POLITECNICO DI TORINO

MASTER's Degree in MECHATRONIC ENGINEERING

Politecnico di Torino

MASTER's Degree Thesis

Reinforcement Learning for MDP-based autonomous driving support system

Supervisors

Prof. Carlo NOVARA

Eng. Milad KARIMSHOUSHTARI

Dr. Fabio TANGO

Candidate

Calogero MAIURI

DECEMBER 2022

Abstract

The mobility sector has been revolutionized in the latest years. One of the most interesting aspects is the autonomous driving challenge that many big players in the automotive field are trying to solve. A good autonomous driving system must accomplish the driving task in a safe and effective way, to correctly replace the human behavior and possibly also to improve it. In this thesis, we propose different approaches to implement a high level autonomous driving support system, capable of understanding the state of the ego vehicle and suggesting, or directly applying, the most desirable behavior in different scenarios. The final model was developed using Reinforcement Learning to train an agent over a Markov Decision Process framework. This method provides a good trade-off between computational cost and versatility. The model has been simulated, tested and validated in a SIMULINK environment, while different key performance indicators have been used to highlight the suitability to handle different driving scenarios.

Commission of **Computer Engineering, Cinema and Mechatronics**

Master's Degree Course in **Mechatronic Engineering**
Artificial Intelligence for Autonomous Driving

Reinforcement Learning for MDP-based autonomous driving support system

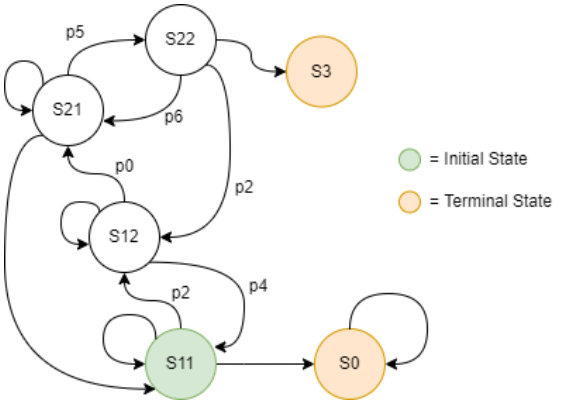
Supervisors
Prof. Carlo NOVARA
Dr. Fabio TANGO
Eng. Milad KARIMSHOUSHTARI

Candidate
Calogero MAIURI

SUMMARY

Recently, the use of artificial intelligence (AI) has become prevalent specifically in the area of autonomous driving. In this thesis, a model for a high level decision making support system is presented, showing the development process, the obtained results and also proposing some possible further improvements, aimed at improving versatility and efficiency of the system. The model is composed of two macro-blocks and several sub-blocks. Each one of these is used to fulfill a specific part of the simulation. The Decision Making block handles the high level control algorithms and reference generation. The implemented controller is using a Model Predictive Control (MPC) approach: the process involves that at each time step t the algorithm minimizes the cost function J over a certain receding time horizon $T = nt$, defined as a multiple of the time step, based on the current state of the system. The computation happens on-line, exploring different predictions. The first step of the optimal trajectory is then implemented as output u, and the process is repeated with the new sampled initial conditions. The output u^* of the NMPC block is then converted into a steering angle δ and a couple of longitudinal forces F_f and F_r, one for each axle, and used in the Vehicle Dynamics block, where a dual track model is used to simulate the vehicle behavior. Finally, the computed dynamic of the EV is used as extra information to be implemented inside the Scenario Reading block, that has the purpose of simulating the behavior inside a predefined environment.

A first and simple approach to automate the decision-making process was carried out by means of a finite state machine, implemented through a StateFlow chart in MATLAB. While this approach is simple to implement, it's hard to scale up and has virtually no versatility.



In order to develop a more complex model of autonomous decision making, Markov Decision Processes are introduced as a tuple of 4 different elements (S, A, P_a, R_a) where S is the state space, A is the action space, P_a is the probability of transition from a state to another state, given an action a, R_a is the reward obtained by performing a transition. The objective, is to use Reinforcement Learning to find the optimal policy π^* that maximizes a cumulative reward function. The simulated scenarios present several cars (actors) in a multi-lane road environment. Based on the conditions of the EV, lanes and actors, it's possible to define all the potential states of the system and the related transitions that happen by performing one of the allowed actions (left/right lane change and lane keeping) . The formal definition of rewards and transition probabilities is achieved by means of multi-dimensional $(n \times n \times m)$ matrices, whose dimensions are related to the carnality of state and action spaces.

Among the possible ways for determining the optimal policy, Dynamic Programming uses the Q-learning function as an intermediate in the case where the transition probabilities and the reward function are completely known. Given a discrete state space S and a discrete action space A, the agent at time step t can move in the world environment, transitioning from state s_t to state s_{t+1} by picking the action a_t. Doing this, it receives an immediate reward r_t that can either be positive, negative or zero. These steps are repeated for $t+1, t+2, t+3...$ until the agent reaches a terminal state s_{ts} (defined a priori). The algorithm has to solve for the maximum of the *discounted expected return function* $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where r is the reward at each time step t, and $\gamma \in (0,1)$ is the the *discount factor*. The Q-value function is then defined as $q_\pi(s,a) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a]$. This function quantifies the quality of a certain state-action pair when following a policy π. It is proven that under the optimal policy π^* the *Bellman Optimality Equation* $q^*(s,a) = E[R_{t+1} + \gamma max_{a'} q^*(s', a')]$ is satisfied. During the training then, the critic is instructed to find the optimal values of the Q-value function by updating them iteratively. These parameters are stored in a human-readable Q-table format.

At each episode, the agent either explores the environment or exploits what it has already learned. The trade-off between these policies is managed by an *Epsilon Greedy* algorithm, where the probability of exploring is exponentially decreased over time with a law of the form $\epsilon = \epsilon_0 e^{-\lambda t}$.

The MDP approach allows for a shifting from a model defined in the continuous domain, to a discrete one. This decoupling is extremely useful in simplifying the process of decision making, but it may also have some drawbacks. The main point towards the use of an MDP state approach is the low requirement in computational power: this is an important element to consider because it can allow the use of a similar model not only for cars but also other kinds of low-power machines and devices, where there may be constraints related to production costs, efficiency, heat dissipation, or weight. On the other hand, the state definition must be complete, in the sense that every possible situation should be included as a main state or an aggregate state. The transitional probabilities are easily accessible to check the immediate and cumulative reward difference when a certain event changes its appearance frequency. It is also important to remember that the reliability of an MDP model is strictly related to the accuracy of the perception and interpretation layers of the AI structure.

One last non-technical consideration has to be made about the paradigm shift when moving from a human-controlled system to an AI-managed one. Mimicking a non-deterministic human behavior may not be safe or efficient in some cases, since we're dealing with a high risk environment where safety must be absolutely granted. In critical situations, it's still not entirely clear whether to emphasize a more ethical or utilitaristic policy.

The trained agent has been tested inside a simulated MATLAB-SIMULINK environment over a randomly generated dataset. Different scenarios of increasing complexity were used to test the stability of the control algorithm and the reliability of the trained policy function. The model reported an optimal decision making performance, correctly defining the states and applying the best action in the totality of the cases. The simulated maneuvers include overtaking, car following, observing safety distance and incoming vehicles, and picking a trade-off between travel time and fuel consumption based on external factors.

Some next steps that could improve the performance and versatility of the model include validation on real-world data and a hybrid approach implementation supported by the use of traditional neural networks.

Table of Contents

List of Tables

List of Figures

Acronyms

ACC
adaptive cruise control

AI
artificial intelligence

EV
ego vehicle

KPI
key performance indicator

MDP
Markov Decision Process

NMPC
nonlinear model predictive control

NN
neural network

PV
preceding vehicle

RL
reinforcement learning

Chapter 1

Introduction

Technology and engineering have become present in most of the field of human lives. Among these, in the last ten to twenty years, the automotive sector has seen a big increase in the usage of different kind of technological assistance, with the aim of improving the driving experience. Suffice it to think how common is it for a crash to be avoided thanks to ADAS systems or how easy it is to plan a trip on a map thanks to GPS.

Recently, the use of artificial intelligence (AI) has become prevalent specifically in the area of autonomous driving. Even if we're still far from the the levels of perfection needed for a mass scale deployment, more and more tests are being conducted on the road with excellent results, in terms of safety but also driving performance.

In this thesis, a model for high level decision making is presented, showing the development process, the obtained results and also proposing some possible further improvements, aimed at improving versatility and efficiency of the system.

The objective of the work is to develop a simple but effective decision making model, used to support the EV during the choice of the best action to perform inside the simulation environment. The During the study, different approaches to the problem have been evaluated and tried:

- Finite state machine approach
- Reinforcement Learning approach, based on a Markov Decision Processes framework

In order to compare the effectiveness of these approaches, some key performance indicators (KPIs) have been defined and used. These are needed to have an exact method of comparison between different simulations and also to compute some useful metrics. The used KPIs are listed in order of relevance:

- EV crashing into the preceding vehicle
- EV picking the best action possible
- Travel time
- Fuel consumption

Regarding the second KPI, it's manually checked after each simulation. The travel time is supposed to be minimized, by keeping the speed as close as possible to the reference speed imposed by the user, or to the speed limit of the road, depending on which one is the lowest. On the other hand, the fuel consumption is not directly extracted from the model, since it doesn't allow for this kind of data, but it's estimated based on the acceleration profile during the simulation episode.

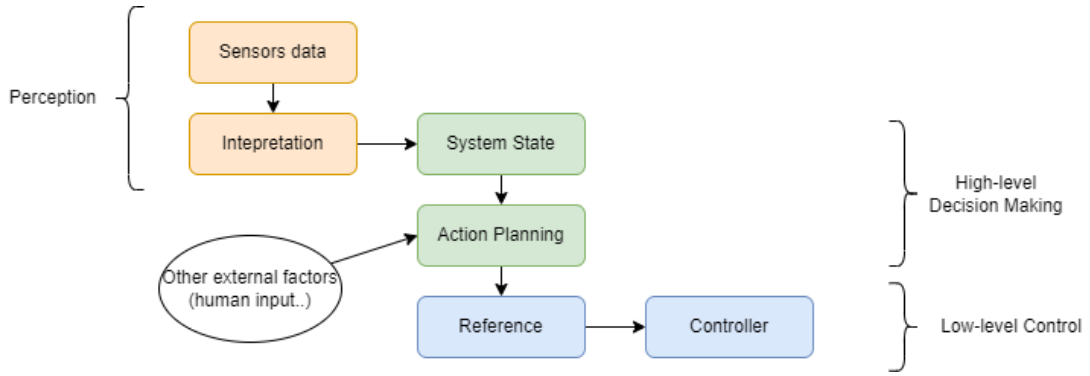


Figure 1.1: Decision making layers simplified scheme.

