Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Hybrid Electric Vehicle energy management control based on stochastic dynamic programming

Master Degree Candidate
Enrico Mion

Advisor
Prof. Maria Elena Valcher

Co-advisor
Prof. Simos Evangelou

Master Degree in
Automation Engineering
2017

To my parents Beatrice and Maurizio
and my sister Veronica.

Contents

1 Introduction - State of the art **7**

2 Markov Chain theory for speed prediction **11**

2.1 Markov Chain theory: main concepts and predicting purposes 11
2.2 Multivariate Markov Chain 14
2.2.1 Model Parameters Estimation 15
2.2.2 Linear Programming 16
2.3 Data Analysis and prediction procedure 17
2.3.1 Min-Max Mapping 17
2.3.2 Data Illustration and Mapping 17
2.3.3 Transition Matrices 20
2.4 Algorithms for prediction of speed: Markov Chain Method 21
2.5 Tree Generation Procedure 23
2.5.1 Tree Generation Procedure for Multiple sequences 24
2.6 Simulations and Results 26
2.6.1 Indian Drivecycle, Markov chain procedure 26
2.6.2 Indian Drive cycle, Multivariate Markov Chain procedure 28
2.6.3 WLTP Drivecycle, Markov chain procedure 31
2.6.4 WLTP Drivecycle, Multivariate Markov chain procedure 33
2.7 Final observations on Markov Chains Methods 35

3 HEV Model **41**

3.1 Parallel Architecture 41
3.2 Vehicle 41
3.3 Transmission and GearBox 42
3.4 Internal Combustion Engine (ICE) 43
3.5 Motor and Battery Branch 44

4 Dynamic Programming **47**

4.1 Dynamic Programming algorithm 47
4.2 DPM Matlab function 49
4.3 Application of the procedure with given drive cycles 49
4.4 Real Time Algorithm and Stochastic Dynamic Programming 55
4.4.1 Cost Function update 58
4.4.2 Real Time Optimal Control Strategy 60
4.5 Simulations with real time strategy 60
4.6 Speed-Dependent Alpha 64

5 Conclusions **71**

A Operating modes for the hybrid electric vehicle **73**

B Dynamic Programming **75**
B.1 Boundary Line Method 75

C Regression and Regularization Term **79**

References **80**

Abstract

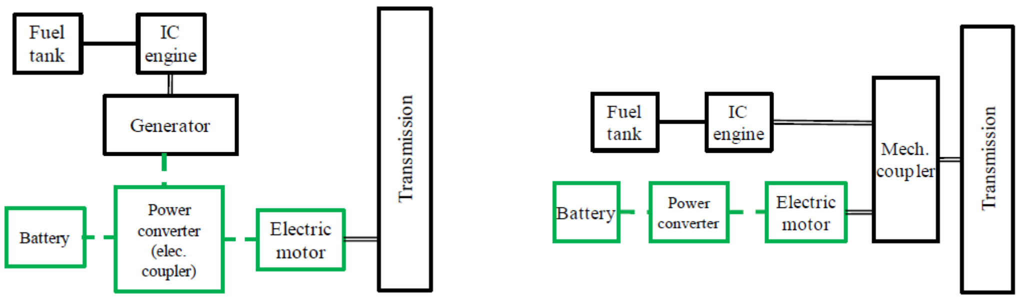
The aim of the project is to develop a real time optimal control strategy which allows to optimize fuel consumption of a hybrid electric vehicle, in particular a scooter (see Figure 1), characterized by a parallel architecture.
The first part of the project is dedicated to find a method to estimate the future vehicle speed. The latter undergoes a quantization procedure and is modelled as a stochastic process; then its future value is predicted by exploiting the main characteristics of Markov Chain theory, considering only the current value of velocity and the transition matrix, computed by using past data, that stores the probability distribution associated to each single speed value. This procedure has been improved by exploiting Multivariate Markov Chain Theory, which allows to predict the next state of speed by using two different data sequences, as for example, the speed and the combination of throttle and brake, or the speed and power profiles.

Figure 1: Picture of a scooter

A dynamic programming algorithm is then exploited to compute the optimal control policy for power energy management, while the vehicle follows the predicted speed in the presence of constraints in battery usage. This procedure chooses the optimal torque split value, which determines how to provide the required power for the current journey. At the beginning, the whole driving cycle and the horizon are known in advance, therefore the procedure works off-line.
The dynamic programming scheme is further reformulated in a stochastic way so that the above speed prediction and subsequent optimization are performed on a short window and then repeated until the end of the vehicle journey in a receding horizon manner. This reformulation introduces further complications in terms of the battery state-of-charge (SOC) range of usage which is addressed by setting up a modified cost function that takes into account both fuel consumption and SOC.

1 Introduction - State of the art

Global warming and climate change are playing an important role in the design and the development of new powertrain architectures in the automotive world. In fact, Hybrid Electric Vehicles demand is rapidly increasing in the global market, due to their ability to save fossil fuels and to exhaust pollutant emissions, without worsening the performances, but on the contrary increasing them. A hybrid vehicle has two or more major sources of propulsion power, such as an internal combustion engine plus an electric motor, ruled by an energy management controller which determines the amount of power to be delivered by each energy source of the vehicle. There are several different powertrain configurations; the most famous are the series and the parallel architectures. We can see two of the schemes in Figure 1.1.



(a) Series **(b)** Parallel

Figure 1.1: Series and Parallel Architectures

According to [1], [2], a series architecture is mainly characterized by three branches: the spark ignition engine, the battery and the electric motor connected with the transmission. Power supply to drive the vehicle is mainly provided by the electric motor within battery state-of-charge constraints; on the contrary, the engine branch works as a generator to recharge the battery or to supply power to the vehicle together with the battery.

This project will instead consider the parallel architecture described in [3]. Here the battery and the engine are mechanically linked through the transmission to propel the vehicle: in fact they both work in tandem to generate the power that drives the wheels. In our project we will exploit this architecture and the model provided, characterized by only one state, the state of charge, and one input, the torque split. This model is similar and simpler with respect to the model of the scooter to which we want to apply our real time strategy.

As we said before, generally the main goal of the optimization task for a hybrid electric vehicle is to reduce fuel consumption and pollutant emissions. Nevertheless, there might be other interests in applying an optimization strategy, for example paper [1] describes a Sports Series Hybrid Vehicle for which the main goal is minimizing the lap time in a particular circuit. For this purpose, the essential feature of the series architecture is that there is no mechanical connection between the engine and the transmissions, so the engine may provide the maximum available power without considering the vehicle speed. The algorithm applied here is based on an indirect optimal control approach which may optimize both powertrain energy and the trajectory with speed profile of a racing HEV inside a given race circuit. More specifically, the goal of this paper consists in computing the optimal control inputs that increase the performances of the vehicle in terms of time T necessary for the vehicle to drive on a given track. Here, differently from the simple model described in [3], the vehicle model has a complex structure characterized by 7 states: the battery charge, the longitudinal and lateral speeds, the yaw rate, the longitudinal and lateral positions on the road strip and the vehicle heading, relative to the road. Also, it has 4 inputs: the battery, generator and brake power and the steering angle. The latter model is characterized by a higher accuracy with respect to the other model, nevertheless it is not suitable for real time purposes.

Paper [2] exploits again an indirect optimal control approach to attain the green driving optimization: the optimization task computes the power sharing among powertrain sources and the vehicle speed profile to achieve fuel consumption minimization for a given vehicle journey, keeping into account comfort requirements and safe driver behaviour.
Finally, the optimization problem investigated in both previous papers is formulated in terms of a cost function which represents the performance criterion we want to achieve (for example fuel consumption), together with a set of constraints to satisfy. The same structure will be adopted in this project, but with a different solution method: indirect optimal control is going to be replaced by dynamic programming, a powerful tool which solves Bellman's equation and can easily handle the given constraints and the non linearity of the problem while obtaining a globally optimal solution.

Dynamic programming is also exploited in paper [4], where a deterministic optimization problem is designed for a parallel architecture of a hybrid truck, because *the results have the clear advantage of being near-optimal, accommodating multiple objectives, and systematic. Depending on the overall objective, one can easily develop power management laws that emphasize fuel economy, and/or emissions.* The target of the project is to minimize a cost function obtained as a combination of the fuel consumption with the pollutant emissions over a given journey with a finite horizon. The optimal control actions, subjected to constraints on the battery state of charge, are represented by the gear-shifting strategy and by the power split between the battery and engine branches. The result is a trade off between fuel economy and emissions. This paper also introduces a penalty factor in the cost function for the terminal state of charge, which penalizes the tendency of depleting the battery. All the computations are performed off-line with a given driving cycle, known in advance. A dynamic programming procedure here manages to compute a global minimum for the cost function while bringing the final state of charge close to the desired final value. Finally, since the gear-shifting strategy is considered an input to optimize, the procedure has to deal with frequent shifts, which are not really desirable for the driveability of the vehicle. So another penalty term is introduced in the cost function to penalize the input configurations which impose a gear shift.

Our main interest is in realizing an optimal control strategy which can work in real time, where the final time or constraints are not defined and the known horizon is very limited. In this sense [5] formulates an infinite-horizon stochastic dynamic optimization problem, where Markov Chain theory is exploited to model driver power demand as a random process. The future driver power request under diverse driving conditions is uncertain, so this procedure allows to estimate future values of power demand. The control strategy provides again power management, reached through the design of an algorithm that determines the optimal way to split the power between the electric motor and the engine. The main goal is still to minimize fuel consumption and emissions, while satisfying constraints such as drive ability, charge sustainment and component reliability; the generated control policy is time-invariant and thus can be easily implemented in real time.

According to our purposes, we must estimate the future speed of the vehicle in order to realize an optimal real time control strategy; Markov Chain theory can be used to model not only power, but also speed or acceleration demand, as explained in [6] and [7]. Paper [8] proposes a different prediction strategy to model power demand based on the construction of a multiple horizon tree with variable depth, exploited to improve the prediction capabilities of the designed stochastic Model Predictive Control. Every branch of the tree is here considered a disturbance realization scenario for the control problem; paths with higher probability are extended further in the future, since they may have more impact on the performance. We will realize this strategy to compare the performances obtained with the longest path to those of the one found by the original algorithm.

The main aim of our project is to implement and test an energy management controller based on a stochastic dynamic programming method [9] to compute the optimal control policy of torque and power energy management. This controller must address both fuel economy and battery usage constraints. In order to do this, the speed of the vehicle has to be estimated according to a Markov Chain. Multivariate Markov Chain theory ([10]) may be exploited if the available profiles are more than one. Finally, this controller may be considered optimal if the Markov chains model predicts with a good accuracy the driver behaviour and the vehicle model is significantly accurate.

Some issues related to real time processing are highlighted and addressed in [11] where the goal was to obtain a functional controller for a real vehicle with the procedure discussed above:

- the procedure must operate within current computation and memory requirements of the hardware;
- pedal response has to be provided rapidly.

These particular issues are not going to be properly considered in this work, because the strategy has not been implemented in a real vehicle, but of course the whole methodology must be fast and efficient, and the whole optimization procedure has to be realized within the interval of one second.

The thesis is structured as follows: chapter 2 introduces Markov Chain theory and provides a description of each prediction method that has been designed and tested. Chapter 3 illustrates the main characteristics of the vehicle model with parallel architecture; chapter 4 instead is dedicated to the optimization procedure with dynamic programming and to design a real time algorithm with stochastic dynamic programming. Finally conclusions and future perspectives are described in chapter 5.

Software for our simulation is Matlab$^{®}$ version R2016b, installed on a laptop with Windows 10, Processor Intel$^{®}$ I7, 8 Gb Ram.

2 Markov Chain theory for speed prediction

2.1 Markov Chain theory: main concepts and predicting purposes

According to [12], given a probabilistic space (Ω, $\mathcal{F}$, Pr), where Ω is a set of outcomes, $\mathcal{F}$ is σ-algebra, that is, a set of subset of Ω, and Pr is a probability measure on Ω, a finite Markov Chain $\mathcal{C}$ is a sequence of random variables ($\mathcal{C}$: $\Omega \rightarrow \mathcal{W}$) that provides a theoretical model to describe behaviours of particular discrete time systems, like speed or throttle of a vehicle. At each time instant, the state of these systems belongs to a finite set $\mathcal{W} = \{w_1, w_2, w_3, ..., w_s\} \subset \mathbb{R}$. Transitions between states are regulated by the following probabilistic law:

Definition 2.1 The sequence of random variables $\mathcal{C}$ is a finite Markov Chain with states belonging to the finite set $\mathcal{W}$, if, for each integer k and with $w_i, w_b, w_j \in \mathcal{W}$, b,i,$j \in [1, s]$ and with $Pr[\mathcal{C}_{k+1} = w_i | \mathcal{C}_k = w_j, ..., \mathcal{C}_1 = w_b] > 0$, we have

$$Pr[\mathcal{C}_{k+1} = w_i | \mathcal{C}_k = w_j, ..., \mathcal{C}_1 = w_b] = Pr[\mathcal{C}_{k+1} = w_i | \mathcal{C}_k = w_j] \tag{2.1}$$

In other words, whatever evolution has characterized $\mathcal{C}$ before the current state, the state at time instant $k+1$ depends only on the current state. The set of transition probabilities $\{t_{ij}, i, j \in [1, s]\}$ determines the probabilistic behaviour of the chain. In particular, t_{ij} describes the probability to have a transition from the current state w_j to the future state w_i.
These probabilities are stored in a stochastic matrix (2.2), which is a nonnegative matrix having each column sum equal to one; in fact, elements belonging to the j-th column of $\mathbf{T}$ are probabilities of transitioning from the current state w_j to the future states w_1, or w_2, ... or w_s. These events are mutually exclusive and exhaustive of all the possibilities, so the sum of their probabilities is equal to 1:

$$\mathbf{T} = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1s} \\ t_{21} & t_{22} & \cdots & t_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ t_{s1} & t_{s2} & \cdots & t_{ss} \end{bmatrix} \tag{2.2}$$

In our particular application, we suppose to know the state $w(k)$ at time k. All other values at time instants $t > k$ are unknown and have to be estimated by means of the Markov Chain theory, whose main property is that the probability distribution of the state $w(k+1)$ at time instant $k+1$ is determined only by the knowledge of the current state of the chain, without considering the previous behaviour of the process; therefore, given the probability distribution $\mathbf{p}(k) \in \mathbb{R}^s$, where $[\mathbf{p}(k)]_j = Pr[w(k) = w_j]$, the probability distribution $\mathbf{p}(k+1)$ is computed according to the following equation:

$$\mathbf{p}(k+1) = \mathbf{T}\mathbf{p}(k), \tag{2.3}$$

where $\mathbf{T} \in \mathbb{R}^{s \times s}$ is the transition probability matrix, where, as we discuss above:

$$[T]_{i,j} = Pr[w(k+1) = w_i | w(k) = w_j] \qquad \forall i, j \in \{1, ..., s\} \tag{2.4}$$

The estimation of the transition probabilities plays an important role in correctly predicting the speed. The first approach considers a sequence $\mathcal{M}$ of states belonging to $\mathcal{W}$, with L given measurements (data a priori). Each state w_i of $\mathcal{M}$ is associated to a quantization interval, defined as $\mathcal{I}_i = \{w \in \mathbb{R} : (w_{i-1} + w_i)/2 < w \leq (w_i + w_{i+1})/2\}$ for all $i \in \{1, ..., s\}$. Then we define:

$$\mathcal{K}_{ij} = \{k \in [1, L] : w(k+1) \in \mathcal{I}_i, w(k) \in \mathcal{I}_j\} \tag{2.5}$$

$$n_{ij} = |\mathcal{K}_{ij}| \tag{2.6}$$

$$n_j = \sum_{i=1}^{s} n_{ij} \tag{2.7}$$

where n_{ij} is the number of transitions from w_j to w_i, while n_j is the whole number of transitions from w_j. The components of the transition matrix are then estimated in this way:

$$[T]_{i,j} = \frac{n_{ij}}{n_j} \qquad \forall i, j \in 1, ..., s. \tag{2.8}$$

This particular assumption is due to the following proposition:

Proposition 2.2 A state w_i is said to be *positive recurrent* if there is a non-zero probability that the process can return to w_i, and the expected return time is not infinite. Consider the set $\mathcal{M}$ of measurement belonging to $\mathcal{W}$. Assume $Pr[w(k) = w_j], j \in 1, ..., s$, is defined by the Markov chain (2.3), and let the transition probability matrix $\mathbf{T}$ be estimated by (2.8). If each state of the Markov Chain $\mathcal{M}$ is positive recurrent, then

$$\lim_{L \to \infty} [T]_{ij} = Pr[w(k+1) = w_i | w(k) = w_j]. \tag{2.9}$$

The estimation procedure (2.8), based on the positive recurrence assumption, works properly with a batch of data which spans the entire state-space of the Markov Chain, that is, every state which belongs to the state-space has to be visited at least once. On the other hand, if for example a state w_t with $t \in \{1, ..., s\}$ is never visited, the corresponding number of transitions n_t is equal to 0, therefore, each component of the t-th column of the matrix is not a number, because of the division by 0 in (2.8). However this issue can be solved by substituting the t-th column with the canonical vector $\mathbf{e}_t = [0, \cdots, 0, 1, 0 \cdots, 0]^T$, with all entries equal to 0 except for the t-th which is unitary,that is, without different informations, the probability to remain in the current state is equal to one, while other transitions have probability equal to 0.

