, 2020.

[6] X. SHI, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. WOO, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[8] S. Liu, K. Koch, B. Gahr, and F. Wortmann, "Brake maneuver prediction – an inference leveraging rnn focus on sensor confidence," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 3249–3255, 2019.

[9] S. Lefevre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *Robomech Journal*, vol. 1, 07 2014.

[10] R. Schubert, E. Richter, and G. Wanielik, "Comparison and evaluation of advanced motion models for vehicle tracking," in *2008 11th International Conference on Information Fusion*, pp. 1–6, 2008.

[11] W. Xiao, L. Zhang, and D. Meng, "Vehicle trajectory prediction based on motion model and maneuver model fusion with interactive multiple models," *SAE International Journal of Advances and Current Practices in Mobility*, vol. 2, pp. 3060–3071, apr 2020.

[12] F. Altché and A. de La Fortelle, "An lstm network for highway trajectory prediction," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 353–359, 2017.

[13] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[14] A. Benterki, M. Boukhnifer, V. Judalet, and C. Maaoui, "Artificial intelligence

for vehicle behavior anticipation: Hybrid approach based on maneuver classification and trajectory prediction," *IEEE Access*, vol. 8, pp. 56992–57002, 2020.

[15] L. Xin, P. Wang, C. Chan, J. Chen, S. E. Li, and B. Cheng, "Intention-aware long horizon trajectory prediction of surrounding vehicles using dual LSTM networks," *CoRR*, vol. abs/1906.02815, 2019.

[16] Y. Zhi, Z. Bao, S. Zhang, and R. He, "Bigru based online multi-modal driving maneuvers and trajectory prediction," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 235, no. 14, pp. 3431–3441, 2021.

[17] H. Zhao, J. Gao, T. Lan, C. Sun, B. Sapp, B. Varadarajan, Y. Shen, Y. Shen, Y. Chai, C. Schmid, C. Li, and D. Anguelov, "Tnt: Target-driven trajectory prediction," in *Proceedings of the 2020 Conference on Robot Learning* (J. Kober, F. Ramos, and C. Tomlin, eds.), vol. 155 of *Proceedings of Machine Learning Research*, pp. 895–904, PMLR, 16–18 Nov 2021.

[18] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, "Home: Heatmap output for future motion estimation," 2021.

[19] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, "Gohome: Graph-oriented heatmap output for future motion estimation," 2021.

[20] N. Deo, E. M. Wolff, and O. Beijbom, "Multimodal trajectory prediction conditioned on lane-graph traversals," 2021.

[21] C. Wang, Y. Wang, M. Xu, and D. J. Crandall, "Stepwise goal-driven networks for trajectory prediction," *CoRR*, vol. abs/2103.14107, 2021.

[22] D. Choi, K. Min, and J. Choi, "Regularizing neural networks for future trajectory prediction via inverse reinforcement learning framework," *IET Computer Vision*, vol. 14, 02 2020.

[23] L. Sun, W. Zhan, and M. Tomizuka, "Probabilistic prediction of interactive driving behavior via hierarchical inverse reinforcement learning," 2018.

[24] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. AAAI*, pp. 1433–1438, 2008.

[25] Z. Huang, J. Wu, and C. Lv, "Driving behavior modeling using naturalistic human driving data with inverse reinforcement learning," 2020.

[26] N. Deo and M. M. Trivedi, "Trajectory forecasts in unknown environments conditioned on grid-based plans," 2020.

[27] Y. Liu, J. Zhang, L. Fang, Q. Jiang, and B. Zhou, "Multimodal motion prediction with stacked transformers," 2021.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[29] G. C. Chasparis, S. Luftensteiner, and M. Mayr, "Generalized input-output hidden-markov-models for supervising industrial processes," *Procedia Computer Science*, vol. 200, pp. 1402–1411, 2022. 3rd International Conference on Industry 4.0 and Smart Manufacturing.

[30] I. Jones and K. Han, "Probabilistic modeling of vehicle acceleration and state propagation with long short-term memory neural networks," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 2236–2242, 2019.