Chapter 2

Problem Definition

2.1 Base model

The work has been carried on over a starting SIMULINK model previously developed for a different research purpose [1]. The above-mentioned model is composed by two macro-blocks and several sub-blocks. Each of these is used to fulfill a specific part of the simulation.

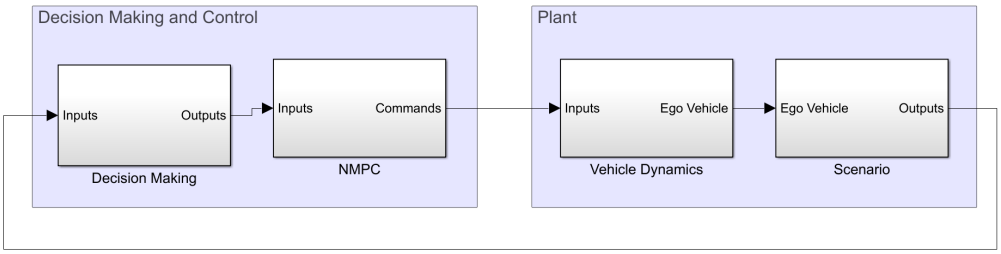


Figure 2.1: Starting SIMULINK model, used as base.

This SIMULINK model is used in combination with a MATLAB script where the needed variables and functions are initialized and computed.

2.1.1 Decision Making

The Decision Making block handles the high level control algorithms and reference generation. The inputs are collected from the environment scenario (more about this later) as a feedback bus signal, containing several variables used for the conditional logic. Among the used variables, some of the most used ones are:

- Ego Vehicle speed v_{ev}

- Ego Vehicle acceleration a_{ev}
- Ego Vehicle yaw rate $\dot{\psi}$
- Current lane
- Lane Width
- Obstacle numbers
- Distance from preceding vehicle
- Lateral offset from the center of the lane

These (and many others) are needed in order to virtualize the sensors, that are not present in this simulation model since the focus is on the decision making process. The conditional logic is handled by a state machine through a MATLAB StateFlow Chart. While powerful this allows for a human-readable format, and quick debugging if needed. Here, the action chosen is used to generate a reference for the following controller block. The standard operation of the starting model was limited to the use of predefined, user-input commands, that could be differentiated between:

- Lane Keeping
- Overtaking
- Stop
- Left Lane Change
- Right Lane Change
- Emergency Stop

Each one of these had to be manually input by the user. The "Overtaking" maneuver performs a complete sequence of *left lane change* followed by a passing phase of *lane keeping* that adapts its duration to the number of vehicles to overtake and in the end a *right lane change* to return to the starting position. Once an action is picked, a reference generator function is called to output a reference signal, used as trajectory for the controller. This trajectory takes into consideration both the action and the possible presence of an obstacle on the path.

2.1.2 NMPC

The implemented controller is using a Model Predictive Control (MPC) approach. This is an efficient common way to handle the behavior of complex models. It exploits different elements to perform the control action:

- a model of the system
- a cost function J, usually quadratic
- an optimization algorithm to minimize J] by applying a control input u

The MPC optimization algorithm can also allow for constraints definitions. The basic process involves that at each time step t the algorithm minimizes the cost function J over a certain receding time horizon $T = nt$, defined as a multiple of the time step, based on the current state of the system. The computation happens on-line, exploring different predictions. The first step of the optimal trajectory is then implemented as output u, and the process is repeated with the new sampled initial conditions.

The implemented controller is using a non-linear model of the system to generate the predictions, thus falling into the category of Non-linear Model Predictive Controllers (NMPC). While the optimization process is similar, it's worth noting that the non-linear problem becomes non-convex. Due to the progress in the numerical computational field, NMPC is increasingly adopted in several industry fields, including the automotive one.

2.1.3 Vehicle Dynamics

The output u^* of he NMPC block is then used in the Vehicle Dynamics block, where a dual track model is used to simulate the vehicle behaviour. The command from the NMPC is converted into a steering angle δ and a couple of longitudinal forces F_f and F_r, one for each axle. The vehicle dynamics block is simulating a 3-degree of freedom dual track model, using the above mentioned inputs to compute the longitudinal, lateral and yaw motion of the vehicle. While a single track (bicycle) model could be enough for this stage of development, it's useful to obtain more precise data with a dual one. The model was tuned on several parameters:

- vehicle mass
- inertia
- Tires corner stiffness
- Aerodynamic and external environmental factors (air conditions, friction coefficients)

Below the equations used for the vehicle dynamics:

$$\ddot{x} = \dot{y}r + \frac{F_{xfl} + F_{xfr} + F_{xrl} + F_{xrr} + F_{xext}}{m}$$

$$\ddot{y} = -\dot{x}r + \frac{F_{yfl} + F_{yfr} + F_{yrl} + F_{yrr} + F_{yext}}{m}$$

$$\dot{\psi} = \frac{a\left(F_{yfl} + F_{yfr}\right) - b\left(F_{yrl} + F_{yrr}\right) + \frac{w_f\left(F_{xfl} - F_{xfr}\right)}{2} + \frac{w_r(F_{xrl} - F_{xrr})}{2} + M_{zext}}{I_{zz}}$$

As a result, the info about position and motion of the ego vehicle (EV) are output to the Scenario block to complete the sequence.

2.1.4 Scenario

Finally, the computed dynamic of the EV is used as extra information to be implemented inside the Scenario Reading block, that has the purpose of simulating the behavior inside a predefined environment, where it's possible to add road boundaries and other vehicles (actors). For the use case of this work several scenarios were designed. Below a simple overtaking scenario is reported, where the EV is following a slower preceding car, both spawned on the right lane of a two-lanes road. The scenarios have been generated via the Driving Scenario Designer app implemented in MATLAB.

Finally the new road boundaries and state variables of EV and actors are fed back to the Decision Making block. The process is repeated at each simulation time step t, with $t = 0.1s$.

Figure 2.2: Overtaking scenario, bird's eye view.

Figure 2.3: Overtaking scenario, egocentric view, 3D environment simulated in Unreal Engine.

Chapter 3

Model Development

3.1 Finite State Machine Approach

A first and simple approach to automate the decision making process was carried out by means of a finite state machine. This is a computational model used in mathematics, useful to describe the transitions between states of a system, following the change of some internal or external variables. In MATLAB this method was implemented trough a Stateflow Chart (Autonomous Policy block in 3.1, inserted inside the Decision Making block).

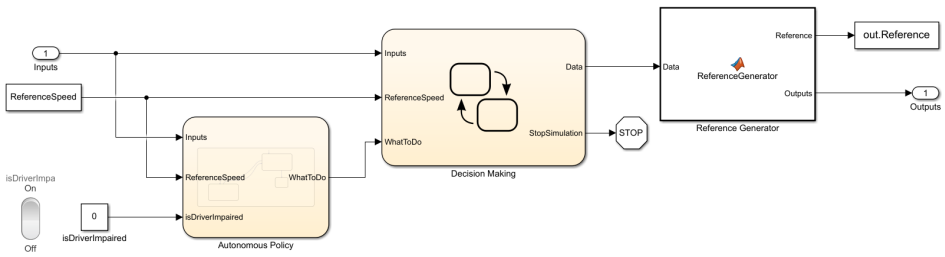


Figure 3.1: Decision Making Stateflow Chart

This block uses different inputs and data from the simulation to define the state of the EV and to determine whether or not a transition is needed by means of conditional logic. In the overtaking use case, the used inputs were defined as in table 3.1, where the last one isDriverImpaired is useful to trigger the emergency stop action, but it's manually input by the user during the simulation (if needed).

Using these variables, the states and the transitions can also be defined.

- State 0: Lane Keeping

OtherVehicleDistance	Distance [m] from preceding vehicle
ReferenceSpeed	Selected speed [m/s] to be followed
CurrentLane	Number of the lane
isDriverImpaired	Check on driver's ability to drive

Table 3.1: State machine variables.

- State 1: Overtaking
- State 2: Emergency Stop

The transitions between these states are managed by constantly checking the distance from the vehicle to overtake, the reference speed and the current lane in which the EV is moving forward.
While this approach is simple to implement, this only depends on the use case. The overtake is a simple maneuver in an ideal situation, but it can become quite complicated quite soon, when external and internal factors start being taken into considerations. For example we could add the information about a third or fourth vehicle approaching, the presence of a curve or a straight road, people on the sidewalk or crossing the street and so on. So while a fully hard-coded approach is quick and easy to implement, it's not always useful due to the lack of versatility.

3.2 MDP Framework Development

In order to develop a more complex model of autonomous decision making, Markov Decision Processes are introduced. These are used in mathematics as a clear way of define and model partly stochastic decision making by using states, actions, rewards and probabilities.

A Markov Decision Process (MDP) is defined as a tuple of 4 different elements:

$$(S, A, P_a, R_a)$$

where:

- S is the state space
- A is the action space
- P_a is the probability of transition from a state to another state, given an action a
- R_a is the reward obtained by performing a transition

A general policy π is a mapping from the state space S to the action space A. The objective, in a defined MDP framework, is to find the optimal policy π^* that also maximizes a cumulative reward function, typically the expected discounted sum of the rewards, over a potentially infinite horizon.

After defining all the possible states of the system (in our case, the EV), we also need to identify which are the actions that allow the system to go from one state to another. These actions may have a certain probability of leading to a state, thus introducing a stochastic factor in the modeling. Finally each transition from one state to another, must give back a reward, that is used by the training algorithm to understand which way to accomplish the task is desirable or not.

3.2.1 MDP State Space

The simulated scenarios where the algorithm has to optimize the policy is composed by several vehicles moving in both directions on a three lanes road (two main lanes, one emergency lane).

The lanes are numbered as follow, from right to left:

- $L0$: emergency lane

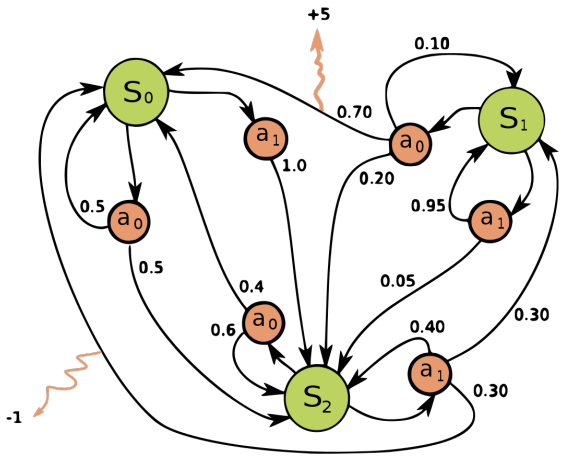


Figure 3.2: Example of three states, two actions MDP.