The previous approach estimates the transition probabilities by considering all the a priori data, which can be regarded as a useful initialization procedure.
In this chapter, according to [8], the speed of the vehicle is modelled as a discrete-time stochastic process $w(\cdot)$. The realization at time $k \in \mathbb{Z}_{0+}$ of the process, $w(k)$, belongs to the finite set $\mathcal{W} = \{w_1, w_2, w_3, ..., w_s\} \subset \mathbb{R}$, where $w_i < w_{i+1}$ for all $i \in \{1, ..., s-1\}$, and represents the quantized value of the current speed. The cardinality of $|\mathcal{W}|$ represents a trade off between the complexity of the stochastic model and its ability to estimate the correct interval of values for the speed. Our main interest is to provide a recursive method to estimate the speed change in real

time; in other words, every time we get a new measurement, the algorithm has to update on-line the transition matrix with the new observed transition. So, the new transition probabilities are computed as follows:

$$[T]_{i,j}(k) = \frac{n_{ij}(k)}{n_j(k)} \qquad \forall i, j \in 1, ..., s. \tag{2.10}$$

The following equations, derived in [7] and in [8], provide a recursive method to update the transition probabilities. The vector $\delta_j \in \{0,1\}^s$, $\forall j \in \{1, ..., s\}$, indicates which transition has occurred at time k: $[\delta_j]_i(k) = 1$ if and only if $w(k) \in I_i$ and $w(k-1) \in I_j$.

$$n_j(k) = n_j(k-1) + \sum_{i=1}^{s} [\delta_j(k)]_i \tag{2.11}$$

$$\lambda_j(k) = \frac{\sum_{i=1}^{s} [\delta_j(k)]_i}{n_j(k)} \tag{2.12}$$

$$[\mathbf{T}(k)]^j = (1 - \lambda_j(k))[\mathbf{T}(k-1)]^j + \lambda_j(k)\delta_j(k). \tag{2.13}$$

where $j \in 1, ..., s$ and $[\mathbf{T}(k)]^j$ is the j-th column of transition matrix T. n_j is the number of transitions starting from the state w_j, λ_j measures the sensitivity of transition matrix to new data, while equation (2.13) updates the j-th column of the transition matrix.

$n_j(0)$ and $\mathbf{T}(0)$ have to be initialized properly: one idea is to use some data available a priori, and the relation (2.8), otherwise we can simply impose $n_j(0) = 0$ and $\mathbf{T}(0) = \mathbf{I}_s$, where $\mathbf{I}_s$ is the identity matrix of dimension s, s being the number of possible states. The former idea is useful if the drive-cycle considered is stationary and, of course, if some data are known a priori; in this case, the algorithm exploits the knowledge of the past of the driving cycle in order to predict immediately with a good accuracy the future values of speed. The other idea instead has to be applied when no data are available a priori: the transition matrix is initialized with the identity matrix, that is, given any state, the probability to remain in the current state is equal to one. At the beginning, the lack of knowledge about the past of the driving cycle makes difficult to predict correctly future values; therefore, we need to wait some data before obtaining an accurate prediction.

According to (2.12), the sensitivity of transition matrix to new data decreases exponentially with the increased amount of data, therefore the whole recursive update procedure described by equations (2.11), (2.12), (2.13) is reliable for stationary drive cycles or for drive cycles for which the available data is limited. The decreasing sensitivity of transition matrix to new data becomes an interesting issue to solve if speed profiles describe real driving conditions, which are usually characterized by different types of road, weather and traffic conditions, and by the driver's status. A possible solution to overcome this problem is to re-initialize the procedure after a big number of data, with $n_j(0) = 0 \ \forall i, j \in 1, ..., s$ and $\mathbf{T}(0) = \mathbf{I}_s$, in order to eliminate old past data which could have a big influence in speed prediction. Another possibility is suggested in [8], where the update procedure for λ is replaced by the following one:

$$\lambda_j(k) = \bar{\lambda} \sum_{i=1}^{s} [\delta_j(k)]_i \tag{2.14}$$

where $\bar{\lambda} \in \{0, 1\}$ is a constant value which trades off the sensitivity of transition matrix to new data with the convergence rate. Unfortunately this approach does not provide a good description of the probability distribution of the next state, because there is no mathematical explanation of how to properly estimate the parameter $\bar{\lambda}$. So this approach will not be discussed any more.

2.2 Multivariate Markov Chain

In the previous section we introduced a procedure to model the speed of the vehicle as a stochastic process, that can be described by a Markov Chain with s states. Now we apply Multivariate Markov Chain theory to extend the algorithm to multiple sequences, in order to develop better models for better estimation performances. For more details see [13]. Let us assume to have K different data sequences that we regard as realizations of discrete-time stochastic processes taking s possible values (states) in the set $\mathcal{W} = \{w_1, w_2, ..., w_s\}$, as we discussed in section 2.1. The number of possible states must be the same for all data sequences because the transition matrices must be square. So we define $\mathbf{p}_l(k)$ as the probability distribution of the l-th sequence at time instant k. At time k, if the l-th sequence is in state w_j, $j \in [1, s]$, the associated probability distribution is initialized to $\mathbf{p}_l(k) = \mathbf{e}_j = [0, ..., 0, \underbrace{1}_{\text{j-th element}}, 0, ..., 0]^T$. Then we introduce $\mathbf{T}^{(lv)}$ as the matrix which stores the transitions probabilities computed considering the transitions occurring from the states belonging to the v-th sequence to the states referred to l-th sequence ($l, v \in [0, K]$). The first step to determine the components of this matrix is to count number the transition frequencies $n_{ij}^{(lv)}$ from the state w_j in the v-th sequence to the state w_i in the l-th sequence; then we need to compute the whole number of transitions $n_j^{(lv)}$ starting from the state w_j.
As we saw in (2.8), the components of the new transition matrices are estimated in this way:

$$[T]_{i,j}^{(lv)}(k) = \begin{cases} \frac{n_{ij}^{(lv)}(k)}{n_j^{(lv)}(k)} & \text{if } n_j^{(lv)}(k) \neq 0 \qquad \forall i, j \in 1, ..., s \qquad \forall l, v \in 1, ..., K. \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

The probability distribution of the v-th sequence at time $k+1$ is determined by the weighted average of the product between the transition matrix $[\mathbf{T}]^{(lv)}$ and the probability distribution at time k, $\mathbf{p}_v(k)$:

$$\mathbf{p}_v(k+1) = \sum_{v=1}^{K} \lambda_{lv} \mathbf{T}^{(lv)} \mathbf{p}_v(k) \qquad \forall l, v \in 1, ..., K, \tag{2.16}$$

where λ_{lv}, $l, v \in 1, ..., K$, are positive weights characterized by the following condition:

$$\sum_{v=1}^{K} \lambda_{lv} = 1 \qquad \forall l, v \in 1, ..., K.$$

Equation (2.16) can be written in matrix form as follows:

$$\mathbf{p}(k+1) \equiv \begin{bmatrix} \mathbf{p}_1(k+1) \\ \mathbf{p}_2(k+1) \\ \vdots \\ \mathbf{p}_K(k+1) \end{bmatrix} = \begin{bmatrix} \lambda_{11}\mathbf{T}^{(11)} & \lambda_{12}\mathbf{T}^{(12)} & \cdots & \lambda_{1K}\mathbf{T}^{(1K)} \\ \lambda_{21}\mathbf{T}^{(21)} & \lambda_{22}\mathbf{T}^{(22)} & \cdots & \lambda_{1K}\mathbf{T}^{(2K)} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{K1}\mathbf{T}^{(K1)} & \lambda_{K2}\mathbf{T}^{(K2)} & \cdots & \lambda_{KK}\mathbf{T}^{(KK)} \end{bmatrix} \begin{bmatrix} \mathbf{p}_1(k) \\ \mathbf{p}_2(k) \\ \vdots \\ \mathbf{p}_K(k) \end{bmatrix} \equiv \mathbf{Q}\mathbf{p}(k) \tag{2.17}$$

Although the sum of the terms in each column of the transition matrices $\mathbf{T}^{(lv)}$ is still equal to 1, the sum of the columns of the matrix $\mathbf{Q}$ is not one, because of the different values of the weights λ_{ij}. However, the following propositions are still valid:

Proposition 2.3 If the parameters λ_{lv}, for $1 \leq l, v \leq K$, are positive, then the matrix $\mathbf{Q}$ has an eigenvalue equal to one and the other eigenvalues have modulus less than or equal to one.

Proposition 2.4 Suppose that $\lambda_{lv} > 0$ for $1 \leq l, v \leq K$ and the matrices $\mathbf{T}^{(lv)}$, $1 \leq l, v \leq K$, are irreducible, i.e., their associated graph is strongly connected. Then there is a unique vector $\mathbf{p}$=$(\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_s)^T$ such that $\mathbf{p} = \mathbf{Q}\mathbf{p}$ and

$$\sum_{i=1}^{K} [\mathbf{p}_v]_i = 1 \qquad [\mathbf{p}_v]_i \geq 0 \qquad 1 \leq v \leq K.$$

The proofs of the previous propositions are given in [13].According to the previous propositions, we can state that $\mathbf{p}_v(k+1)$ is a probability distribution vector, while $\mathbf{p}(k+1)$ is not.
Now we need to define a method to obtain an estimation of the model parameters λ_{lv}.

2.2.1 Model Parameters Estimation

The vector $\mathbf{p}$ can be described as $\mathbf{p}$=$(\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_s)^T$. The estimate of $\mathbf{p}$ is $\hat{\mathbf{p}}$=$(\hat{\mathbf{p}}_1, \hat{\mathbf{p}}_2, ..., \hat{\mathbf{p}}_s)^T$. The idea described in [13] is to find λ_{lv} which minimizes $\|\hat{\mathbf{p}} - \mathbf{Q}\hat{\mathbf{p}}\|$ for a certain vector norm $\|\cdot\|$. Thus, according to the proof of Proposition 2.4:

$$\begin{bmatrix} \hat{\mathbf{p}}_1 \\ \hat{\mathbf{p}}_2 \\ \vdots \\ \hat{\mathbf{p}}_K \end{bmatrix} \approx \begin{bmatrix} \lambda_{11}\mathbf{T}^{(11)} & \lambda_{12}\mathbf{T}^{(12)} & \cdots & \lambda_{1K}\mathbf{T}^{(1K)} \\ \lambda_{21}\mathbf{T}^{(21)} & \lambda_{22}\mathbf{T}^{(22)} & \cdots & \lambda_{2K}\mathbf{T}^{(2K)} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{K1}\mathbf{T}^{(K1)} & \lambda_{K2}\mathbf{T}^{(K2)} & \cdots & \lambda_{KK}\mathbf{T}^{(KK)} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{p}}_1 \\ \hat{\mathbf{p}}_2 \\ \vdots \\ \hat{\mathbf{p}}_K \end{bmatrix} \tag{2.18}$$

Once considered equation (2.18), one possible way to estimate the parameters λ_{lv} is to solve the following problem:

$$\begin{cases} \qquad \min_{\lambda} \|\hat{\mathbf{Q}}\hat{\mathbf{p}} - \hat{\mathbf{p}}\| \\ \text{subject to} \\ \qquad \sum_{v=1}^{K} \lambda_{lv} = 1 \\ \qquad \lambda_{lv} \geq 0 \quad \forall\, v \in \{1, ..., K\} \end{cases} \tag{2.19}$$

Then by using$\|\cdot\|_\infty$[1], as vector norm, equation (2.19) becomes the following one:

$$\begin{cases} \qquad \min_{\lambda} \max_{i} \left| \left[\sum_{v=1}^{K} \lambda_{lv}\mathbf{T}^{(lv)}\mathbf{p}_v - \mathbf{p}_l \right]_i \right| \\ \text{subject to} \\ \qquad \sum_{v=1}^{K} \lambda_{lv} = 1 \\ \qquad \lambda_{lv} \geq 0 \quad \forall\, v \in \{1, ..., K\} \end{cases} \tag{2.20}$$

where $[\mathbf{z}]_i$ denotes the i-th entry of the vector $\mathbf{z}$. Finally, as suggested in [13] and [10], the previous problem can be reformulated as s linear programming problems. Let us define the quantity ω_j as an auxiliary variable considered as the objective function that has to be minimized. So, for each l

[1]Given a vector of n components $\mathbf{X} = [x_1, x_2, ...x_n]$,$\|\mathbf{X}\|_\infty = \max_i |x_i|, \forall i \in 1, ..., n$

we have:

$$\begin{cases} \qquad \min\limits_{\lambda} \omega_l \\ \text{subject to} \\ \qquad \begin{pmatrix} \omega_l \\ \omega_l \\ \vdots \\ \omega_l \end{pmatrix} \geq \hat{\mathbf{p}}^{(l)} - \mathbf{B} \begin{pmatrix} \lambda_{l1} \\ \lambda_{l2} \\ \vdots \\ \lambda_{lK} \end{pmatrix} \\ \qquad \begin{pmatrix} \omega_l \\ \omega_l \\ \vdots \\ \omega_l \end{pmatrix} \geq -\hat{\mathbf{p}}^{(l)} + \mathbf{B} \begin{pmatrix} \lambda_{l1} \\ \lambda_{l2} \\ \vdots \\ \lambda_{lK} \end{pmatrix} \\ \qquad \sum_{v=1}^{K} \lambda_{lv} = 1, \qquad w_l \geq 0 \\ \qquad \lambda_{lv} \geq 0 \qquad \forall\, v \in \{1, ..., K\} \end{cases} \tag{2.21}$$

where

$$\mathbf{B} = [\hat{\mathbf{T}}^{l1}\hat{\mathbf{p}}^{(1)} | \hat{\mathbf{T}}^{l2}\hat{\mathbf{p}}^{(2)} | \dots | \hat{\mathbf{T}}^{ls}\hat{\mathbf{p}}^{(s)}]. \tag{2.22}$$

The previous problems can be solved by using linear programming procedures, which will be discussed in the next subsection.