[31] J. Su, P. A. Beling, R. Guo, and K. Han, "Graph convolution networks for probabilistic modeling of driving acceleration," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–8, 2020.

[32] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016.

[33] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.

[34] A. Ess, B. Leibe, and L. Van Gool, "Depth and appearance for mobile scene analysis," in *2007 IEEE 11th International Conference on Computer Vision*, pp. 1–8, 2007.

[35] A. Lerner, Y. Chrysanthou, and D. Lischinski, "Crowds by example," *Computer Graphics Forum*, vol. 26, no. 3, pp. 655–664, 2007.

[36] X. Huang, P. Wang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, "The ApolloScape open dataset for autonomous driving and its application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 2702–2719, oct 2020.

[37] V. Ramanishka, Y.-T. Chen, T. Misu, and K. Saenko, "Toward driving scene understanding: A dataset for learning driver behavior and causal reasoning," 2018.

[38] A. Rasouli, I. Kotseruba, and J. K. Tsotsos, "Are they going to cross? a benchmark dataset and baseline for pedestrian crosswalk behavior," in *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pp. 206–213, 2017.

[39] S. Malla, B. Dariush, and C. Choi, "Titan: Future forecast using action priors," 2020.

[40] A. Rasouli, I. Kotseruba, T. Kunic, and J. K. Tsotsos, "Pie: A large-scale dataset and models for pedestrian intention estimation and trajectory prediction," in *International Conference on Computer Vision (ICCV)*, 2019.

[41] "Complete vehicle model." https://it.mathworks.com/help/physmod/sdl/ug/about-the-complete-vehiclemodel.html.

[42] "Scene interrogation in 3d environment." https://it.mathworks.com/help/vdynblks/ug/scene-interrogation.html.

[43] MathWorks, "Conventional vehicle reference application." https://it.mathworks.com/help/autoblks/ug/conventional-vehicle-referenceapplication.html.

[44] H. Hu, Z. Lu, Q. Wang, and C. Zheng, "End-to-end automated lane-change maneuvering considering driving style using a deep deterministic policy gradient algorithm," *Sensors (Basel, Switzerland)*, vol. 20, 09 2020.

[45] "Highway gym environment." https://highwayenv.readthedocs.io/en/latest/quickstart.html.

[46] "Udacity's self-driving car simulator." https://github.com/udacity/self-driving-car-sim.

[47] MathWorks, "Driving scenario designer." https://it.mathworks.com/help/driving/ref/drivingscenariodesigner-app.html.

[48] AVSimulation, ""avsimulation — avs : Automotive simulation software and solutions." https://www.avsimulation.com/.
[49] Microsoft, "Airsim." https://github.com/Microsoft/AirSim.
[50] "Torcs - open racing car simulator." https://sourceforge.net/projects/torcs/.
[51] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: an open urban driving simulator," *CoRR*, vol. abs/1711.03938, 2017.
[52] "Carla simulator." https://carla.org/.
[53] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.
[54] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines." https://github.com/hill-a/stable-baselines, 2018.
[55] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.
[56] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018.
[57] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *CoRR*, vol. abs/1805.12114, 2018.
[58] L. Pineda, B. Amos, A. Zhang, N. O. Lambert, and R. Calandra, "Mbrl-lib: A modular library for model-based reinforcement learning," *Arxiv*, 2021.
[59] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.
[60] N. Chukamphaeng, K. Pasupa, M. Antenreiter, and P. Auer, "Learning to drive with deep reinforcement learning," in *2021 13th International Conference on Knowledge and Smart Technology (KST)*, pp. 147–152, 2021.
[61] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," *CoRR*, vol. abs/1906.08253, 2019.
[62] W. Welford, J. M. Brebner, and N. Kirby, *Reaction times*. Stanford University, 1980.
[63] J. Ho and S. Ermon, "Generative adversarial imitation learning," 2016.
[64] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," 2021.

List of Tables

List of Figures