- $L1$: main lane
- $L2$: overtaking lane

While the involved vehicles (actors) are defined as:

- EV: ego vehicle, starts in lane 1
- PV : preceding vehicle, moving in lane 1
- OV : oncoming vehicle, moving in lane 2, but opposite direction

Before defining the states of the EV and actors, it's useful to introduce some variables:

- v_{EV} : longitudinal speed of EV
- v_PV : longitudinal speed of PV
- v_{ref} : reference speed for EV
- $v_{th} \leq v_{ref}$: threshold speed, below which it's convenient to overtake

- x_{OV} : relative distance between the OV and EV
- d_{saf} : safety distance

State S_{EV} of EV

1. $EV1$: EV in L0, car stopped
2. $EV2$: EV in L1
3. $EV3$: EV in L2

State S_{EVd} of EV's driver

1. NS : normal state, able to drive
2. IS : impaired state, unable to drive

This state is eventually used to trigger the emergency stop action.

State S_{PV} of PV

1. $PV1$: no preceding vehicle is present, lane 1 is free
2. $PV2$: preceding vehicle is present and $v_{PV} < v_{th}$
3. $PV3$: preceding vehicle is present and $v_{PV} \geq v_{th}$

State S_{OV} of OV

1. $OV1$: no oncoming vehicle is present, lane 2 is free
2. $OV2$: oncoming vehicle is present and $|x_{OV}| < d_{saf}$
3. $OV3$: oncoming vehicle is present and $|x_{OV}| \geq d_{saf}$

The total state S_T of the system is the combination of the individual states of the elements of the system, that is $S_T = (S_{EV}, S_{EVd}, S_{PV}, S_{OV})$. This would bring the total number of possible combinations to $N = card(S_T) = 3 \times 2 \times 3 \times 3 = 54$.

While a complete state definition could give a more precise overview of the model behavior, this is not useful on a computational level. Some of the states are not useful for our purpose, because they may lead to the same preferable output. For this reason it's advisable to collapse the state-space S_T to a subset S of the same, aggregating the single states, as in 3.2.

States	Definition
S11	EV traveling in L1, not possible or useful to overtake
S0	Emergency Stop in L0, terminal state
S12	EV in L1, overtake possible
S21	EV in L2, re-entry in L1 not possible
S22	EV in L2, re-entry in L1 possible
S3	EV in L1, overtake complete, terminal state

Table 3.2: State space for the overtake maneuver.

The simulation always starts in state $S11$ and the decision making system should aim at obtaining the full transition to state $S3$ using the minimum number of steps to avoid redundancy of states and to minimize the control effort.

3.2.2 MDP Action Space

While the base model allowed for several action outputs, some of those where redundant. For instance the *overtake* action was just a combination of left lane change, lane keeping and right lane change. In order to have a more meaningful notation and to simplify the training, the possible actions have been reduced to just three basic behaviors:

- llc: left lane change
- lk: lane keeping
- rlc: right lane change

thus defining the action space $A = \{llc, lk, rlc\}$.

While this is necessary to correctly define the MDP framework, it is not sufficient. In fact, in a real road environment, with two (or multiple cars) it's important to

take into account non-deterministic aspects: for example different drivers may have different driving skills or behaviors (likelihood to complete an overtake or to keep following the preceding car), a possible overtake may become risky or dangerous if a third vehicle appears in the opposite direction, the re-entry in the main lane may be impossible due to a group of preceding vehicles and so on.

In order to have a more comprehensive model but without going too much into detail to avoid over specifying the environment characteristics, some environmental probabilities have been implemented:

- $p0$: left lane change success
- $p1$: right lane change success
- $p2$: probability of a PV appearing and becoming overtakable
- $p3$: probability that the PV disappears
- $p4$: re-entering possibility
- $p5$: probability that if the EV doesn't re-enter when allowed, re-entry becomes impossible

These probabilities helps modeling the states transitions and are useful to understand how the training process changes when these are altered.

As it can be noticed, to correctly define the framework, it must hold true that $\sum_{i=1}^{N} p_i^x = 1$ where N is the number of outgoing state transitions from state x. In addition, the technological implementation of this modeling requires that a transition must exist from each state for every possible action. This does may not be true in the reality: for instance, in most of the cases it's not possible (or recommended) to perform a left lane change while the EV is proceeding in the left lane $L2$. To manage these borderline cases, the model will assume that performing those actions in those states, will reward with a negative loss and the transition will keep the state unchanged.

Inside the MATLAB environment, the transitional probabilities were defined in a compact multi-dimensional matrix form. This improves the computational efficiency while remaining relatively easy to read for non-machine users. Given a state space S and an action space A:

- $n = card(S)$ is the number of states in S
- $m = card(A)$ is the number of actions in A

from/to	S11	S0 (T)	S12	S21	S22	S3 (T)
S11	a = kl r = 0 p = 1-p2	a = rlc r = -1 p = 1	a = rlc r = 1 p = p2	a = llc r = -1 p = 1-p5	a = llc r = -1 p = p5	
S0 (T)		a = ∀ r = 0 p = 1				
S12	a = rlc r = -1 p = p4	a = rlc r = -1 p = 1	a = llc, kl r = -1, 0 p = 1-p0, 1-p4	a = llc r = 1 p = p0		
S21	a = rlc r = -1 p = 1			a = kl r = 1 p = 1-p5	a = kl r = 2 p = p5	
S22			a = rlc r = 1 p = p2	a = kl r = -2 p = p6	a = llc, kl r = -1, -1 p = 1, 1-p6	a = rlc r = 1 p = 1-p2
S3 (T)	a = ∀ r = 0 p = 1					

Table 3.3: Overview of the state transitions, rewards and probabilities. States with (T) are terminal states.

then the *transition probability matrix* $T(s, s^{'}, a)$ is a $(n \times n \times m)$ matrix where the generic entry a_{xyz} represents the transitional probability from state x to state y following action z:

$$T(s, s'|a = z) = \begin{pmatrix} p_{1,1,z} & p_{1,2,z} & \cdots & p_{1,n,z} \\ p_{2,1,z} & p_{2,2,z} & \cdots & p_{2,n,z} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n,1,z} & p_{n,2,z} & \cdots & p_{n,n,z} \end{pmatrix} \tag{3.1}$$

Following a similar approach, the *reward transition matrix* $R(s, s^{'}, a)$ is a $(n \times n \times m)$ matrix where the generic entry a_{xyz} represents the immediate reward obtained by the agent after moving from state x to state y following action z.

$$R(s, s'|a = z) = \begin{pmatrix} r_{1,1,z} & r_{1,2,z} & \cdots & r_{1,n,z} \\ r_{2,1,z} & r_{2,2,z} & \cdots & r_{2,n,z} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n,1,z} & r_{n,2,z} & \cdots & r_{n,n,z} \end{pmatrix} \tag{3.2}$$

A complete schematic of the transition matrices' structure is shown in figure 3.3.

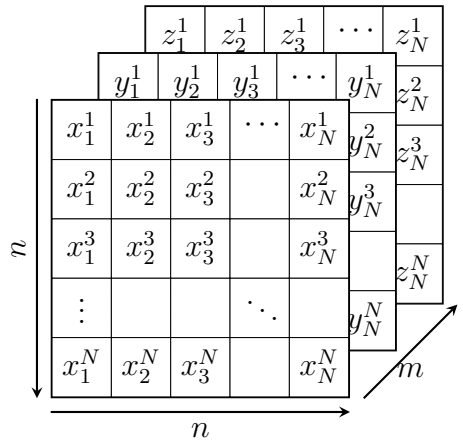


Figure 3.3: 3-d structure of T and R matrices, The dimensions are given by the cardinality of state and action spaces.

3.2.3 MDP graph

The modeling explained up to now can be visualized on a graph where the main states transitions and probabilities are highlighted, figure 3.4. To improve the readability, only the aggregated states are showed and not every transition is reported.

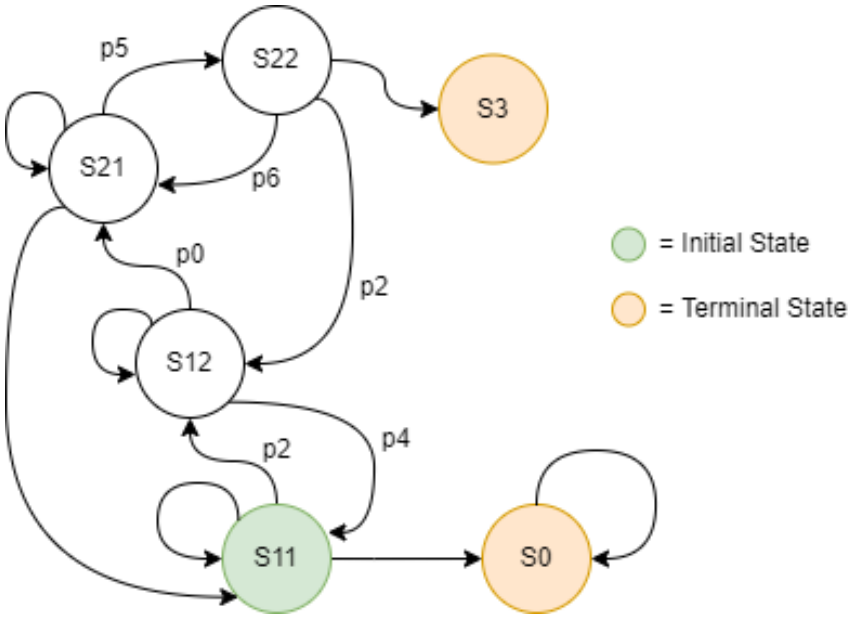


Figure 3.4: MDP states and transitions.

3.3 Reinforcement Learning Approach

Markov Decision Processes allow for a formal definition in mathematical terms of systems (in this case the environment we are dealing with). Once the states and transitions are defined, it's also important to develop an optimal policy that an automated agent can use to pick the best action to perform, given a certain state. Instead of using a manually input policy, it's a good idea to use Reinforcement Learning (RL) to pick a suitable and optimal one.

RL uses machine learnng to deal with sequential decision-making optimization problems [2]. In the RL paradigm, an intelligent agent learns a control strategy by mainly interacting with the environment. Because of each action, the agent steps from one state to another and generates and immediate reward, visible to the agent. The goal of the agent is to learn a mapping from states to actions that maximizes a cumulative function of rewards. The agent then searches for the best sequence of actions and not for decisions that are locally optimal [3].