2.2.2 Linear Programming

Definition 2.5 A linear programming problem takes the following form:

$$\begin{cases} \qquad \min\limits_{\hat{\mathbf{x}}} f^T \hat{\mathbf{x}} \\ \text{subject to} \\ \qquad \mathbf{A}\hat{\mathbf{x}} \leq \mathbf{b} \\ \qquad \mathbf{A}_{eq}\hat{\mathbf{x}} = \mathbf{b}_{eq} \\ \qquad \mathbf{z} \leq \hat{\mathbf{x}} \leq \mathbf{u} \end{cases} \tag{2.23}$$

where $f^T\hat{\mathbf{x}}$ is the objective function to minimize with respect to the vector $\hat{\mathbf{x}}$, which is subject to some constraints expressed either as inequalities $\mathbf{A}\hat{\mathbf{x}} \leq \mathbf{b}$, $\mathbf{z} \leq \hat{\mathbf{x}} \leq \mathbf{u}$ (Vectors $\mathbf{z}$ and $\mathbf{u}$ contain the lower and the upper constraints, respectively) or as equalities $\mathbf{A}_{eq}\hat{\mathbf{x}} = \mathbf{b}_{eq}$.

In (2.21), $\hat{\mathbf{x}}, \mathbf{A}, \mathbf{b}, \mathbf{A}_{eq}, \mathbf{b}_{eq}$ are:

$$\hat{\mathbf{x}} = \begin{pmatrix} \omega_j \\ \lambda_{l1} \\ \lambda_{l2} \\ \vdots \\ \lambda_{lK} \end{pmatrix} \qquad \mathbf{A} = \begin{bmatrix} -\mathbf{1}_s & -\mathbf{B} \\ -\mathbf{1}_s & \mathbf{B} \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -\hat{p}^{(l)} \\ \hat{p}^{(l)} \end{bmatrix} \qquad \mathbf{A}_{eq} = \begin{bmatrix} 0, 1, 1, \dots 1 \end{bmatrix} \qquad \mathbf{b}_{eq} = \begin{bmatrix} 1 \end{bmatrix} \tag{2.24}$$

where $\mathbf{1}_s$ is a vector $s \times 1$ of ones. So problem (2.21) for each l becomes:

$$\begin{cases} \qquad \min_{\lambda} \omega_l \\ \text{subject to} \\ \qquad \mathbf{A}\hat{\mathbf{x}} \leq \mathbf{b} \\ \qquad \mathbf{A}_{eq}\hat{\mathbf{x}} = \mathbf{b}_{eq} \\ \qquad \sum_{v=1}^{K} \lambda_{lv} = 1, \qquad \omega_l \geq 0 \\ \qquad \lambda_{lv} \geq 0 \qquad \forall\, v \in \{1, ..., K\} \end{cases} \tag{2.25}$$

These linear programming problems can be solved by using the *interior point method*, an algorithm which solves linear and non linear convex optimization problems. Matlab® function *linprog* implements the algorithm and provides the right values for the different λ.

2.3 Data Analysis and prediction procedure

2.3.1 Min-Max Mapping

We now introduce a tool which will be useful to analyse the data involved in our project. Let us consider a sequence of data $\mathbf{x}$ characterized by a maximum x_{max} and a minimum x_{min}, we want to bijectively map this sequence into a new interval of data $\mathbf{y} \in [y_{min}, y_{max}]$. The new value assumed by each component of sequence $\mathbf{x}$ can be computed according to the following equation:

$$y = (y_{max} - y_{min}) * (x - x_{min})/(x_{max} - x_{min}) + y_{min}; \tag{2.26}$$

where $y \in \mathbf{y}$ and $x \in \mathbf{x}$.

2.3.2 Data Illustration and Mapping

Now we illustrate various data profiles which we will use to simulate previous algorithms in order to compare performances and evaluate the efficiency of every implemented method. The first drive-cycle discussed is called *Indian*, it describes a journey of 95 seconds of a scooter and is made up of three data sequences: the first collects the speed values, and it is illustrated in Figure 2.1; the second instead contains percentage values about the flow of power from the engine, i.e, the throttle; finally, the third contains force values applied to slow down or stop the motion of the vehicle, i.e., the brake.

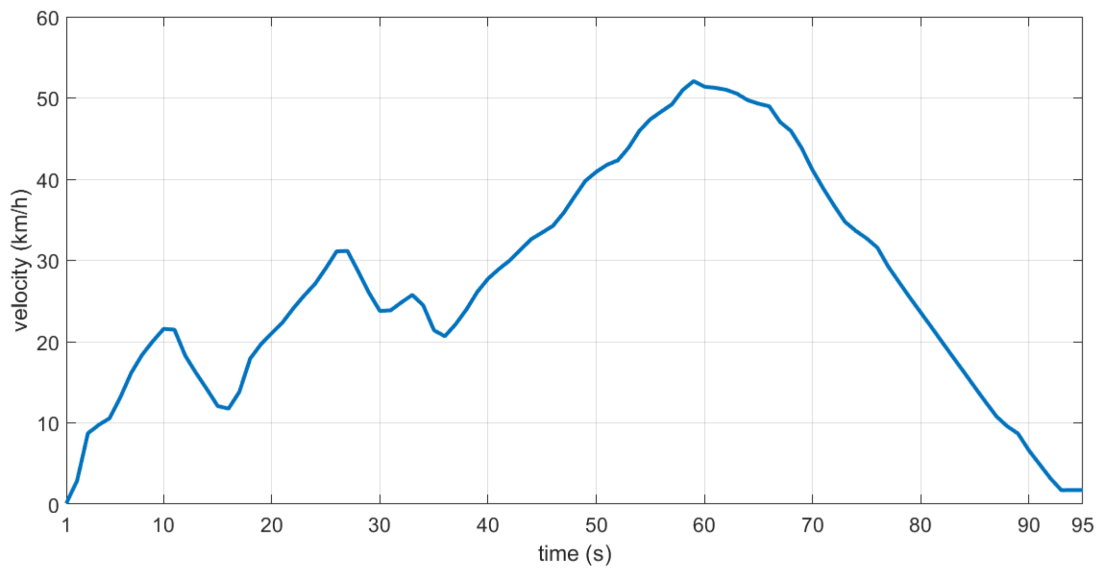


Figure 2.1: The figure illustrates the speed profile of the Indian Drive-Cycle; the profile is a data sequence containing 95 speed values of a journey of a scooter.

The complementarity between throttle and brake data is exploited to merge the two sequences: first of all, each data of the profiles are normalized with a min-max mapping procedure, within the intervals $[0, 1]$ for the throttle and $[-1, 0]$ for the brake, respectively.
Once the normalization is performed, they are added together as illustrated in Fig. 2.2.

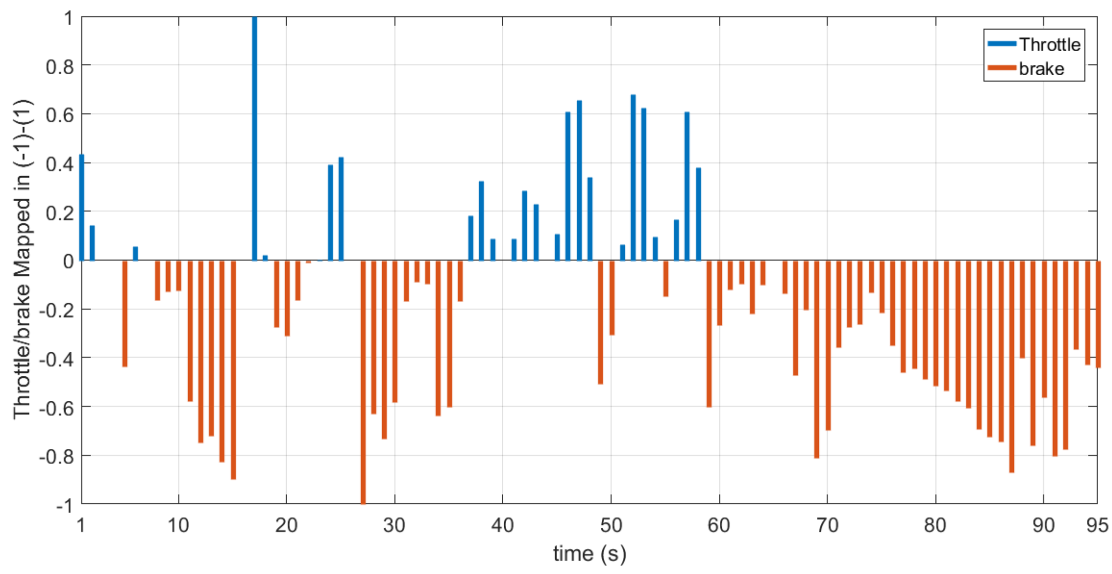


Figure 2.2: The figure illustrates the first normalization on throttle and brake data of the Indian Drive Cycle.

Then, another operation of mapping is applied to the obtained sequence, in order to refer it to the interval $[0, 60]$, where 60 [km/h] is considered the maximum velocity allowed for a scooter. The result is shown in Fig. 2.3.

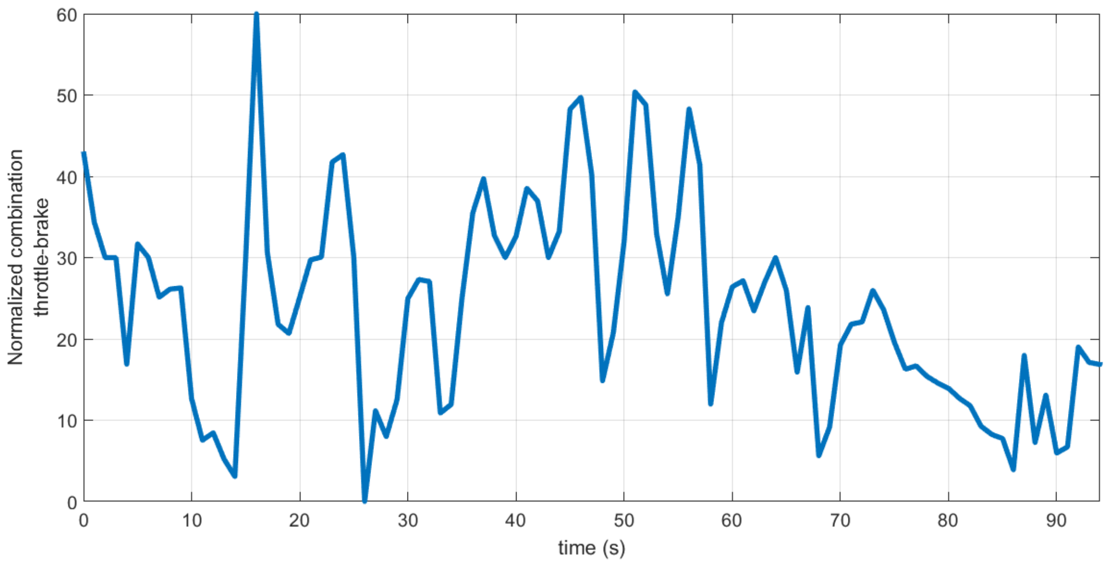


Figure 2.3: The figure illustrates the sequence after the second normalization on throttle and brake data of the Indian Drive Cycle.

The second drive cycle involved is the Worldwide Harmonized Light Vehicles Test Cycle (*WLTP*), which defines *a global harmonized standard for determining the levels of pollutants and CO2 emissions, fuel or energy consumption, and electric range for light-duty vehicles (passenger cars and light commercial vans)*. This consists of four parts, *drive-cycleL*, *drive-cycleM*, *drive-cycleH*, and *drive-cycleE*, respectively, which describe the behaviour of a vehicle in different situations (low, medium, high and very high speed, respectively), and are useful to evaluate the adaptability of the procedures to very different real time behaviour. These drivecycles are characterized by two profiles, illustrated in Fig. 2.4 which are speed and power, respectively.

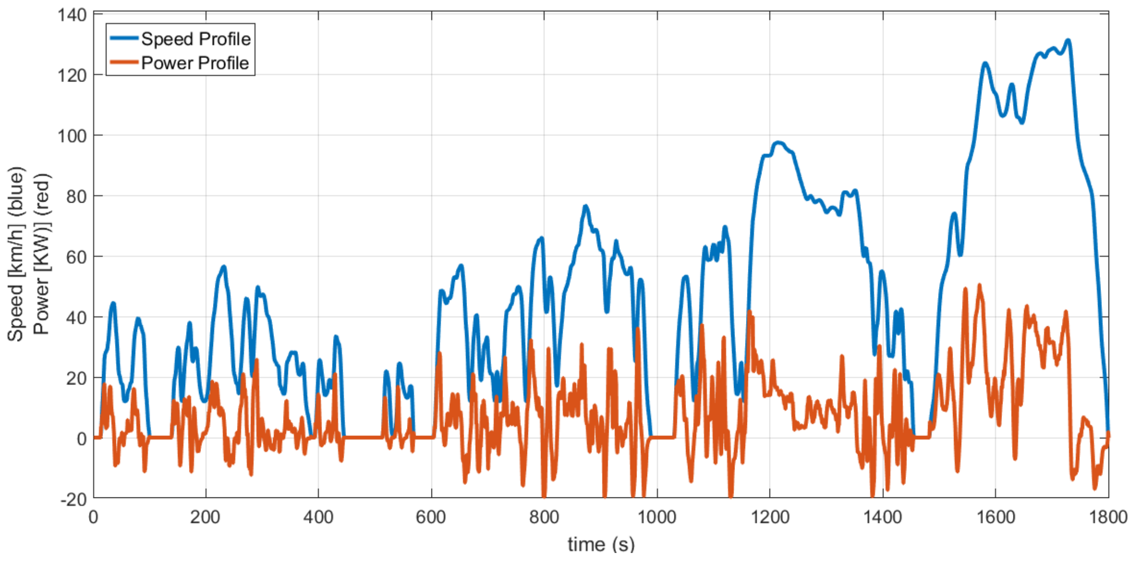


Figure 2.4: The figure illustrates both WLTP profiles. The maximum reachable velocity in this case is set as 140 km/h

Before evaluating the behaviour of every algorithm for different driving cycles, we now discuss how the data have to be collected and analysed by our procedure. Scooters can usually reach 60 [km/h] as maximum speed, while cars 140 [km/h], and the velocity intervals are $[0, 60]$ [km/h] and $[0, 140]$ [km/h], respectively. As we discussed in section 2.1, the speed is modelled as a stochastic process $w(\cdot)$ taking values in $\mathcal{W} \subset \mathbb{R}$, so we have to quantize and then refer each speed value to a particular interval of speed $\mathcal{I}_i$. The operation of quantization introduces an error, because the current velocity is approximated. A key role in the prediction accuracy is played by the size of those intervals, which have to be chosen properly: a large interval can reduce the complexity of the algorithm, making it faster, but as a consequence the accuracy of the prediction could become poor, because the quantization error becomes significant. A small interval instead is able to improve the accuracy of the estimation, but could slow down the whole procedure. A good compromise for our purpose is obtained with the value $l_q = 0.5$ [km/h]; so the speed range for a scooter is split into 121 intervals $(60/l_q + 1)$, starting from $w_1 = [0, 0.25)$, continuing with $w_2 = [0.25, 0.75)$ and so on until the last one $w_{121} = [59.75, 60]$. If we are using the second drive cycle, instead, the maximum velocity is 140 [km/h], so the number of intervals is $140/0.5 + 1 = 281$. Every time a new data value x is acquired, it has to be referred to its interval w_j according to the following procedure:

$$w_j = 1 + round(x/0.5), \tag{2.27}$$

where *round* is the approximation procedure which rounds the real number $x/0.5$ to its nearest integer. When the fractional part is exactly 0.5, the round function approximates with the smallest integer larger than $x/0.5$. Another important factor is the sampling time, which is the interval between the acquisition of two consecutive data. Our procedure will acquire a data sample at every second (sampling time $T_s = 1s$).