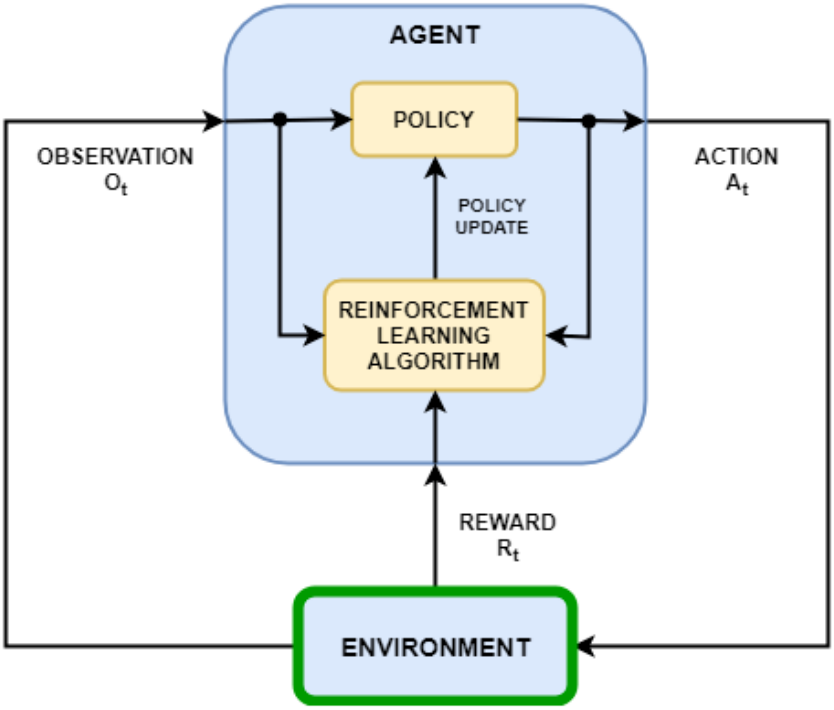


Figure 3.5: General Reinforcement Learning scheme.

A general RL framing is schematized as in figure 3.5. The main block components are:

- the *environment*
- the *agent*, which includes:
  - a *policy*, used to pick an action
  - a *RL algorithm*, used to update the policy

In our case, the environment is already defined as a Markov Decision Process, with the respective state space as observation domain and the action space as possible output of the agent. The reward is also defined by means of a n by n by m reward matrix, where n is the dimension of the state space and m the one of the action space.

The process of designing a properly trained agent consists in two phases:

1. *training phase* during which the agent can explore the environment and learn the best actions path through it
2. *testing and validation phase* during which an already trained agent is deployed inside a real or simulated environment to check performance

During the training phase, the algorithm has to tune the policy in order to maximize the cumulative reward received by the environment. This can be done either through a model-based or model-free approach. While the first is a more efficient way of proceeding, it's also more complex and requires a precise knowledge of the environment under testing. This could be possible in some cases, but not always. Besides, a model-free approach is easier to adapt to unseen states, simply by extending the training and updating the policy.

Since the model has been developed with the aim of maintaining a low computational effort throughout the design, train and testing process, the resulting choice has been to use a Q-learning algorithm.

3.4 Q-Learning algorithm definition

Among the possible ways for determining the optimal policy π^*, Dynamic Programming (DP) [4] uses the Q-learning function as an intermediate in the case where the transition probabilities and the reward function are completely known. Q-learning is a model-free method for RL, where a critic function is used to estimate the expected reward. In the case of a finite MDP, this algorithm is proven to maximize the expected reward [5] and to find an optimal policy. This section reports the general structure of the optimization algorithm used during the Q-learning training process [6] [7].

Given a discrete state space S and a discrete action space A, the agent at time step t can move in the world environment transitioning from state s_t to state s_{t+1} by picking the action a_t. Doing this, it receives an immediate reward r_t that can either be positive, negative or zero. These steps are repeated for $t+1, t+2, t+3...$ until the agent reaches a terminal state s_{ts} (defined a priori).

In order to maximize the total reward obtained by the environment over a simulation episode, the algorithm has to solve for the maximum of the *discounted expected return function* G_t:

$$G_t = r_{t+1} + \gamma r_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.3}$$

where r is the reward at each time step t, and $\gamma \in$ (0,1) is the the *discount factor* that increases the value for early rewards and decreases it for later ones.

The algorithm also uses another function Q, the so called *Q-value function* or *Action-Value function*, that tells us the quality of a certain state-action pair (s, a) when following a policy π. The policy, can be seen as a probability distribution in a given state, to pick a certain action.

$$\begin{aligned} q_\pi(s,a) &= E_\pi[G_t | S_t = s, A_t = a] \\ &= E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a] \end{aligned} \tag{3.4}$$

It is proven [6] that under the optimal policy π^* the *Bellman Optimality Equation* is satisfied:

$$q^*(s,a) = E[R_{t+1} + \gamma max_{a'} q^*(s^{'}, a^{'})] \tag{3.5}$$

The meaning of 3.5 is that for any state-action pair, the Q-value function is equal to the sum of the next immediate reward obtained and the maximum discounted expected return of any possible state-action pair. During the training, the critic is instructed to find the optimal values of the Q-value function by updating them iteratively using the Bellman equation. For not too large environment dimensions, it's possible to store the value of the critic function in a human-readable table format, called *Q-table*. The iterative algorithm to update the Q-values is defined as follows:

$$q^{new}(s,a) \longleftarrow q(s,a) + \alpha[r + \gamma max_{a'} q(s^{'}, a^{'}) - q(s,a)] \tag{3.6}$$

where:

- $q^{new}(s,a)$ is the updated value of the critic function
- $q(s,a)$ is the current value
- r is the next immediate reward
- $\alpha \in (0,1]$ is the *learning rate*
- $\gamma \in (0,1)$ is the *discount factor*

In particular, α is an important hyperparameter to tune since it appears as a weight in the equation. It largely conditions the convergence and speed of the training process.

During the training, the agent may assume two kinds of behavior, when picking the action to perform:

- exploration
- exploitation

The first, exploration, is the preferred method during the initial phase: the critic is not trained or poorly trained and it's overall better to pick a random action to perform, checking the resulting reward and end state after the transition. This is necessary to store new information and to update the q-value. After some time, a

preferable path may appear, in the form of an higher probability of best outcome for one specific action with respect to the others. In this case, the exploitation leads to the best results in terms of reward, leading the agent to use the stored experience instead of relying on trial and error. This trade-off can be formalized with the introduction of the $\epsilon \in (0,1]$ variable and the *Epsilon Greedy* algorithm:

$$a(t) = \begin{cases} \text{any} \quad a(t) & \text{with probability} \quad \epsilon \\ arg_a[max(q_t(s,a))] & \text{with probability} \quad (1-\epsilon) \end{cases} \tag{3.7}$$

By tuning the value of ϵ it's possible to promote a more explorative or cautious behavior, avoiding getting stuck in local optima. Besides, to gradually shift the exploration policy, an exponential decay factor λ can be defined, gradually decreasing the value of ϵ to an asymptotic minimum value ϵ_{min} as the number of episodes increases, following a law of the form $\epsilon = \epsilon_0 e^{-\lambda t}$.

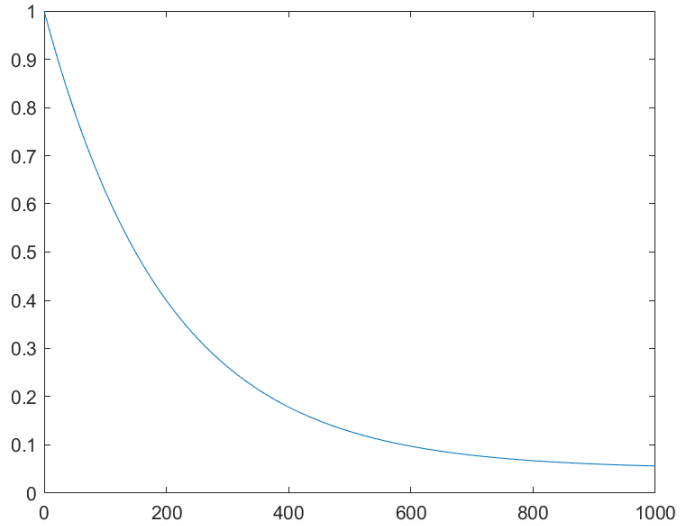


Figure 3.6: Epsilon Greedy algorithm decay over a 1000 episodes.

To summarize, the step by step process is:

1. The Q-value critic function $q(s, a, \phi)$ (where ϕ are the learnable paramaters) is initialized (at zero, or random values).
2. For each training episode:
   (a) Observe the initial state s.
   (b) Choose an action, either the best one or random, using 3.7.
   (c) Observe the received reward r and final state s'.

(d) Compute the value function target, using 3.5:

$$vft = \begin{cases} r + \gamma max_a q(s', a, \phi) & \text{if s' is not a terminal state} \\ r & \text{if s' is a terminal state} \end{cases}$$

(e) Compute the temporal difference ΔQ between the target and actual value:

$$\Delta Q = vft - q(s, a, \phi$$

(f) Update the critic function using ΔQ and α:

$$q(s, a) = q(s, a, \phi) + \alpha \cdot \Delta q$$

(g) Set the new observation state $s = s'$.

Table 3.4 shows how the Q-table is updated by storing the experience information during the training process.

Not trained			**50% Training**			**100% Training**		
0	0	0	2.88	11.77	-1.00	2.84	22.85	-0.95
0	0	0	0	0	0	0	0	0
0	0	0	11.01	3.23	-1.00	22.15	3.19	-0.98
0	0	0	0	11.33	4.48	0	22.39	4.44
0	0	0	2.30	0.33	10.80	2.26	0.33	21.52
0	0	0	0	0	0	0	0	0

Table 3.4: Example of Q-table updated by iterative optimization algorithm.

3.5 Training results

The training has been performed in MATLAB on a mid-to-high level CPU. The training algorithm requires some additional parameters, that are reported in 3.5. These parameters have been tuned by trial and error procedure to optimize the convergence process of the learning with a trade-off between speed and accuracy.

Agent Options	
Learn Rate	0.1
Discount Factor	1
Epsilon	0.9
Epsilon Decay	0.01
Training Options	
Max Episodes	1000
Stop Training Criteria	Average Reward

Table 3.5: Options for training process of the agent.

An example of training session is shown in ref . The blue lines represent the episode reward (light blue) and the rolling average over the last 30 episodes (dark blue). In this case the agent learns the optimal policy in about 120 episodes.