2.3.3 Transition Matrices

Real time procedures receive one data per second from each data sequence involved. Once new data are acquired, new transitions are set between previous and current states of the sequences, and the matrices which store the transitions between states are recursively updated according to equations (2.11), (2.12), (2.13). When transition matrices are finalized, the algorithm can start predicting next states of speed.

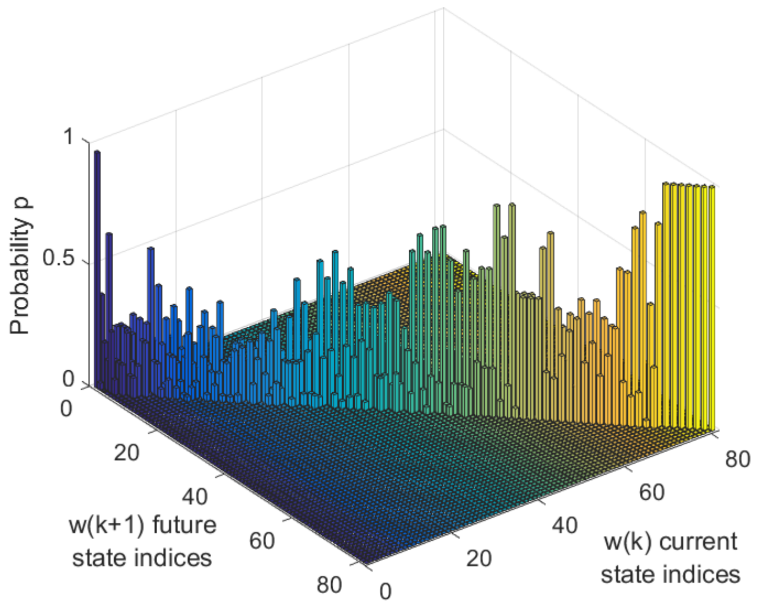


Figure 2.5: 3D Illustration of a Transition matrix

If the speed profile is the only one we are interested in, only one transition matrix has to be updated to properly compute the probability distribution of the stochastic process (see equation (2.30)). If data sequences are two (speed and power or speed and combination of throttle and brake), transitions occur between states of the same sequence and between data values belonging to different sequences, thus the involved transition matrices are four. Equation (2.17) becomes:

$$\begin{bmatrix} \mathbf{p}_1(k+1) \\ \mathbf{p}_2(k+1) \end{bmatrix} = \begin{bmatrix} \lambda_{11}\mathbf{T}^{(11)} & \lambda_{12}\mathbf{T}^{(12)} \\ \lambda_{21}\mathbf{T}^{(21)} & \lambda_{22}\mathbf{T}^{(22)} \end{bmatrix} \begin{bmatrix} \mathbf{p}_1(k) \\ \mathbf{p}_2(k) \end{bmatrix} \tag{2.28}$$

(2.28) can also be written as:

$$\begin{cases} \mathbf{p}_1(k+1) = \lambda_{11}\mathbf{T}^{(11)}\mathbf{p}_1(k) + \lambda_{12}\mathbf{T}^{(12)}\mathbf{p}_2(k) \\ \mathbf{p}_2(k+1) = \lambda_{21}\mathbf{T}^{(21)}\mathbf{p}_1(k) + \lambda_{22}\mathbf{T}^{(22)}\mathbf{p}_2(k) \end{cases} \tag{2.29}$$

where $\mathbf{p}_1$ and $\mathbf{p}_2$ are probability distributions: the former allows to compute the next state of speed, the latter instead allows to evaluate the future values of the second sequence (for example, the power); $\mathbf{T}^{(11)}$, $\mathbf{T}^{(22)}$ store the probabilities of transitions between previous and current state of the same profile, while $\mathbf{T}^{(12)}$ count probabilities of transition between previous state of power and current state of the speed profile. Finally $\mathbf{T}^{(21)}$ stores transition probabilities between previous values of speed and current state of the power profile. Figure 2.5 shows the shape of a transition matrix which stores transitions between speed data at the end of a simulation. The shape of this matrix is diagonal, that is, probabilities are centred around the diagonal axis; this is because vehicle speed is highly correlated, and transitions usually occur between states which are quite close to each other. Thus, a transition between two states of speed that are very distant usually has a probability value equal to 0.

After computing the transition probabilities matrices and defining how to compute probability distributions, the priority becomes to define a method which is able to use these matrices and the current state of the input sequences to predict the future states with a good accuracy. In section 2.4 we will discuss the general algorithm derived through Markov Chain Theory, while in 2.5 and 2.5.1 we will introduce possible variants of the algorithm.

2.4 Algorithms for prediction of speed: Markov Chain Method

The first method exploits equation (2.3): the probability distribution at time $k+1$ is computed to determine the most probable future state, by means of the product between the transition matrix $\mathbf{T}$ and the column vector of the current probability distribution.

Algorithm 1 Markov Chain Procedure

Data Acquisition and Initialization: At time k current data is acquired and then mapped into the corresponding state $w_j, j \in \{1, 2, ..., s\}$. $\mathbf{T}$ is updated based on the last transition between the previous state and the new current state according to (2.13). The probability distribution is then initialized to the column vector $\mathbf{p}(k) = \mathbf{e}_j = [0, ..., 0, \underbrace{1}_{\text{j-th element}}, 0, ..., 0]^T$, while L represents the number of states to predict.

Update iteration: For $t = 1 : L$, the probability distribution at time $k+t$ is updated by

$$\mathbf{p}(k+t) = \mathbf{T}\mathbf{p}(k+t-1) = \mathbf{T}^t\mathbf{p}(k) \tag{2.30}$$

Search for the maximum probability value: The most probable transition starting from current state w_j at time t is chosen by taking the highest probability value in vector $\mathbf{p}(k+t)$; the state w_i, which is associated to the highest probability value is selected as future state.
Path Creation: Starting from w_j, the initial state, the procedure builds a path which contains the future states of the process, computed during the previous steps.

Note that at the beginning of the procedure, when the initial state is w_j, $\mathbf{p}(k) = \mathbf{e}_j = [0, ..., 0, 1, 0, ..., 0]^T$, $\mathbf{p}(k+1)$ become $[\mathbf{T}(k)]^j$, that is the j-th column of the transition matrix $\mathbf{T}$ at time k. If more data sequences are involved, the procedure is the same, but transitions matrices are four and probability distributions are two, and we have to update them according to (2.29). The procedure is summarized in Fig. 2.6.

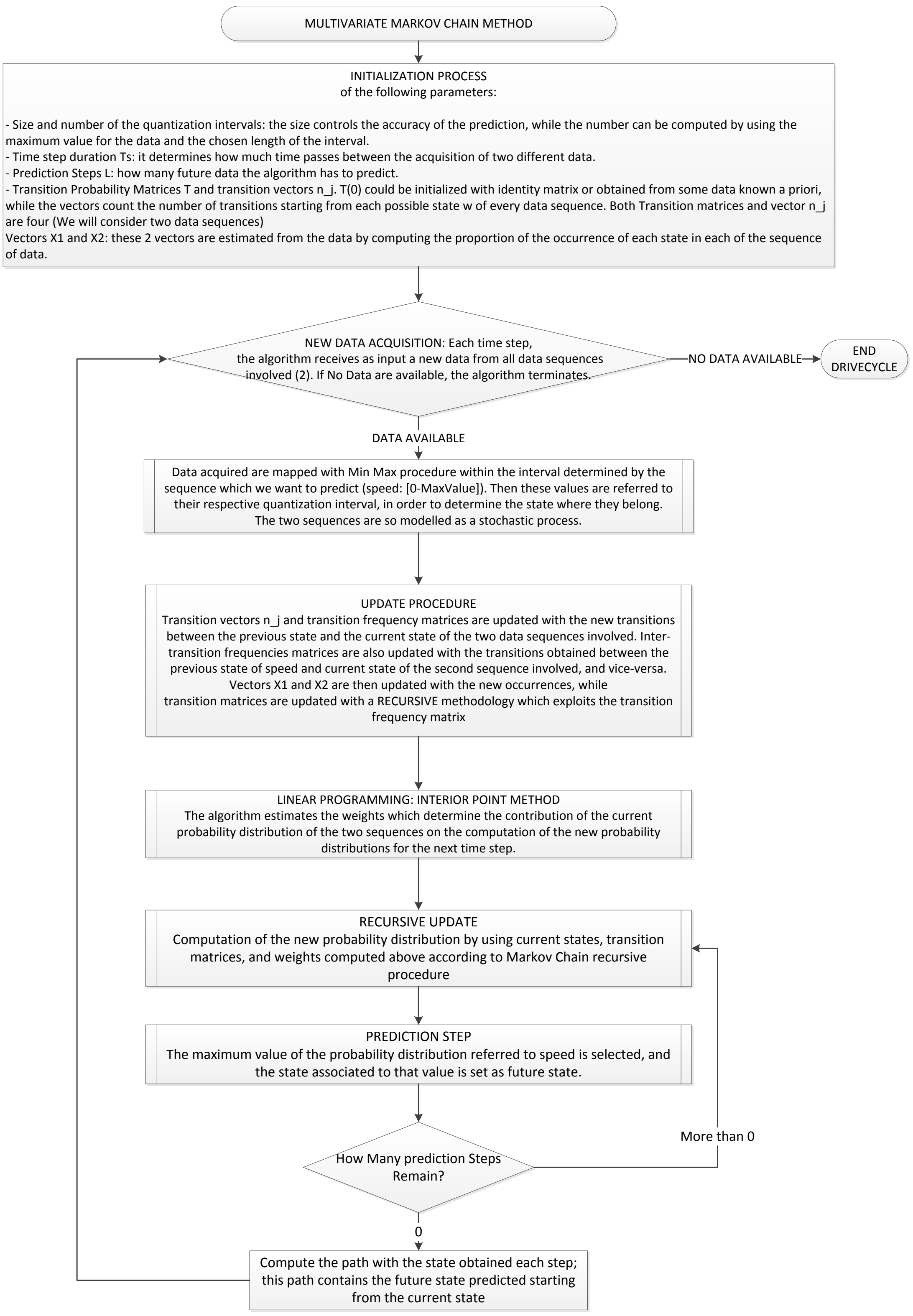

MULTIVARIATE MARKOV CHAIN METHOD
INITIALIZATION PROCESS of the following parameters:
- Size and number of the quantization intervals: the size controls the accuracy of the prediction, while the number can be computed by using the maximum value for the data and the chosen length of the interval.
- Time step duration Ts: it determines how much time passes between the acquisition of two different data.
- Prediction Steps L: how many future data the algorithm has to predict.
- Transition Probability Matrices T and transition vectors n_j. T(0) could be initialized with identity matrix or obtained from some data known a priori, while the vectors count the number of transitions starting from each possible state w of every data sequence. Both Transition matrices and vector n_j are four (We will consider two data sequences)
Vectors X1 and X2: these 2 vectors are estimated from the data by computing the proportion of the occurrence of each state in each of the sequence of data.
NEW DATA ACQUISITION: Each time step, the algorithm receives as input a new data from all data sequences involved (2). If No Data are available, the algorithm terminates.
NO DATA AVAILABLE
END DRIVECYCLE
DATA AVAILABLE
Data acquired are mapped with Min Max procedure within the interval determined by the sequence which we want to predict (speed: [0-MaxValue]). Then these values are referred to their respective quantization interval, in order to determine the state where they belong. The two sequences are so modelled as a stochastic process.
UPDATE PROCEDURE
Transition vectors n_j and transition frequency matrices are updated with the new transitions between the previous state and the current state of the two data sequences involved. Inter-transition frequencies matrices are also updated with the transitions obtained between the previous state of speed and current state of the second sequence involved, and vice-versa. Vectors X1 and X2 are then updated with the new occurrences, while transition matrices are updated with a RECURSIVE methodology which exploits the transition frequency matrix
LINEAR PROGRAMMING: INTERIOR POINT METHOD
The algorithm estimates the weights which determine the contribution of the current probability distribution of the two sequences on the computation of the new probability distributions for the next time step.
RECURSIVE UPDATE
Computation of the new probability distribution by using current states, transition matrices, and weights computed above according to Markov Chain recursive procedure
PREDICTION STEP
The maximum value of the probability distribution referred to speed is selected, and the state associated to that value is set as future state.
How Many prediction Steps Remain?
More than 0
0
Compute the path with the state obtained each step; this path contains the future state predicted starting from the current state

Figure 2.6: Scheme of the Multivariate Markov Chain Method for speed prediction

2.5 Tree Generation Procedure

The second method realizes a "scenario tree" which describes the most likely paths of future sequence realizations by exploiting the main characteristics of a Markov Chain. This scenario tree provides all the admissible paths of the stochastic process, as described in [8], where the goal was to solve an optimization problem with a control sequence per scenario. Our aim, instead, is to find the most probable path among all the available scenarios; thus the chosen path must have sufficient depth in order to provide the desired prediction steps. The main elements of this procedure are the following:

- The set of Tree Nodes, $\tau = \{\mathcal{N}_1,\mathcal{N}_2,....,\mathcal{N}_n\}$, where $\mathcal{N}_1$ is the root node, while $\mathcal{N}_n$ is the last node added to the tree; each node represents a possible state for the sequence of data involved in the procedure.
- The probabilities $p_{\mathcal{N}_i}$ of reaching the node $\mathcal{N}_i$,$i = 1, ...m$, starting from the root node; these values are the edges of the tree and they are computed through the equation $\mathbf{p}(k+1) = \mathbf{T}(k)^j$.
- The set of Leaf Nodes $S \subset T$ contains the last node of every existing path in the tree; its cardinality is $n_{leaf} = |S|$.
- The set of Candidate Nodes $C = \{C_1, C_2, ..., C_c\}$: these nodes do not belong to τ, but they represent new possible future states connected by an edge (a probability value) to an existing state $\mathcal{N}_i$ or to a leaf node. c is the dimension of this set, and p_C is the vector containing the probabilities of the candidate nodes.

At the beginning, the tree τ and the leaf set consist of the root node $\mathcal{N}_1$ alone, that is associated to the current state $w(k)$ of the stochastic process, while the set of candidate nodes is evaluated by considering all possible s states that the stochastic process $w(\cdot)$ can reach at the next time step. The candidate node C_i^*, linked to the root node by the edge with maximum probability, is chosen as $\mathcal{N}_2$, added to τ and S, while it is removed from C. $\mathcal{N}_1$ is no more a leaf node, so it is removed from S. The set of candidate nodes is then updated by including all possible children states connected to the last node added to the tree. The vector which stores the probability of each possible edge is expanded with the total probability[2] of the new candidate nodes related to the new leaf node. Finally the procedure is repeated until the tree contains the maximum number of nodes (as suggested by [8]) or until we reach the desired depth for a path. The algorithm is fully described in the following table and is illustrated in Fig. 2.7.