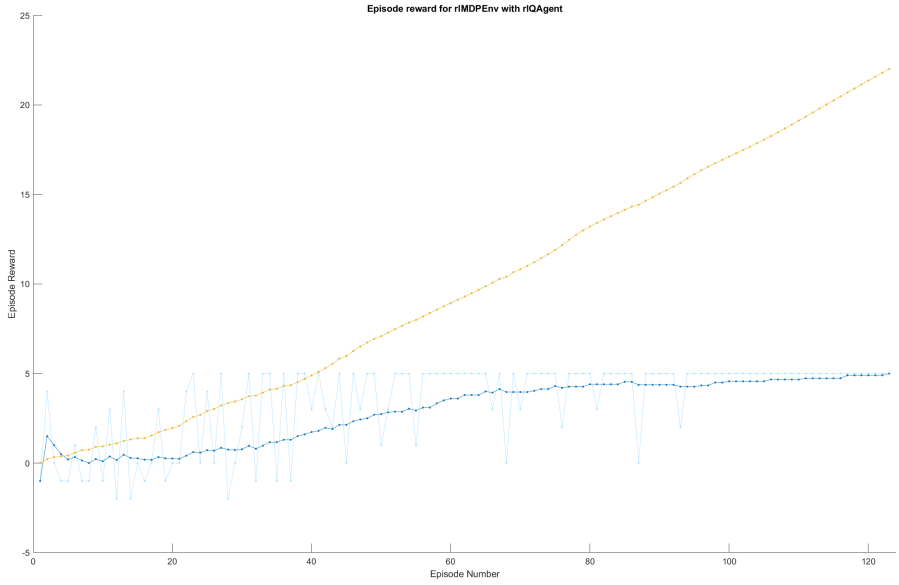


Figure 3.7: Training session, the agent learns the optimal policy by reaching the global maximum reward.

It is important to consider the effect of the environment on the training. The

example shown in 3.7 is using a simplified model of the MDP environment where the transition probabilities are either set to 0 or 1. This indeed speeds up the process but it's not close to a real case. When tuning down the probabilities (for example allowing for cases of non-perfect overtake maneuvers, presence of external factors etc.), effectively adding the stochastic factor, the graph may not converge anymore to the same values or it may not converge at all. In this case it's useful to rethink the reward function and the training criteria. An example of non convergent training session is shown in 3.8.

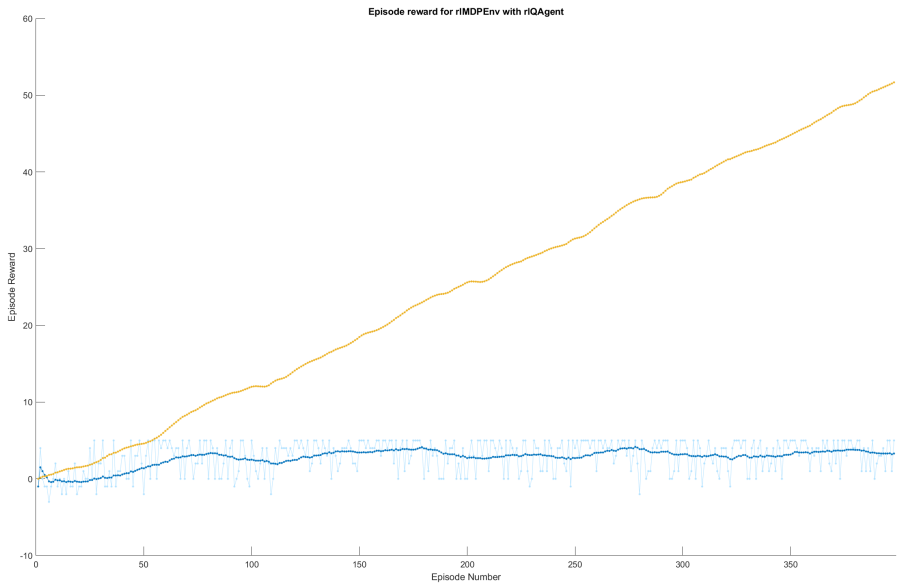


Figure 3.8: Training session with non-convergent behavior.

3.6 Considerations about an MDP-RL approach

Markov Decision Processes are only one of the several ways in which a problem like this could be tackled. There are several reasons why this method is preferable, with respect to a more complex one. By defining the environment as a framework of states and transitions, we're effectively shifting from a model defined in the continuous domain, to a discrete one. This decoupling is extremely useful to simplify the process of decision making but it may also have some drawbacks to take into account while dealing with it.

First of all, the state definition must be complete, in the sense that every possible situation should be included as main state or aggregate state, to avoid incurring in unexpected behaviors. This is of course hard to predict a priori, but it can be improved by time through simulation, specifically by simulating extreme or uncommon cases or by using customized explorative policy functions. A good approach could be using a nested structure, with macro-states containing more specific situation-based states. The same reasoning could be done for the action space, following a similar logic. This initial process is facilitated by the use of a model-free algorithm that can more easily adapted in later stages of development.

As a second factor, the transitional probabilities are defined but not fixed. This means that it's possible to understand how the model changes when a certain state transition is made more or less probable. While of course it's not useful to simulate an unrealistic environment, it may be useful to check the immediate and cumulative reward change when a certain case changes its appearance frequency. This could allow the manufacturer for a more precise and reliable risk assessment, already in the training phase of the process.

Although not covered in this paper, it is important to remember that the reliability of an MDP model is also strictly related to the accuracy of the sensors data, figure 3.9. If the external inputs (may they be camera frames, LiDAR, Radar or others) are not correctly interpreted by the sensor acquisition systems, this could lead to a bad state definition and all the related consequences. To prevent this it's essential to correctly define the number and type of sensors needed for the specific application and supervise the data acquisition process to ensure there are no identification errors along the way.

On the other hand, the main point towards the use of an MDP state approach is probably the low requirement in computational effort. Since the work is carried on in a decoupled discrete domain, there is a remarkable reduction in training time and usage of the processor units. The tests conducted for the writing of this

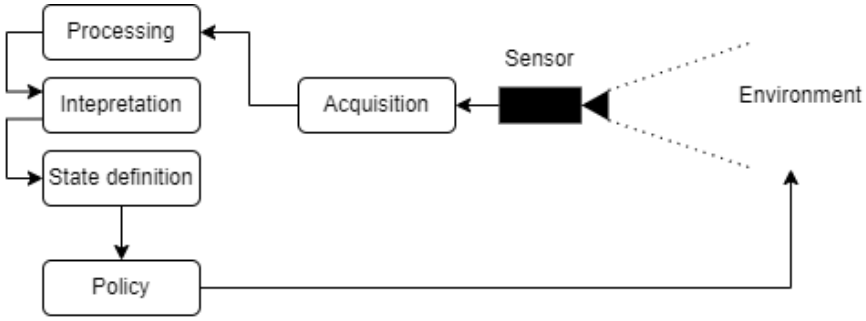


Figure 3.9: Sensors acquisition and state definition process.

paper were performed on a general-use machine with average computational power, with no particular negative effects to be noticed. This element can be of main importance because it can allow the use of the MDP framework not only for cars but also other kinds of low-power machines and devices (for instance drones, rovers or other kinds of self-driving automated systems) where there may be constraints related to production costs, efficiency, heat dissipation, or weight (due to bigger battery packs needed for more complex CPU architectures).

3.7 Deterministic vs Stochastic approach

The system proposed in the previous chapters, is exploiting a simple but in some sense limited deterministic approach. In a real human-driven scenario, each choice is based not only on rational factors but also on some other elements that have an important weight in the decision making process. It's easy to imagine, for instance, how a human driver could approach in extremely different ways the same scenario, at different times.

While it's intuitive to think that it's not possible to predict the outcome of an action where a human is involved, it's questionable if this is the result of real stochastic process. In fact, this could just be related to the small quantity of informations about the system that the observer could have. Imagining to try to predict the answer to the question: "Will a human driver try to overtake the next vehicle?", this could be a much easier or harder task to complete based on the amount of knowledge we have about the related subject. For instance, factors like:

- is the subject on time or late with respect to the scheduled arrival time?
- is the subject a skilled driver?
- is the vehicle powerful enough to allow for a safe and quick overtake?

could be fundamental to make an optimal prediction and at the same time they contribute to reduce greatly the stochastic aspect of the system, bringing it much closer to a deterministic function. When trying to replicate or substitute that same human driver, the consequences need to be taken into account. AI methods analyze a certain number of variables to understand the state of the system and then use an optimal policy (that can either be static and previously trained or dynamic and subject to constant updating) to pick the best output. While it's possible to largely increase the dimension of the input space to better characterize the status of the system (for instance, using as input advanced sensors data like mood-detection cameras or neural interfaces) or, on the other hand, to use neural networks that can simulate random elements, using stochastic weights or transfer functions [8], we're still far away from matching the complexity of a human-level decision process. Even if, in an hypothetical scenario, we could do it, it's still an open question to understand whether it is convenient to do this.

An unpredictable artificial intelligence may not be safe or efficient in some cases. The issue that arises is already of central importance in the autonomous driving field, bringing up ethical and moral aspects. One of the main concerns in applying artificial intelligence to cars is of course related to the safety of driver and passengers, but also how the car should behave in the world environment under critical situations. The main difference that puts AI applied to normal cars, on a whole other level of attention with respect to other kinds of machines is that the environment in which the AV is moving, is completely unregulated, unpredictable and strictly tied to the human sphere. This element is non-existent or greatly reduced in other types of environment (for instance an industrial robotic arm, or a space exploration rover that operates in human-free or human-restricted settings)

During the development of a deterministic policy then, manufacturers should choose how to act in certain situations, trying to take into account every possible variation that could cause a system fault. This is a hard task (if not impossible) to tackle and while the number of car crashes is supposed to be reduced in the near future, it's still not clear by how much and by when [9].

On the other side, when a critical situation occurs, the limits of a deterministic policy appear. A 2016 study [10] showed the difference in public acceptance of different AI policy applied to AVs. In an extreme situation, where the car must decide either to put in danger the passengers or other people, the public opinion accepted and approved the existence of "ethical AIs" that try to minimize the overall damage (for example limiting the loss of life, at the cost of also endangering the passengers) but was also less incline to buy such a model of car, indirectly preferring a policy that emphasizes the passengers protection.

Chapter 4

Results

4.1 Simulation and Testing

The trained agent has been tested inside a simulated MATLAB-SIMULINK environment. Different scenarios of increasing complexity were used to test the stability of the control algorithm and the reliability of the trained policy function. While it's possible to manually script the scenario, they were created inside the Driving Scenario Designer app that offers an useful graphical interface to setup road geometry and boundaries, ego vehicle's initial position and speed, and additional actors if needed.