Algorithm 2 Tree Generation Procedure

```
**Data Acquisition and Initialization:** At any time $k$, current velocity data $w_j, j \in \{1, 2, ..., s\}$
is acquired and then quantized.
Set $\tau = \{\mathcal{N}_1\}$, $p_{\mathcal{N}_1} = 1$, vector $\mathbf{p}_C = \mathbf{T}(k)^j$, $c = s$, $n = 1$;

**while** $Level < Level_{max}$ (or $n < n_{max}$)
set $i* = argmax_{i\in\{1,...,c\}}\mathbf{p}_C$ and find $C_i^*$
set $\mathcal{N}_{n+1} = C_i^*$ and $\tau = \tau \cup \mathcal{N}_{n+1}$
update $\mathbf{p}_C = [\mathbf{p}_C, p_C(i^*)\mathbf{T}(k)^{i^*}]^T$;
set $n = n + 1$
set $c = c + s - 1$
update $Level$ if the found node has increased the depth of the tree
remove the $i^*$-th component of $p_C$.
**end while**
```

$Level_{max}$ is the number of prediction steps required, i.e., the desired depth for the longest path of the tree; $Level$ is the current depth reached by the algorithm.

[2]Each entry of the vector stores the product of the probabilities on the edges of the path. The latter starts from the root node until the candidate node.

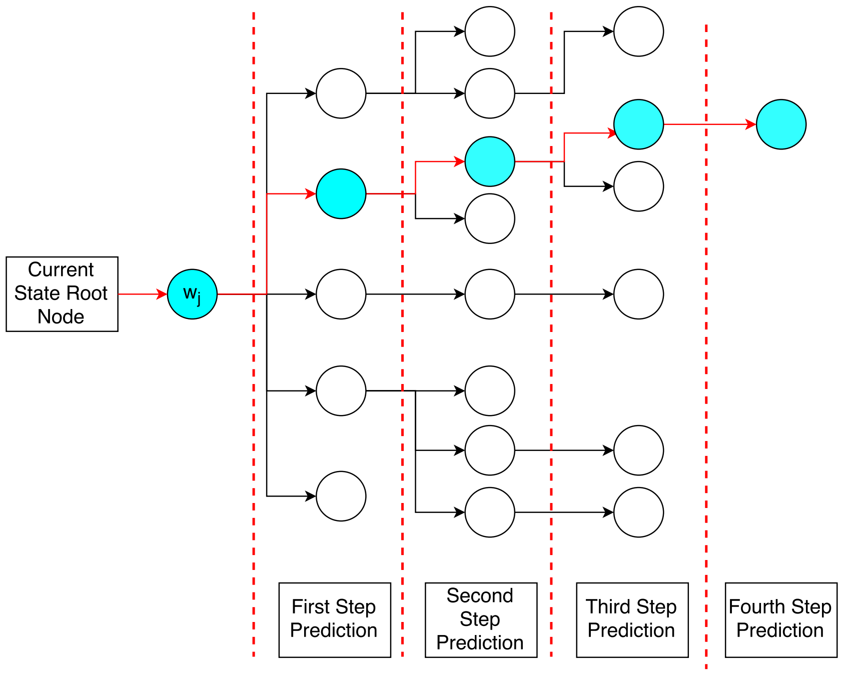


Figure 2.7: This picture provides a scheme of the procedure described by Algorithm 2: w_j is the root node; the best path (here described by red arrows and cyan states) is chosen once the last node added belongs to the requested prediction level. As we can see from the picture, the procedure provides a lot of paths of different depth and chooses the one which firstly reached the desired depth (in this case, the fourth step).

2.5.1 Tree Generation Procedure for Multiple sequences

The procedure described in section 2.5 can not be used with Multivariate Markov Chains, because the probability distribution at time $k+1$ is computed with the contribution of a second sequence of data which is not considered by the previous algorithm. To solve this issue we can introduce a new tree τ_2 which allows us to predict future states of the second sequence. The building procedure of this new tree proceeds in parallel with the first one; when the length of the longest path of the first tree is greater than the longest path belonging to the second tree, this new tree is updated according to the Markov Chain rule (equation (2.30)) until its longest path has the same length as the corresponding one in the first tree. Once this happens, the last node of this path is used to compute the probability distribution of the first sequence at time $k+1$. Given current values w_j and w_i of the two data sequences, $\mathbf{p}(k+1)$ is computed according to the following equation:

$$\mathbf{p}(k+1) = \lambda_{11}\mathbf{T}_{11}(k)^j + \lambda_{12}\mathbf{T}_{12}(k)^i$$

Where $\mathbf{T}_{11}(k)^j$ is the j-th column of the transition matrix $\mathbf{T}_{11}$, while $\mathbf{T}_{12}(k)^i$ is the i-th column of the transition matrix $\mathbf{T}_{12}$. Scheme shown in Fig. 2.8 explains the whole tree procedure with the variant for Multivariate Markov Chains.

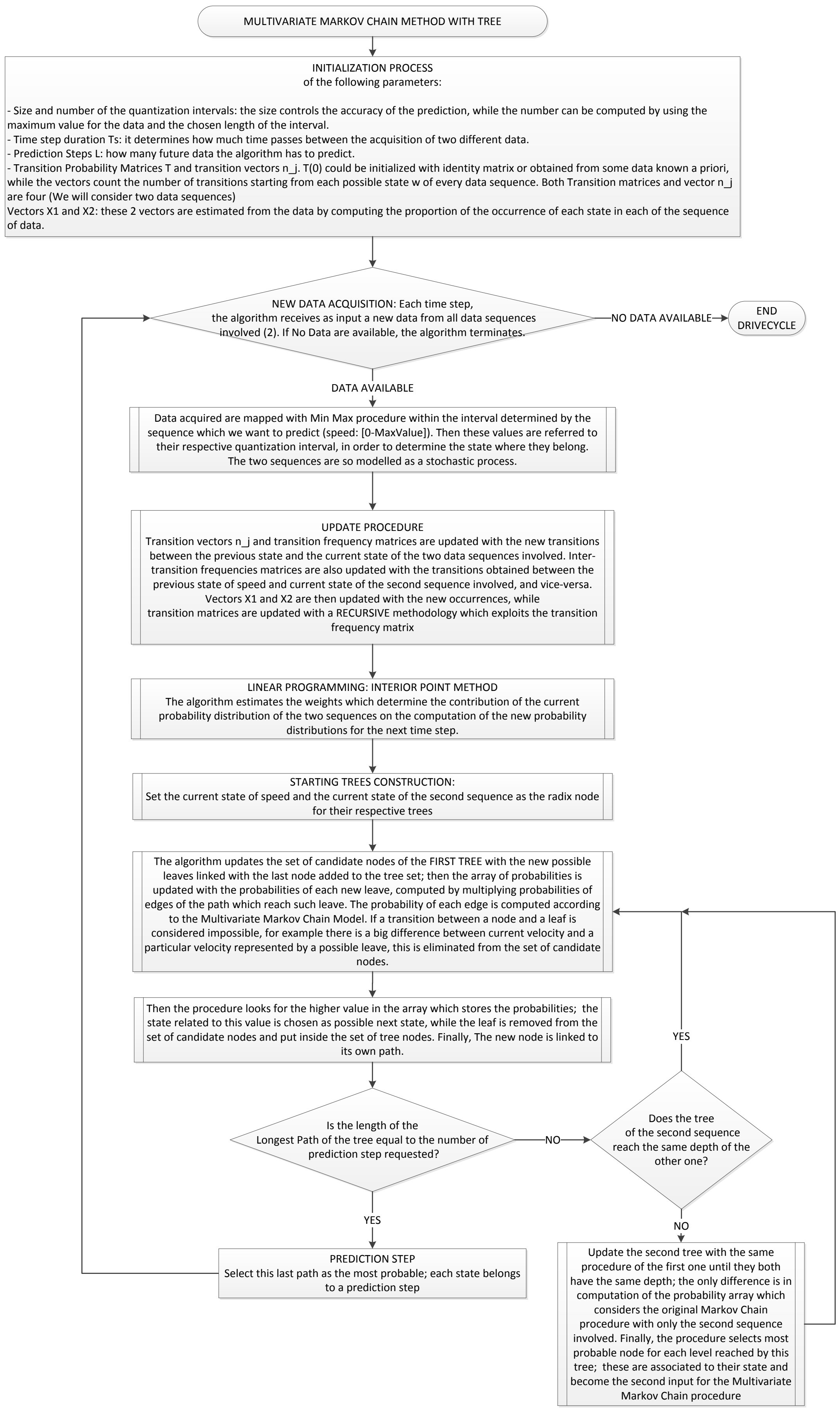

MULTIVARIATE MARKOV CHAIN METHOD WITH TREE
INITIALIZATION PROCESS of the following parameters:
- Size and number of the quantization intervals: the size controls the accuracy of the prediction, while the number can be computed by using the maximum value for the data and the chosen length of the interval.
- Time step duration Ts: it determines how much time passes between the acquisition of two different data.
- Prediction Steps L: how many future data the algorithm has to predict.
- Transition Probability Matrices T and transition vectors n_j. T(0) could be initialized with identity matrix or obtained from some data known a priori, while the vectors count the number of transitions starting from each possible state w of every data sequence. Both Transition matrices and vector n_j are four (We will consider two data sequences)
Vectors X1 and X2: these 2 vectors are estimated from the data by computing the proportion of the occurrence of each state in each of the sequence of data.
NEW DATA ACQUISITION: Each time step, the algorithm receives as input a new data from all data sequences involved (2). If No Data are available, the algorithm terminates.
NO DATA AVAILABLE
END DRIVECYCLE
DATA AVAILABLE
Data acquired are mapped with Min Max procedure within the interval determined by the sequence which we want to predict (speed: [0-MaxValue]). Then these values are referred to their respective quantization interval, in order to determine the state where they belong. The two sequences are so modelled as a stochastic process.
UPDATE PROCEDURE Transition vectors n_j and transition frequency matrices are updated with the new transitions between the previous state and the current state of the two data sequences involved. Inter-transition frequencies matrices are also updated with the transitions obtained between the previous state of speed and current state of the second sequence involved, and vice-versa. Vectors X1 and X2 are then updated with the new occurrences, while transition matrices are updated with a RECURSIVE methodology which exploits the transition frequency matrix
LINEAR PROGRAMMING: INTERIOR POINT METHOD The algorithm estimates the weights which determine the contribution of the current probability distribution of the two sequences on the computation of the new probability distributions for the next time step.
STARTING TREES CONSTRUCTION: Set the current state of speed and the current state of the second sequence as the radix node for their respective trees
The algorithm updates the set of candidate nodes of the FIRST TREE with the new possible leaves linked with the last node added to the tree set; then the array of probabilities is updated with the probabilities of each new leave, computed by multiplying probabilities of edges of the path which reach such leave. The probability of each edge is computed according to the Multivariate Markov Chain Model. If a transition between a node and a leaf is considered impossible, for example there is a big difference between current velocity and a particular velocity represented by a possible leave, this is eliminated from the set of candidate nodes.
Then the procedure looks for the higher value in the array which stores the probabilities; the state related to this value is chosen as possible next state, while the leaf is removed from the set of candidate nodes and put inside the set of tree nodes. Finally, The new node is linked to its own path.
Is the length of the Longest Path of the tree equal to the number of prediction step requested?
NO
Does the tree of the second sequence reach the same depth of the other one?
YES
NO
YES
PREDICTION STEP Select this last path as the most probable; each state belongs to a prediction step
Update the second tree with the same procedure of the first one until they both have the same depth; the only difference is in computation of the probability array which considers the original Markov Chain procedure with only the second sequence involved. Finally, the procedure selects most probable node for each level reached by this tree; these are associated to their state and become the second input for the Multivariate Markov Chain procedure

Figure 2.8: Tree procedure for the Multivariate Markov Chain algorithm. The goal is to predict the future states of then speed.

2.6 Simulations and Results

In this section we simulate the previous algorithms with the driving cycles described in subsection 2.3.1. The Software used for simulation is Matlab® R2016b. The goal of these simulations is to compare the performances in order to choose the best procedure for our purposes. The mean squared error (MSE) is used to measure the quality of the performances; this quantity measures the average of the squares of the deviation, i.e., the difference between the obtained prediction of speed and the true speed. If we denote by x the desired speed, by $\hat{x}$ the prediction and by N the number of data considered, the following equation explains how to compute the MSE:

$$MSE = \frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{N}. \tag{2.31}$$

Another instrument to evaluate the performances of the various estimators is the mean absolute error, which computes the average of the absolute error between the desired and the estimated speed. This quantity measures how much close the predictions are to the true values, and we can compute it in the following way:

$$MAE = \frac{\sum_{i=1}^{N}|x_i - \hat{x}_i|}{N} = \frac{\sum_{i=1}^{N} e_i}{N}. \tag{2.32}$$

MAE is more robust to outliers than MSE is: MAE assigns equal weight to the data, while MSE emphasizes the extremes, because the square of a small number (smaller than 1) is even smaller, on the contrary the square of a big number is even bigger. The differences between the prediction and the desired data usually occur because of the randomness of the real time driver behaviour, which is influenced by many different factors, like traffic congestion, weather and road condition. The ability of the estimator is to quickly learn the changes in behaviour and, as a consequence, to improve the prediction of the speed.

2.6.1 Indian Drivecycle, Markov chain procedure

The first simulation tests the ability of the Markov Chain algorithm to predict the speed profile of Indian drive cycle, by using only the speed data sequence. This profile consists of 95 data, and we assume that it is repeated eight times, thus we can see how the algorithm improves the prediction of a repeated drive cycle starting without any kind of information. The transition matrix is initialized to the identity matrix, i.e., $\mathbf{T}(0) = \mathbf{I}_s$, and every component of the vector $n_j(0)$, which counts the number of transitions starting from the different states, is initialized to zero.
We can see from Fig. 2.9 that the algorithm does not provide any benefit during the first part of the drive-cycle (70-90 seconds), because, in this phase, it is updating the transition probabilities matrix $\mathbf{T}$, so it can return as future speed profile only a delayed version of the drive-cycle itself, because the identity matrix may suggest only the current value of speed as most probable future state. We have already seen that this issue can be solved by initializing the transition matrix with some a priori data.
Once the transition probability matrix has stored the whole behaviour of the drive cycle, the algorithm starts working properly and the computed prediction can better describe the behaviour of the real driver action. Unfortunately, another issue concerning the identification of the speed profile is the small number of data which characterizes this drive-cycle: during computation, when the procedure looks for the maximum value of probability in the array which contains the description of the probability distribution, it could find two or more different values which have the same probability to became the next value of speed. The procedure privileges as next state the last occurrence in the array, that is the state corresponding to the highest velocity. We can see in Fig. 2.9 that when the speed increases, the prediction follows correctly the desired speed; when instead the velocity is reducing due to the breaking action, the prediction has some small peaks which corrupt the whole performance. This issue will be solved by considering a second sequence of data which is able to provide information about throttle and brake actions. When the driving cycle is stationary, as in this case, after the second repetition the performances do not improve significantly; some

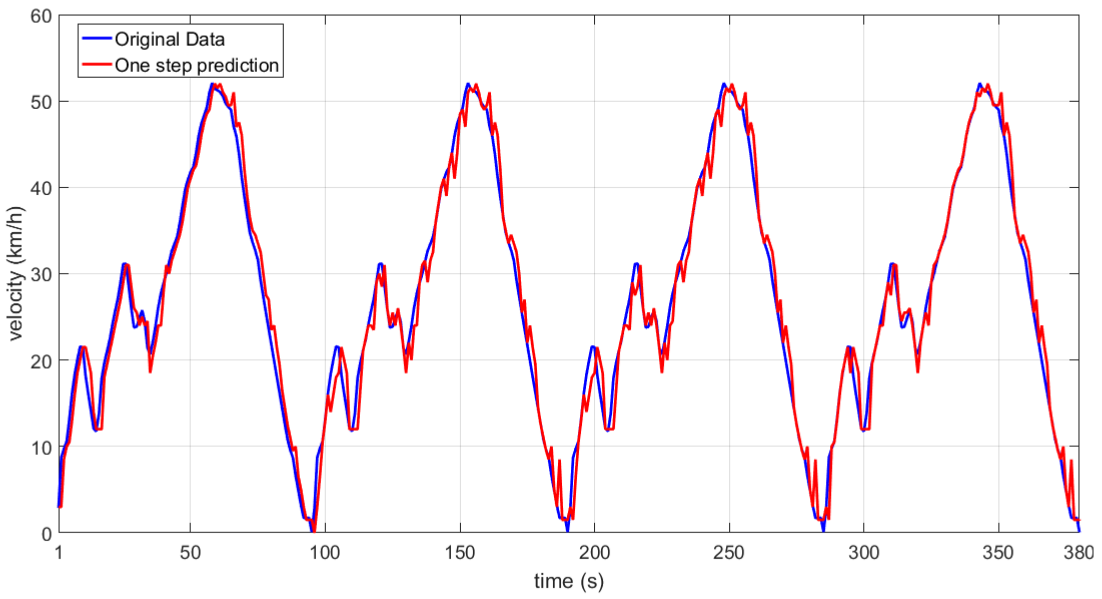


Figure 2.9: Comparison between original data of repeated Indian driving cycle (blue) and data predicted by the Markov Chain Procedure, considering only speed data. Four repetitions are displayed.

little variations happen because Matlab® approximates the term λ of the equation (2.12), which decreases exponentially when the number of data increase. These small approximations cause some little discrepancies in the values of the probabilities distributions, therefore the procedure could slightly change the choice of a future state.