The used scenarios are increasingly complex variations of an overtake maneuver, where the agent has to pick the best outcome taking into consideration several external factors. In order of complexity:

1. Simple overtake maneuver, minimizes travel time.
2. Overtake/car following maneuver based on a convenience policy, trade-off between travel time and fuel consumption.
3. Real-life scenario, several cars proceed in both direction, policy must consider additional safety measures to avoid crashing.

4.1.1 Use case 1: simple overtake maneuver

Some key frames were extracted from the simulation and are reported in the next pages in figure 4.1. The MDP state and action output were recorded during the simulation, to check the accuracy of the state tracking and the policy enforcement by the agent, figure 4.2. Finally some parameters like velocity and acceleration are used to compare different cases and to compute KPIs, figure 4.3 and 4.4.

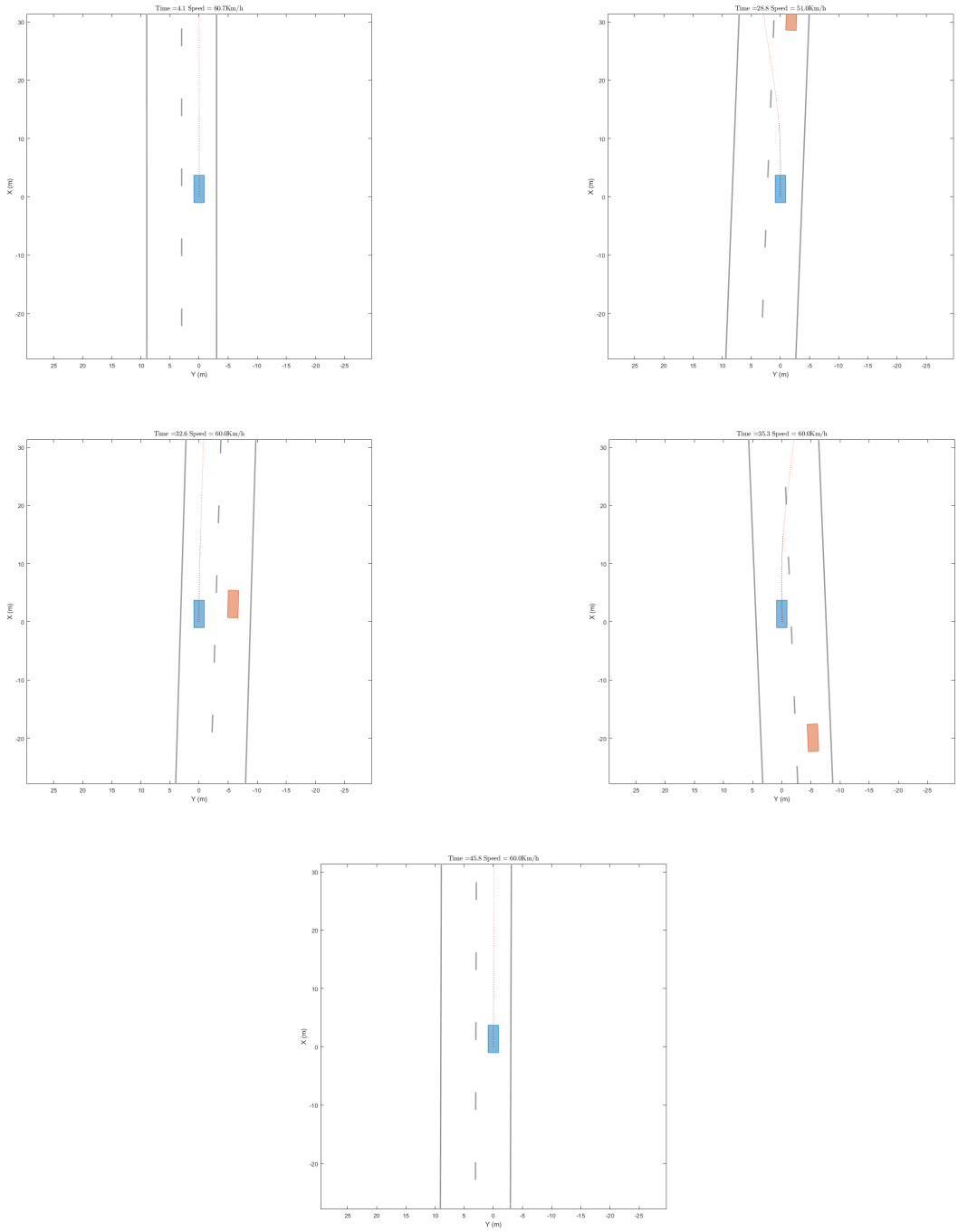


Figure 4.1: Simple overtake maneuver. When the EV approaches the PV, the agent transitions from state S11 to state S12. A reference is generated for the controller and the overtake maneuver starts. From state S21, the re-entry is allowed and the controller performs the right lane change.

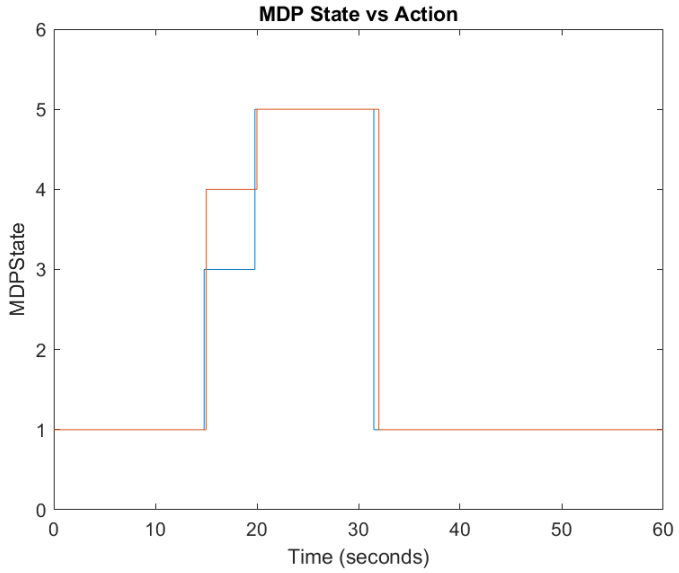


Figure 4.2: MDP state and action output. The correct action is performed after a short delay from state recognition.

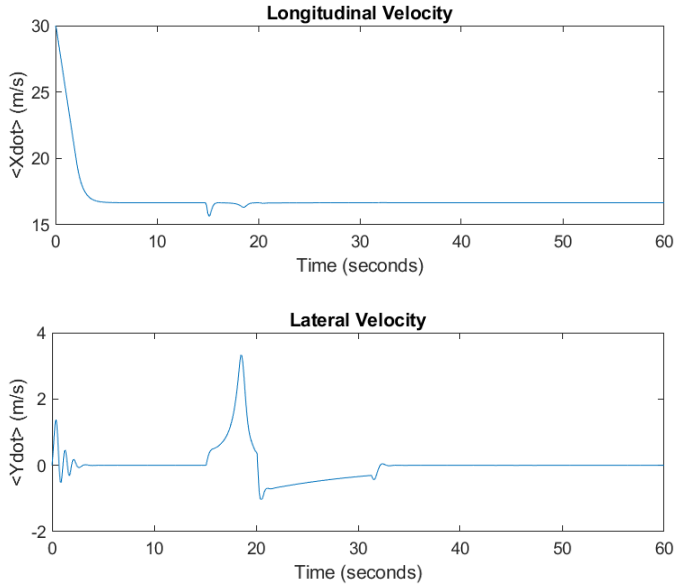


Figure 4.3: Longitudinal and lateral velocity profiles. The overtake is clearly visible. The initial conditions of the car determine the transient.

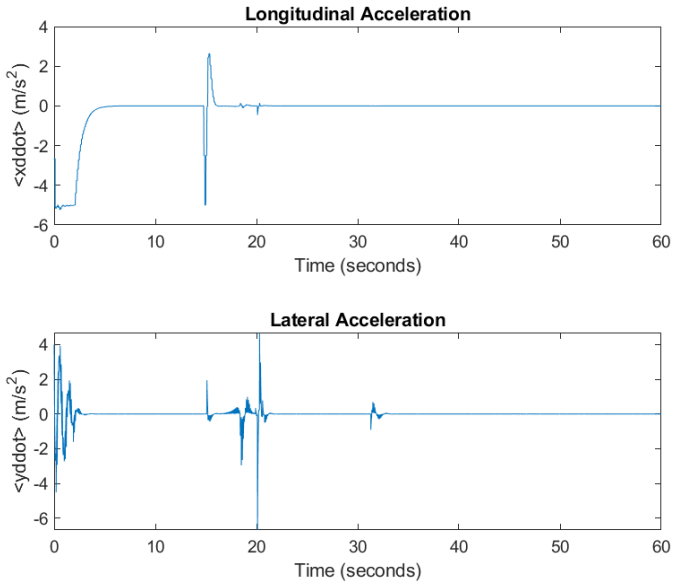


Figure 4.4: Longitudinal and lateral acceleration profiles.

4.1.2 Use case 2: Overtake/Car following

This use case involves a preceding vehicle, but the agent is instructed to choose a more convenient policy, either an overtake action or a car following action, based on the speed of the actor. The threshold has been set to $v_{th} = 0.7v_{ref}$ of the reference speed. The car following case is reported below, figure 4.5.

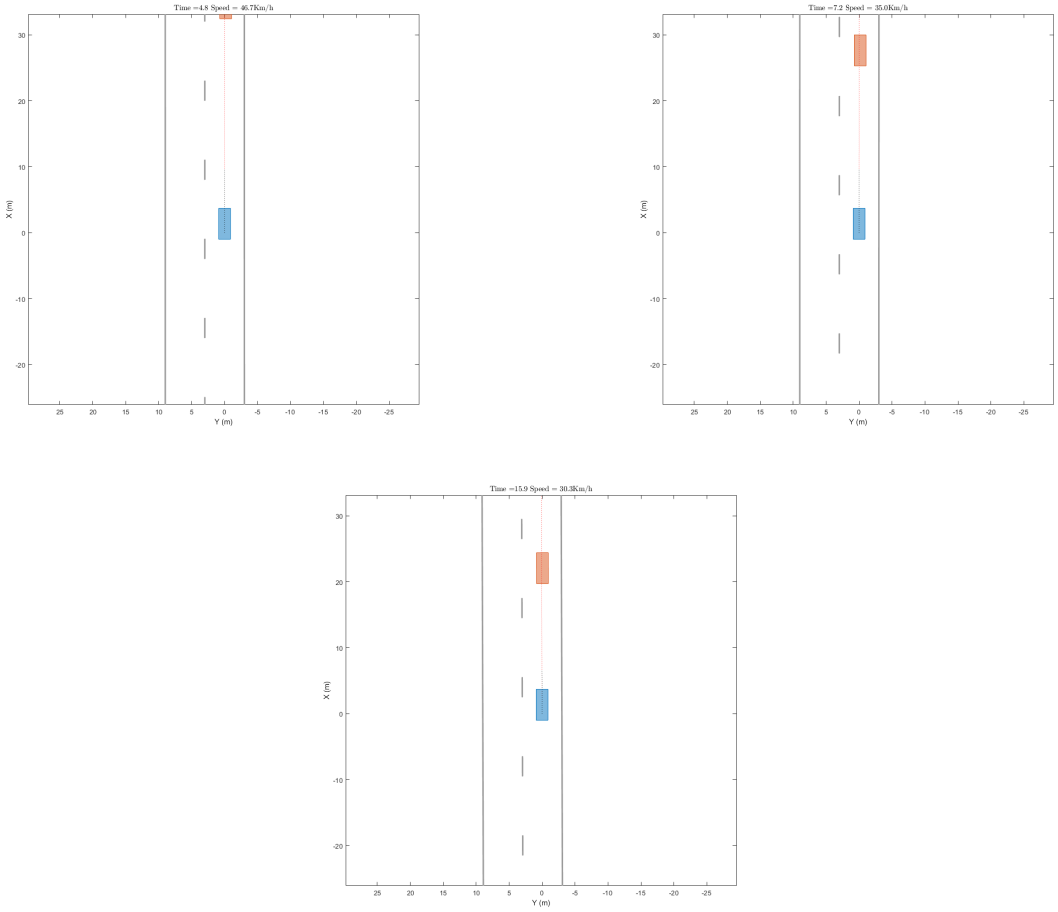


Figure 4.5: Car following maneuver. The overtake is not recommended due to the PV having an acceptable speed.

The EV approaches the preceding car and switch to Adaptive Cruise Control (ACC) mode, the speed is higher than the policy threshold so there is no state transition. The check on the preceding speed is computed during the switch from Cruise Control (CC) to ACC. As a result, the MDP state and action do not change during the simulation. They are reported in figure 4.6 (MDP state and action overlap). The dual case, when the preceding vehicle is slower and $v_th \geq 0.7v_ref$ is

not reported since it gives similar results to 4.1.1.

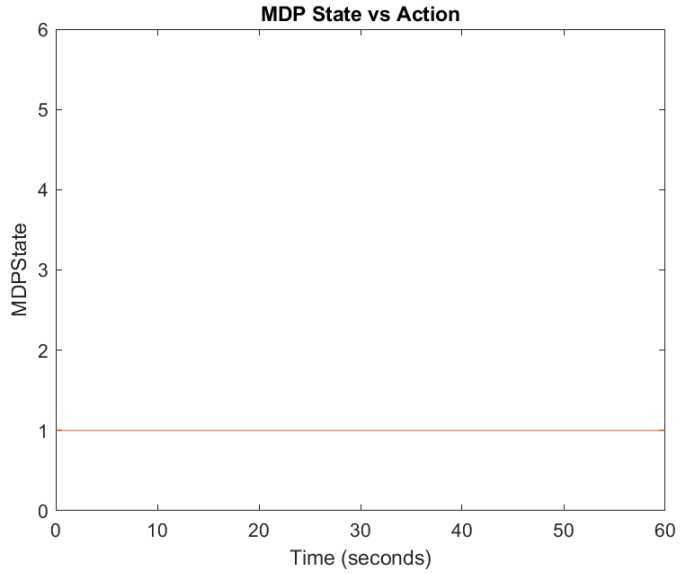


Figure 4.6: MDP state and action output. No action is performed and the system remains in state S11.

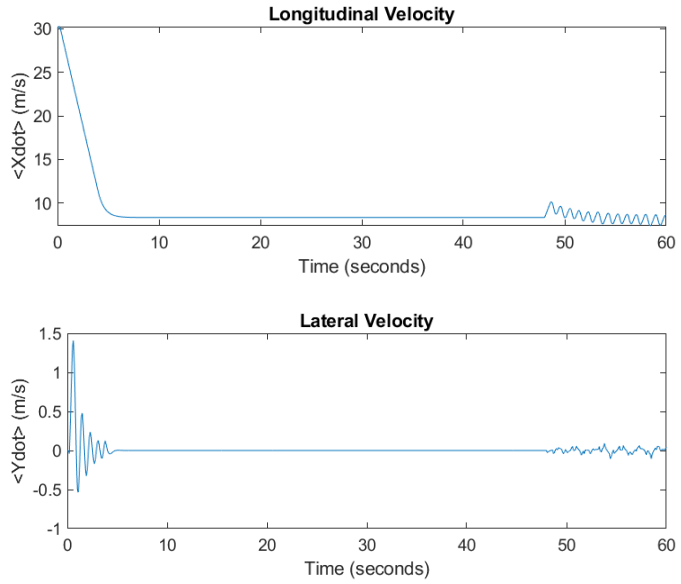
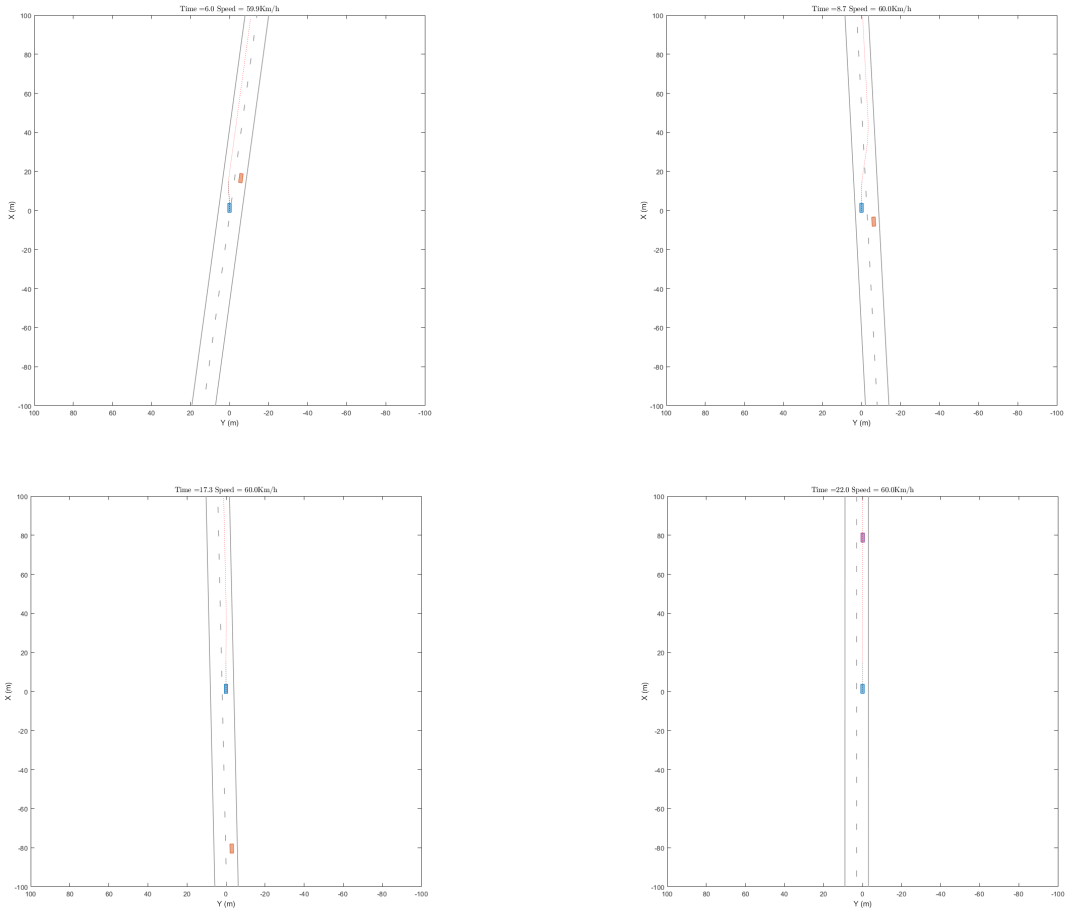


Figure 4.7: Longitudinal and lateral velocity profiles. The initial transient depends on the initial conditions.

4.1.3 Use case 3: Overtakes sequence scenario

Finally, a more complex scenario was used to test the full functionality and performance of the policy. This scenario involves a sequence of two overtakes, where the second one requires a more strict safety procedure due to a third vehicle approaching the group in the opposite direction. The agent is then required to understand and act accordingly, checking for safety distance before attempting to complete the maneuver. The safety distance is computed according to two parameters d_1 and d_2 that represents the front and rear distance with respect to the EV inside which the left lane is required to be clear of incoming vehicles. The simulations is showed below, figure 4.8. The MDP state diagram confirms the state transitions when encountering the different vehicles and the correct action output as a consequence, 4.9. Since the states are aggregated, the state in which the EV can't perform the overtake due to the presence of an incoming vehicle is included in state $S11$, defined as *overtake not possible or not useful*.