The one step prediction is the same for both procedures discussed above, standard Markov Chain algorithm and tree procedure. Let us see now how the two algorithms work when the required prediction steps are more than one: Fig. 2.10 compares the original drive cycle with One-Step, Two-Steps and Three-Steps predictions computed with the Markov Chain Procedure.

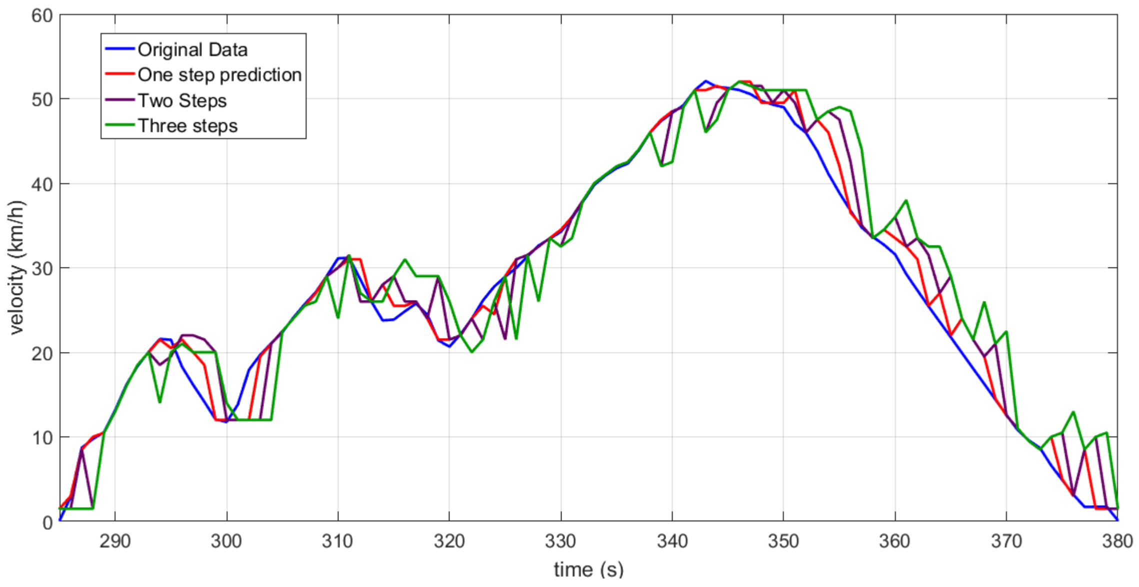


Figure 2.10: Comparison between original profile of Indian Driving cycle (blue) and data predicted by the Markov Chain Procedure, considering one, two, three prediction steps. Data values range in 285-380 [s]

Of course, the difference between predicted and original data values increases with the number

of computed steps; in fact, starting from the current state, the higher is the number of required prediction steps, the lower is the probability to correctly guess the future state, especially if we do not have available information about throttle and brake (or power), which could help understanding better the whole behaviour.
Figure 2.11 compares the two speed predictions (three steps) computed thanks to the different algorithms described.

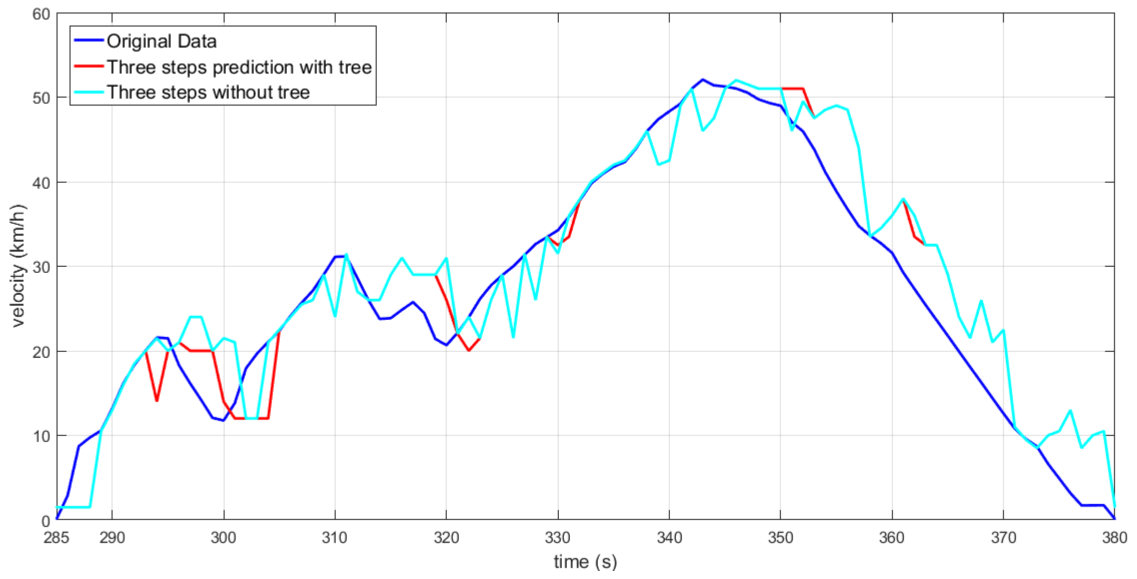


Figure 2.11: Comparison between original profile of Indian Driving cycle (blue) and data predicted by the general procedure of Markov Chain (cyan) and by tree procedure (red), considering only the speed data sequence.

2.6.2 Indian Drive cycle, Multivariate Markov Chain procedure

Let us discuss the benefits provided by the Multivariate Markov Chain algorithm, which computes the prediction by taking into account the contribution of another sequence of data, as, in this case, the combination of throttle and brake. As we saw in section 2.2, the introduction of a new sequence of data requires the computation of four transition matrices, two of them storing inter-transition probabilities between the two data sequences. The shape of the latter matrices is not as diagonal as the other two, so the transition probability distribution, associated to the current state, is distributed between all existing transitions (with the simple Markov Chain method, the probabilities were concentrated in a small interval centred around the current state). Nevertheless our vehicle can only gradually increase or decrease the speed, due to mechanical and physical constraints, so transitions between states characterized by very different values of speed cannot really happen, in spite of these transitions might be associated to a probability value different from 0. Therefore, we need to limit the number of future states to consider: in our case, we state that the difference between two successive velocity values ($w(k+1)$-$w(k)$) cannot exceed V=7 [km/h], in both acceleration and deceleration phases. Therefore, with $l_q = 0.5$ [km/h], only 29 possible values of speed are considered [3].
Fig. 2.12 illustrates the comparison between the original data set and the predicted one, computed without any information about the drive-cycle involved, that is, $\mathbf{T}(0) = \mathbf{I}_s$ and $n_j(0) = 0, 1 \leq j \leq s$. The first part of the driving cycle can be considered as a training sequence, therefore, as before, the prediction obtained here is just a delayed version of the original data. Starting from the second repetition of the drive cycle, we can see that the red line follows almost perfectly the desired

[3] 29 states=$2 \cdot \frac{V}{l_q} + 1$. We obtain this number by observing that we have 14 values of speed in the interval of 7 [km/h] ($\frac{V}{l_q}$). Then, by considering both acceleration and deceleration, this number has to be doubled: $2 \cdot \frac{V}{l_q}$. Finally, we need to consider the current state +1.

behaviour (blue line) in every part of the drive cycle; small visible differences are simply due to quantization errors. Of course, if we build the transition matrices with some prior data before starting the procedure, the algorithm would be able to predict with optimal results already from the starting point.

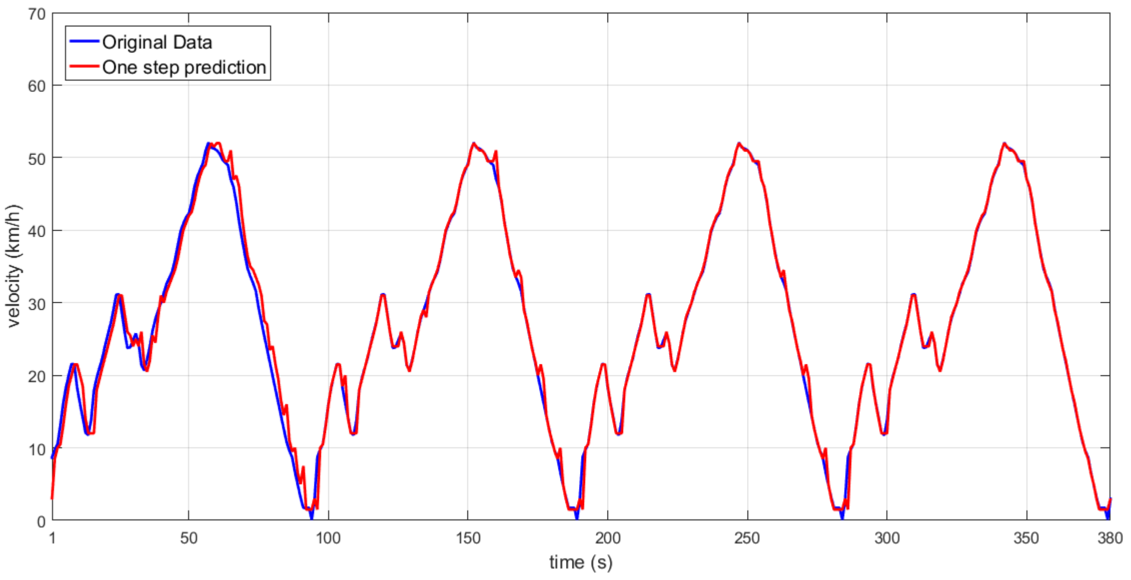


Figure 2.12: Comparison between original data of Indian driving cycle (blue) and one step prediction computed with Multivariate Markov Chain procedure (red), which consider two data sequences. Only four repetitions of Indian driving cycle are illustrated

Fig. 2.13 shows the behaviours of the one, two and three steps predictions compared to the original data. The more we increase the number of required prediction steps, the worse are the results achieved, but, also in this case, they are better than the predictions computed with the simple Markov Chain strategy.

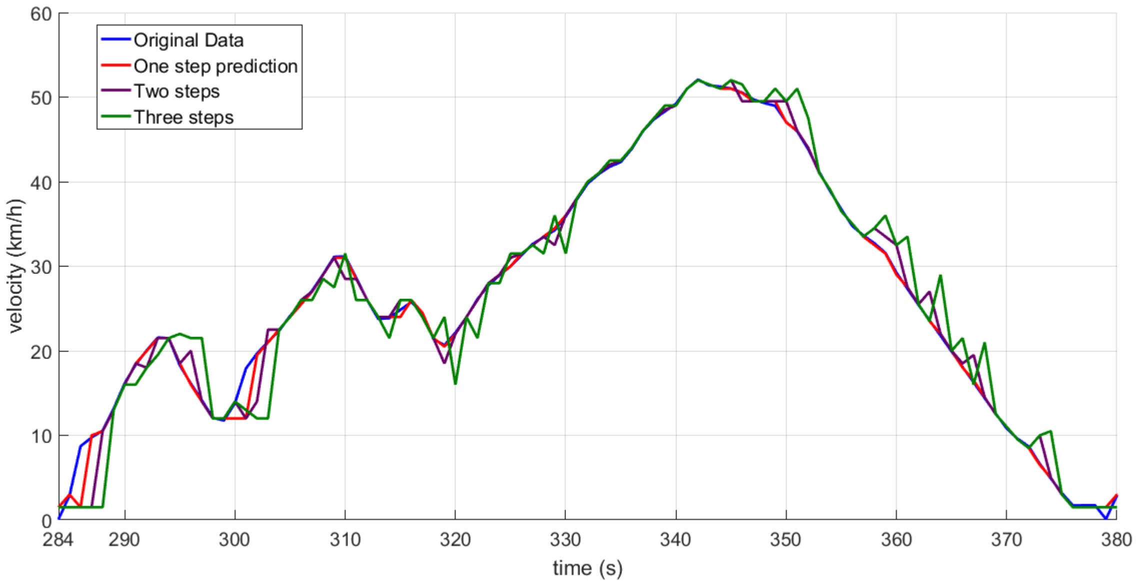


Figure 2.13: Comparison between original data of Indian Driving cycle (blue), one step prediction (red), two steps (green), and three steps predictions (cyan) computed with Markov Chain Algorithm, considering only speed data

To confirm the previous statements, we have stored the MSE and MAE values in tables 2.1 and 2.2.

According to these tables, we can say that the Multivariate Markov Chain Procedure fits almost everywhere the original behaviour, therefore errors are smaller than with the simple Markov Chain method, and mainly due to quantization process. The strategy which builds and exploits the tree does not improve the performance of the general method; actually, the errors are even worse. Generally, if we want to predict a stationary drive cycle and the algorithm applied exploits the Multivariate Markov Chain, the higher the number of data, the more accurate should be the prediction. Sometimes this does not happen, because probability values of two different transitions could be very close; the update procedure and the weights computed by linear programming might change these values during time, so the choice of the future state may change accordingly; this could cause larger MSE or MAE errors in some small intervals, like the one computed excluding the first 500 data.

Table 2.1: Indian drive-cycle with simple Markov Chain strategy: 760 is the number of data.

Data Profiles: Speed and combination of throttle and brake	MSE	MSE: 100 data excluded	MSE: 500 data excluded	MAE: km/h	MAE: 100 data excluded	MAE: 500 data excluded
ONE STEP Markov Chain	3.5471	3.5193	3.3732	1.0590	1.0558	1.0464
TWO STEPS Markov Chain	12.3082	12.1056	11.2140	2.2879	2.262	2.161
THREE STEPS Markov Chain	25.1692	24.6284	24.6860	3.6730	3.6226	3.5815
ONE STEP Multivariate Markov Chain	0.8782	0.7466	0.9511	0.3257	0.2963	0.3655
TWO STEPS Multivariate Markov Chain	4.0905	3.8667	4.7119	1.0503	1.0181	1.291
THREE STEPS Multivariate Markov Chain	11.2087	10.7902	12.5838	1.9980	1.9370	2.1801

Table 2.2: Indian drive cycle with Markov Chain Strategy characterized by tree procedure: 760 is the total number of data.