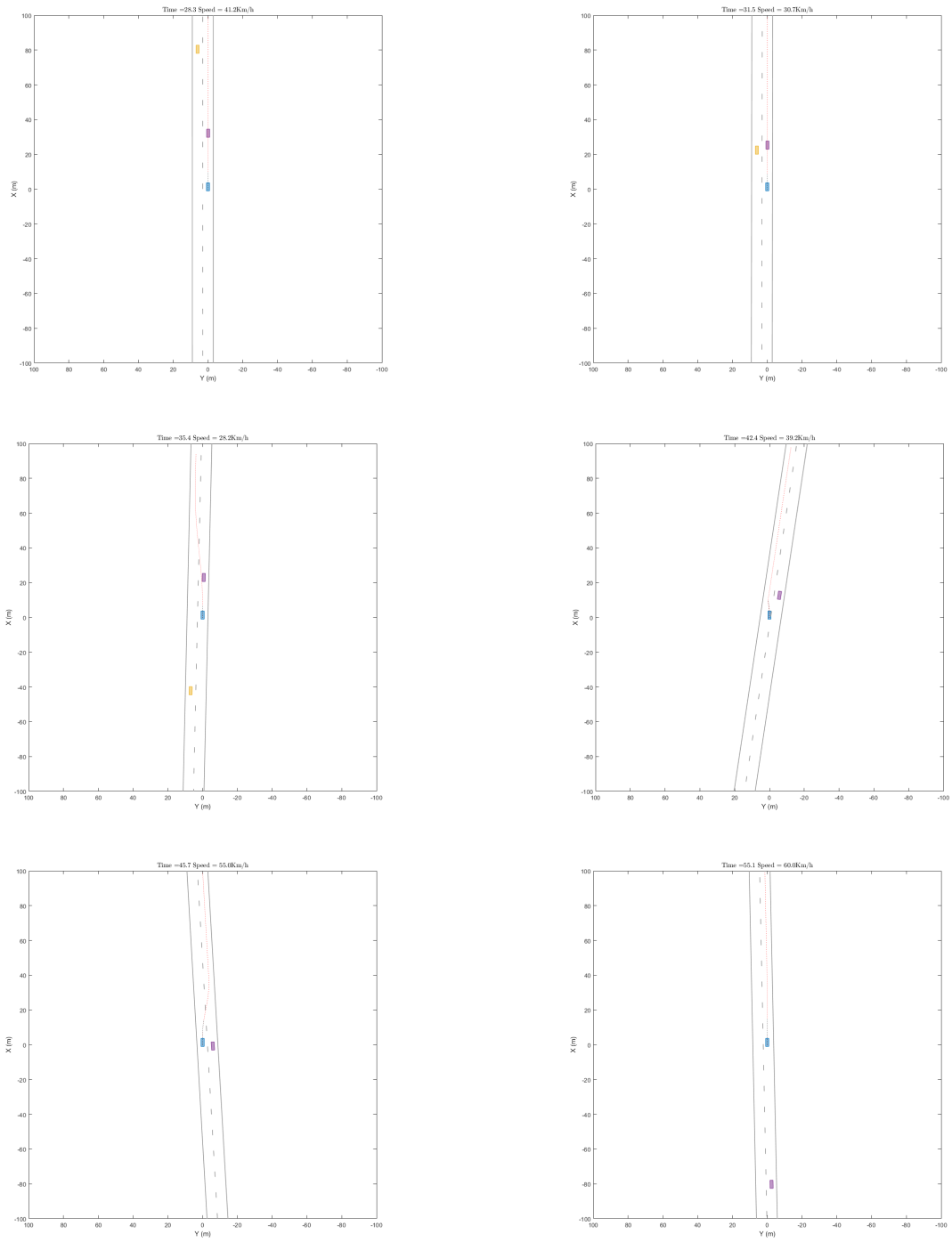


Figure 4.8: Overtakes sequence scenario. Three actors are involved. The second overtake is delayed due to the presence of an OV.

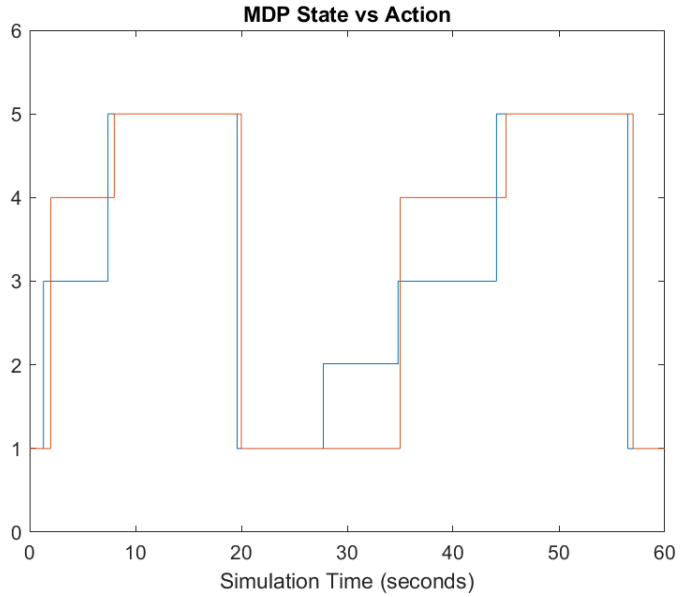


Figure 4.9: MDP state and action output. To improve comprehensibility, the state S11 has been split into two sub-states. At around $t = 28s$ it's possible to see how the EV recognizes the OV and avoid starting a maneuver until the safety distance parameter is met.

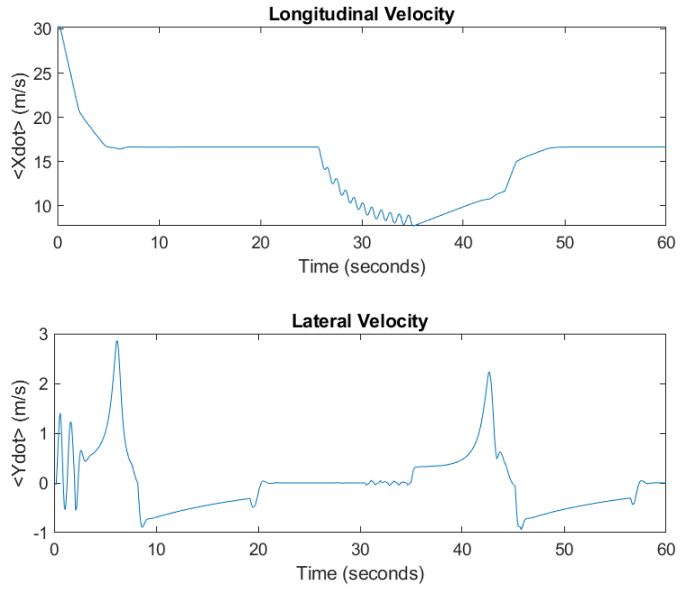


Figure 4.10: Longitudinal and lateral velocity.

4.2 Validation

After testing, in order to validate the results, the system was simulated again under randomized initial condition. This assures that the agent hasn't been locally optimized for a specific scenario. The policy must perform in a similar way under close but not equal initial conditions. A randomizing MATLAB script was used to obtain this, varying the position and velocity of EV and actors at each iteration. The simulation has then been performed again on this scenario dataset, for about 50 total episodes for each use cases. The agent reported a 100% completion rate of the dataset, without any error or unpredictable behaviors.

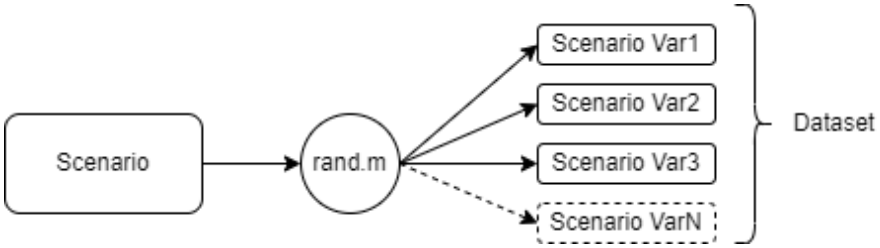


Figure 4.11: Random scenarios dataset generation process.

4.3 Further Improvements

As a first step forward, the presented model could be validated on real-world data, possibly extracted from an actual vehicle during a cycle of specific controlled maneuvers. While it's possible to implement it in a controlled environment, the cases handled in this paper are specific and simple ones, if compared to the real use of a car in an urban environment.

To allow for a more extensive implementation of a similar AI, some improvements would be needed. The main issue of this approach is related to the poor scalability of the algorithm when the number of states considerably increases. When a high number of states is present, it's not ensured that the agent will explore the entirety of the state space and could settle on local optima. While this problem could be overcome with a fine tuned exploration policy, the efficiency of the algorithm could decrease. On the other hand, also the Q-table policy function approach could become not as useful, making other less human-readable methods more convenient.

A potential solution could be to use different highly specialized agents, each one trained to handle a subset S' of the state space S, for instance a single maneuver or environmental situation. These could also extend their operability in a vertical

top-down way, for example using an agent to handle a general macro-decision block and several other on a lower level for specific actions.

If the constraints on the computational resources are more relaxed, another way of development could involve an hybrid solution, where discrete and continuous policies cooperate on different decision making levels. This could let more versatile neural networks to be used, with greater multidimensional input spaces.

Chapter 5

Conclusion

The presented model was proven of being able to handle scenarios from basic to intermediate complexity, while keeping low computational requirements. This could serve as a starting point for a more extensive study of the field and application to real systems.

While the clear, human readable structure is a another major plus point, it's important to also consider the drawbacks. In fact, it may not suit applications where the system is dealing with very large state and action space, due to the uncertainty of full exploration during the training and increasing complexity of the matrix form reward function.

In this case it's advisable to attempt an hybrid approach that either uses a discrete AI as the backbone of the decision making system, and then complement it with a traditional neural network (NN) on a lower automation level, or on the other hand, one that exploits the Markov Decision Process states as one of the observation variables of the agent, to speed up the training process.

Bibliography

[1] A. Castellano, M. Karimshoushtari, C. Novara, and F. Tango. «A Supervisor gent-Based on the Markovian Decision Process Framework to Optimize the Behavior of a Highly Automated System». In: *Augmented Cognition*. Ed. by Dylan D. Schmorrow and Cali M. Fidopiastis. Cham: Springer International Publishing, 2021, pp. 351–368. ISBN: 978-3-030-78114-9 (cit. on p. 3).

[2] Bellman R. *Dynamic Programming, 6th ed.* Dover Publications, 1957 (cit. on p. 18).

[3] Barto A. G. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning), 3rd ed.* The MIT press, 1998 (cit. on p. 18).

[4] Olivier Pietquin and Fabio Tango. «A reinforcement learning approach to optimize the longitudinal behavior of a partial autonomous driving assistance system». In: *ECAI 2012*. IOS Press, 2012, pp. 987–992 (cit. on p. 20).

[5] Francisco S Melo. «Convergence of Q-learning: A simple proof». In: *Institute Of Systems and Robotics, Tech. Rep* (2001), pp. 1–4 (cit. on p. 20).

[6] Peter Dayan and CJCH Watkins. «Q-learning». In: *Machine learning* 8.3 (1992), pp. 279–292 (cit. on p. 20).

[7] Wikipedia contributors. *Q-learning — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-November-2022]. 2022. URL: `https://en.wikipedia.org/w/index.php?title=Q-learning&oldid=1120641991` (cit. on p. 20).

[8] Claudio Turchetti. *Stochastic models of Neural Networks*. IOS Press, 2004 (cit. on p. 28).

[9] Insurance Institute for Highway Safety. *Self-driving vehicles could struggle to eliminate most crashes*. 2020. URL: `https://www.iihs.org/news/detail/self-driving-vehicles-could-struggle-to-eliminate-most-crashes` (cit. on p. 28).

[10] Jean-François Bonnefon, Azim Shariff, and Iyad Rahwan. «The social dilemma of autonomous vehicles». In: *Science* 352.6293 (2016), pp. 1573–1576. DOI: 10.1126/science.aaf2654. eprint: https://www.science.org/doi/pdf/10.1126/science.aaf2654. URL: https://www.science.org/doi/abs/10.1126/science.aaf2654 (cit. on p. 28).