Data Profiles: Speed and combination of throttle and brake	MSE	MSE: 100 data excluded	MSE: 500 data excluded	MAE	MAE: 100 data excluded	MSE: 500 data excluded
ONE STEP Markov Chain	4.0561	4.0173	3.9235	1.1676	1.1590	1.1366
TWO STEPS Markov Chain	12.9874	12.8924	12.5884	2.4375	2.4122	2.3589
THREE STEPS Markov Chain	26.3611	26.1958	25.8087	3.6654	3.6254	3.5727
ONE STEP Multivariate Markov Chain	1.5589	1.4198	1.3480	0.4892	0.4580	0.4548
TWO STEPS Multivariate Markov Chain	6.1912	5.8835	5.7004	1.7624	1.7240	1.6886
THREE STEPS Multivariate Markov Chain	13.2668	12.7409	12.1012	2.5303	2.4780	2.3791

2.6.3 WLTP Drivecycle, Markov chain procedure

Now we want to test our algorithms with the WLTP drive cycle, described in 2.3.1 and characterized by two profiles, one of speed and one of power, which both consist of 1804 data values (one data per second).

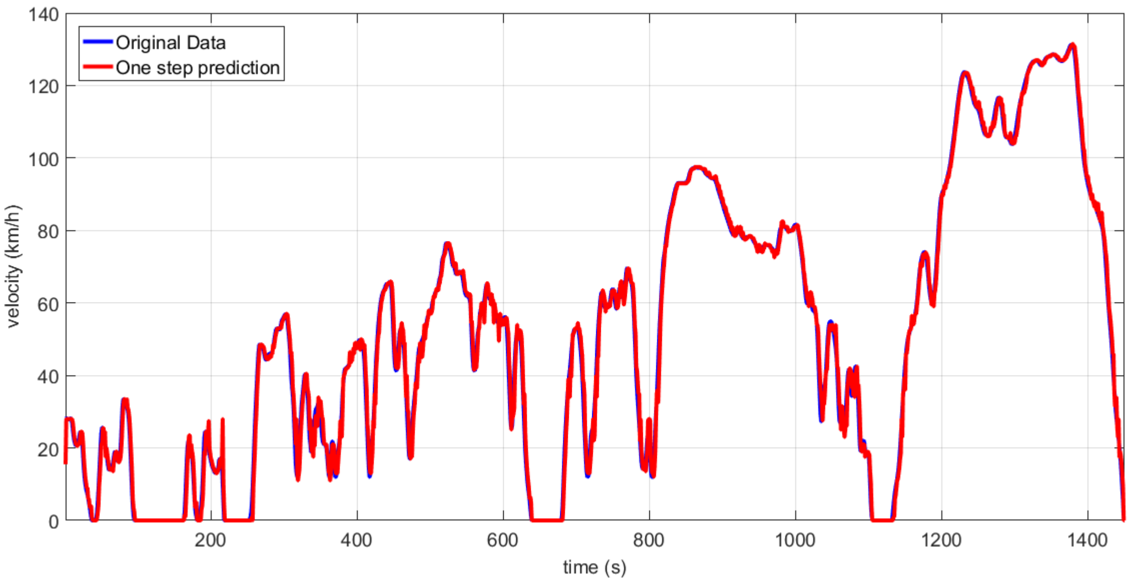


Figure 2.14: Comparison between original data of WLTP driving cycle (blue) and one step prediction (red) computed with Markov Chain Procedure, considering only speed data. The interval is 1-1400 $[s]$

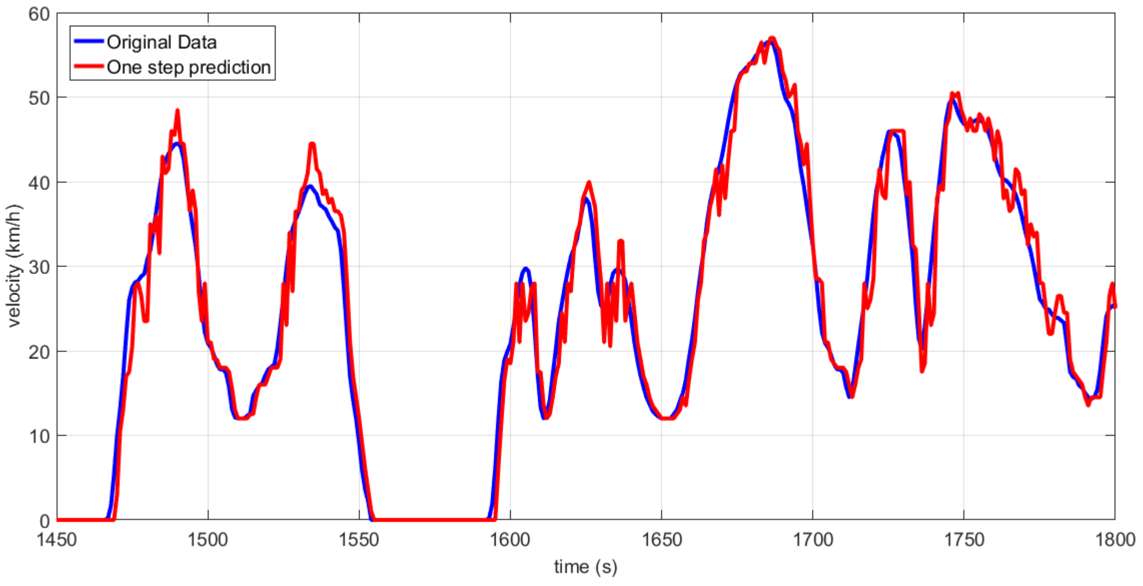


Figure 2.15: Comparison between original data of WLTP driving cycle (blue) and one step prediction (red) computed with Markov Chain Strategy. The interval is 1400-1700 $[s]$

This drive cycle is far from being stationary, because it represents very different types of driver actions; therefore it is very interesting to test the capability of different algorithms to quickly learn the new driver actions. We saw with previous simulations that a good starting point for getting good performances is to initialize the transition matrices involved with some prior data which possibly cover the whole interval of speed. In the following simulations we will consider the first 350 seconds as training data, and the drive cycle is repeated twice (3258 data involved). If no prior data are available, transition matrices could be initialized as follows: $\mathbf{T}^{(11)} = \mathbf{T}^{(22)} = \mathbf{I}_s$, while inter-transition matrices $\mathbf{T}^{(12)}$, $\mathbf{T}^{(21)}$ can be initialized with uniform values for each component: $\mathbf{T}^{(11)}_{ij} = \mathbf{T}^{(22)}_{ij} = 1/N$, where N is the number of possible values for speed and power data sequences. Figures 2.14 and 2.15 show the comparison between original data and one step prediction data computed with the Markov Chain general strategy without considering the power profile. The first figure illustrates the interval starting from the first data to the 1450th, while the second figure shows the interval between 1450th to 1800th data. Finally Fig. 2.16 adds two and three steps prediction profiles, computed with the Markov Chain strategy.

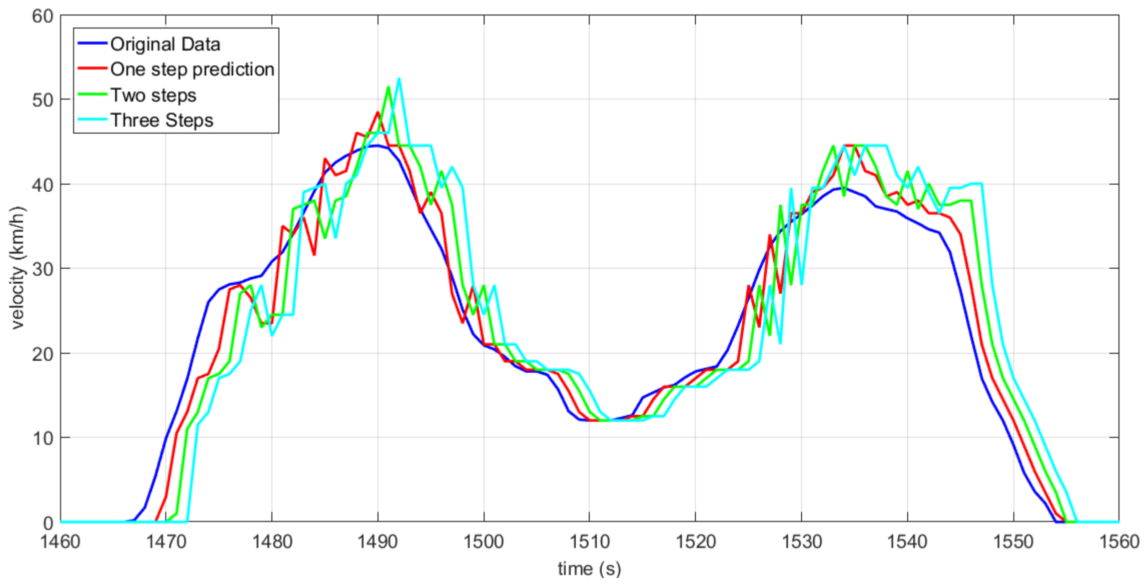


Figure 2.16: Comparison between original data of WLTP driving cycle (blue), one step prediction (red), two steps (green), and three steps predictions (cyan) computed with Markov Chain Strategy. Interval considered is 1460-1560

2.6.4 WLTP Drivecycle, Multivariate Markov chain procedure

The same drivecycle is used to test the Multivariate Markov Chain procedure. Figures 2.17 2.18,2.19 illustrate the comparison between the original speed profile and the one step prediction profile in three different intervals. As before, 350 data are used as training data to initialize the transition matrices involved.

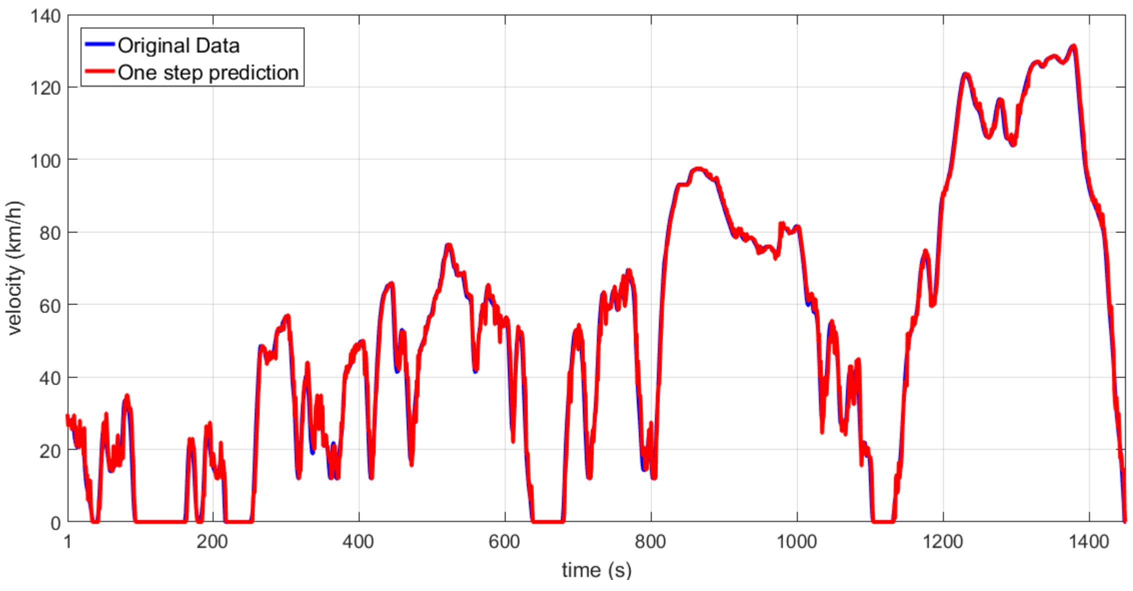


Figure 2.17: Comparison between original data of WLTP driving cycle (blue) and one step prediction (red) computed with Multivariate Markov Chain strategy. Interval $1 - 1450$ [s]

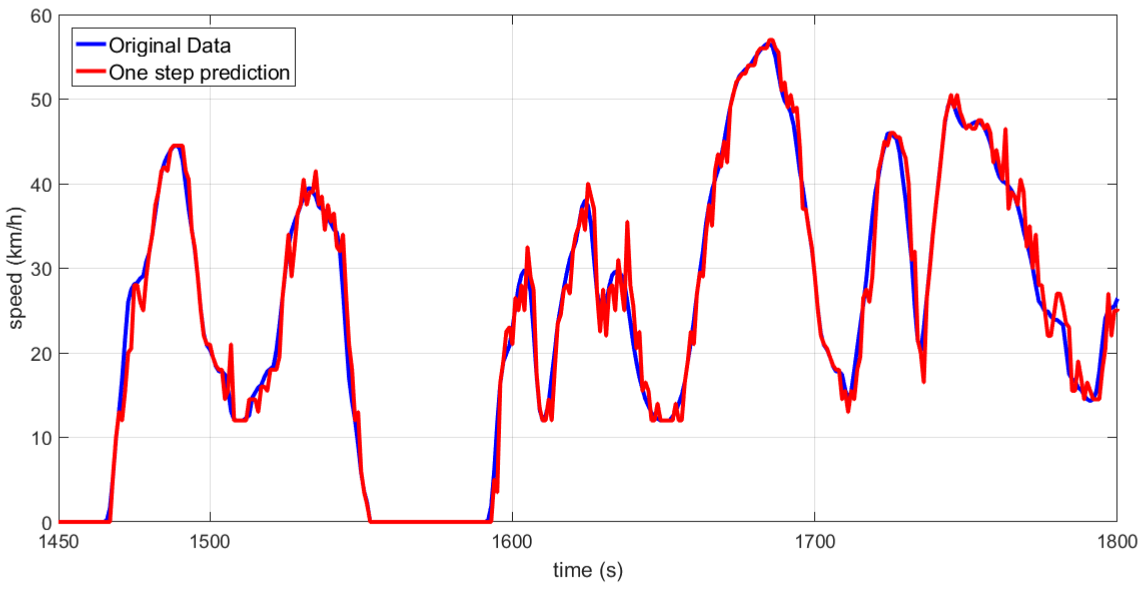


Figure 2.18: Comparison between original data of WLTP driving cycle (blue) and one step prediction (red) computed with Multivariate Markov Chain strategy. Interval $1450 - 1800$ [s]

Previous pictures show that the prediction profiles follow correctly the original dataset, nevertheless sometimes there are some undesired peaks which spoil the performances.

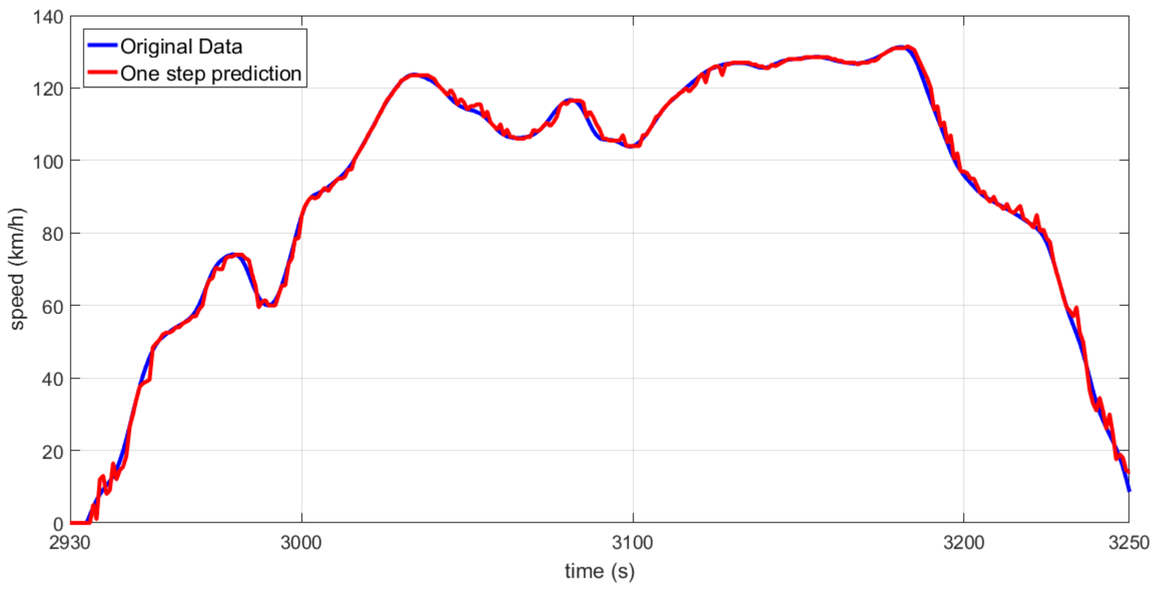


Figure 2.19: Comparison between original data of WLTP driving cycle (blue) and one step prediction (red) computed with Multivariate Markov Chain strategy. Interval 2930 − 3250 [s]

Figure 2.20 compares the behaviour of the different profiles of predicted speed with the original one. Of course, the larger is the number of the prediction steps required, the larger are the discrepancies between original and predicted profile.

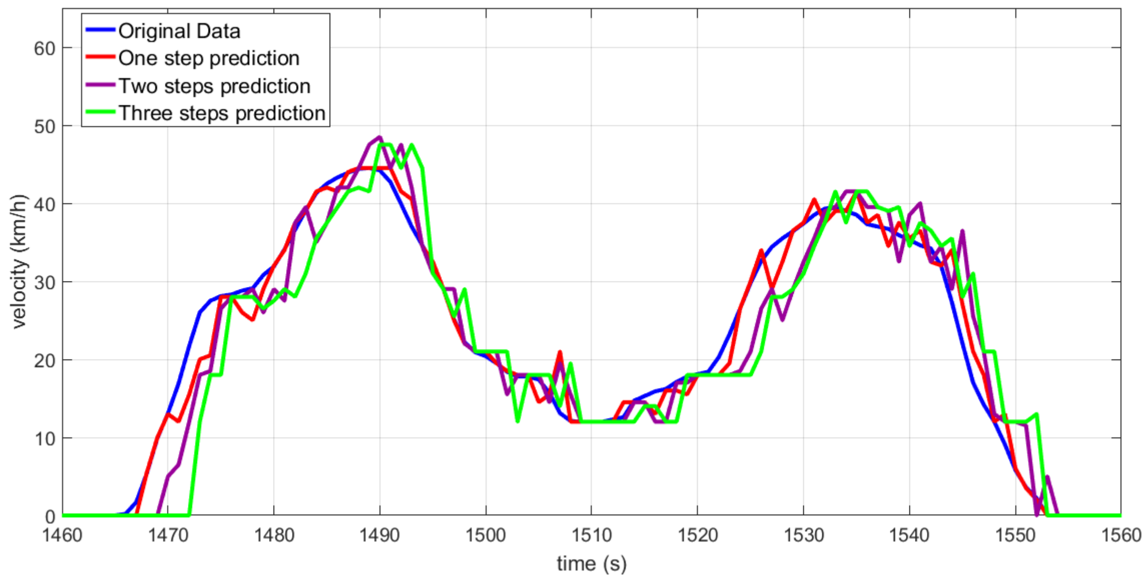


Figure 2.20: Comparison between original data (blue) of WLTP driving cycle, one step prediction (red), two steps (purple), and three steps predictions (green), computed with the Multivariate Markov Chain strategy.

Finally we compute and then store in the following table the values of MSE and MAE for the different procedure adopted. The drivecycle involved is repeated twice, transition matrices are initialized with 350 data used as prior knowledge, so the first data considered for the simulation is *the* 351th. As before, we can see from the table that the more the number of data increases, the more accurate is the prediction, but only the Multivariate Markov Chain method provides a significant improvement.

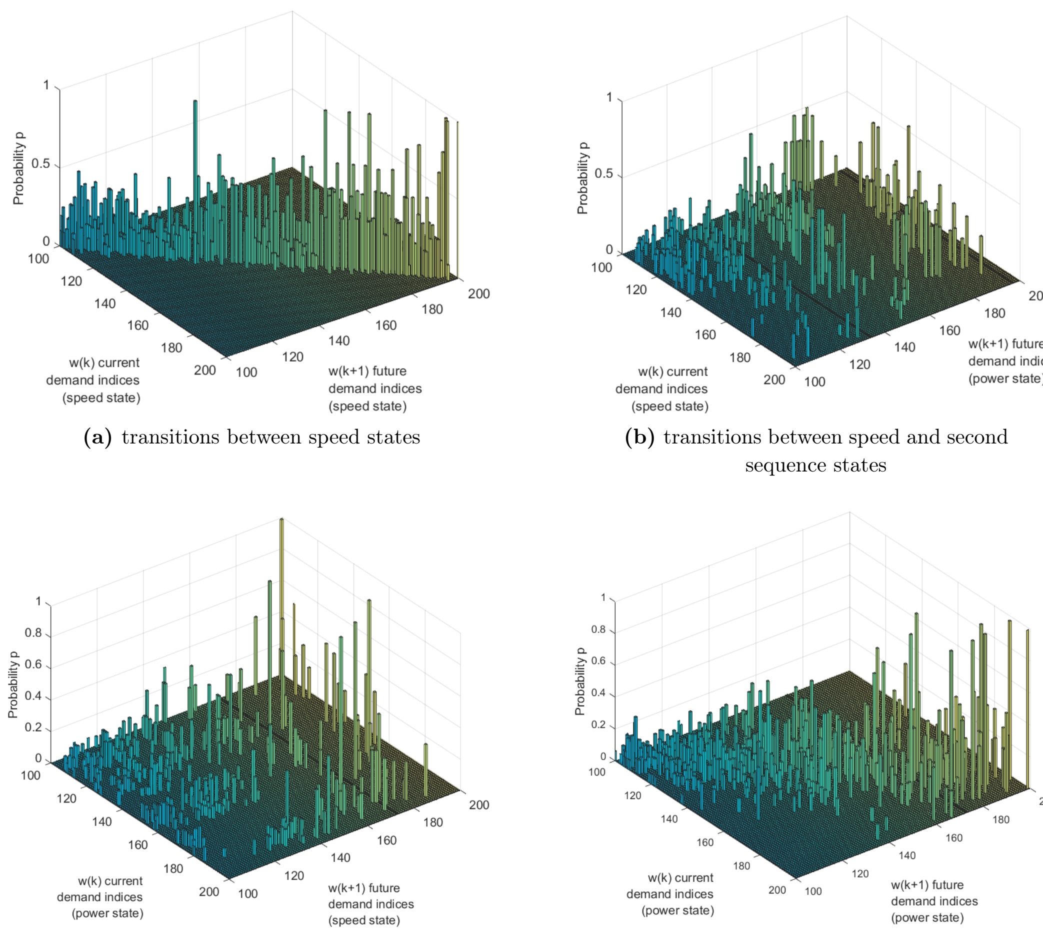
Table 2.3: Errors computed with WLTP driving cycle and predictions obtained with Multivariate Strategy.

Algorithm	MSE	MSE: 500 data excluded	MSE: 1500 data excluded	MAE: km/h	MAE: 500 data excluded	MAE: 1500 data excluded
ONE STEP Markov Chain with Tree	6.0924	5.8222	5.8641	1.5433	1.5198	1.5124
TWO STEPS Markov Chain with Tree	18.6421	17.5716	17.7509	2.7836	2.7199	2.7351
THREE STEPS Markov Chain with Tree	37.5305	35.4198	35.9765	4.0455	3.9490	3.9944
ONE STEP Markov Chain	5.9343	5.9024	5.8913	1.5347	1.5293	1.5138
TWO STEPS Markov Chain	17.5108	16.5261	16.4863	2.7358	2.6802	2.6776
THREE STEPS Markov Chain	35.0612	33.2475	33.4315	3.9489	3.8689	3.8971
ONE STEP Multivariate Markov Chain	5.2948	4.8356	4.3823	1.3634	1.2804	1.1631
TWO STEPS Multivariate Markov Chain	16.1117	14.9719	14.0280	2.5711	2.4720	2.3658
THREE STEPS Multivariate Markov Chain	30.6684	29.1340	27.5511	3.6910	3.5719	3.4365

2.7 Final observations on Markov Chains Methods

In this chapter we discussed possible different strategies useful to predict a sequence of data in real time. The first algorithm exploits Markov Chain theory and information provided by one data sequence, the one for the speed, to predict the future states of speed. The main ingredients of this procedure are the current state of velocity and the transition probabilities matrix, which stores the probability distribution related to each state, computed using past transitions between data. Then, a different strategy for prediction has been implemented: this idea, based on the Markov Chain Theory, builds a tree with paths of different depths. The path which first reaches the desired prediction level is chosen as the optimal one. Finally, Multivariate Markov Chain theory is exploited to introduce another sequence of data which contributes to improving results obtained with the previous procedure. This methodology needs to build four different transition matrices to store transitions between states of the same sequence and inter-transitions between two different sequences. The employment of more data increases the computation time, but contributes to provide better results.

Our simulations tried to estimate two very different types of drive cycle: the first one is made up of three profiles of 95 data, repeated several times, of speed, throttle and vehicle braking, respectively. The complementarity of throttle and brake profiles was exploited to combine two sequences in a single profile. The second drive cycle instead provides a speed and a power profile, which both have 1808 data. Power and the combination between throttle and vehicle braking are useful sequences



(a) transitions between speed states

(b) transitions between speed and second sequence states

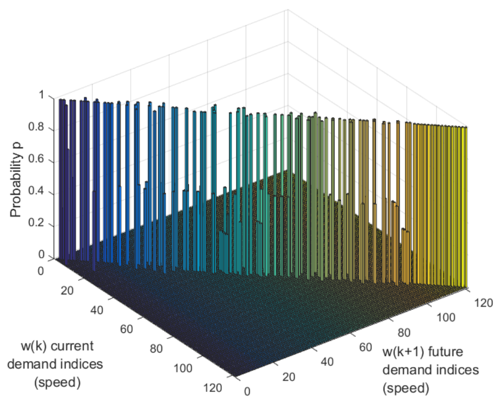
(c) transitions between second sequence and speed states

(d) transitions between power states

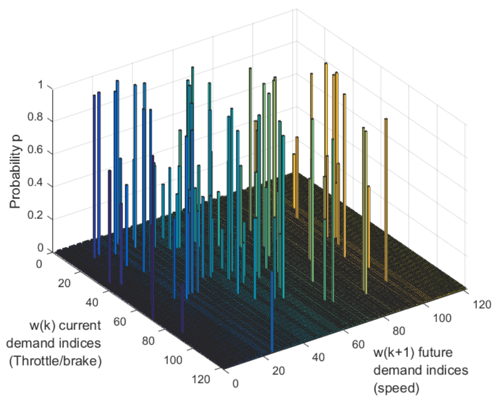
Figure 2.21: Transition matrices (states from 100 to 200) computed with WLTP drivecycle

for Multivariate Markov Chain method.

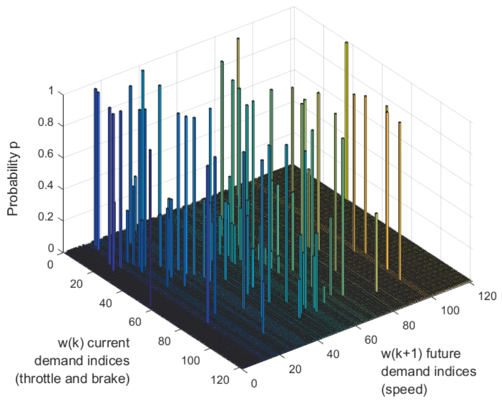
The first issue to solve was the initialization procedure of transition matrices, which can be done by using a set of data known a priori. These matrices have a key role in the performances of the algorithms, in fact the more we know about drive-cycle before starting, the more accurate is the prediction from the beginning; nevertheless, if the transition matrices are initialized with a data sequence which does not fit the drive cycle involved, the procedure takes some time to update the transition matrices and as a consequence, prediction might not be very accurate at the beginning. Of course, if no data values are available a priori, matrices must be initialized with the method discussed in subsection 2.6.3.
Transition matrices computed at the end of the simulation with WLTP drive cycle are illustrated in Fig. 2.21, while others computed with the Indian drive cycle are represented in Fig. 2.22.
Figures show that the shape of transition matrices obtained with WLTP drive cycle is completely different from the shape of the others matrices computed with Indian profiles. The main reasons are the different number of data (95 against 1804), different type of driver actions and different vehicles: WLTP represents the behaviour of a car in different environments, with low and high speed values starting from 0 to 140 [km/h], Indian drive cycle shows instead a simple repeated behaviour of a scooter, which can reach a maximum of 60 [km/h]. Consequently, we have less transitions (due to the small number of data), that are associated with higher values of probability. In the latter case, estimating correctly the next value of speed becomes easier.
The computation of the weights obtained with linear programming depends on the shape of the



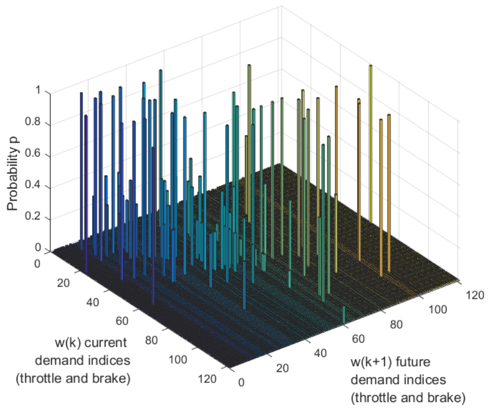
(a) transitions between speed states



(b) transitions between speed and second sequence states



(c) transitions between second sequence and speed states



(d) transitions between states belonging to second sequence

Figure 2.22: Transition matrices computed with Indian driving cycle.

transition matrices. By looking at them in Figure 2.21, probabilities stored in inter-transition matrices are very spread and have small values, that is, every inter-transition identified between power and speed data (and vice versa) is associated with a small probability value. This led to a smaller influence of power data in predicting speed of the second drive cycle: in fact, the average of the weight values λ_{11} and λ_{12}, computed by linear programming procedure, are 0.7594 and 0.2406, respectively. The same weights computed with the Indian drive-cycle are instead 0.5232 and 0.4768; the second data sequence provides a significant contribution (48%) in the computation of the probability distribution $\mathbf{p}_1(k+1)$, described by equation (2.33), while speed values contribute for almost 76% in the computation of $\mathbf{p}_1(k+1)$ for WLTP drive cycle.

$$\mathbf{p}_1(k+1) = \lambda_{11}\mathbf{T}^{11}\mathbf{p}_1(k) + \lambda_{12}\mathbf{T}^{11}\mathbf{p}_2(k) \tag{2.33}$$

The length of each interval of speed for the quantization task is another important parameter to set. Our choice was 0.5 [km/h], which is a good compromise between fast computation and prediction accuracy, and we used this value for all simulations above. We want to compute now MSE and MAE values when we choose larger sizes for the interval; smaller sizes are not really interesting because the computation time becomes higher and not suitable for real time procedures. The drive cycle considered in simulation is WLTP, because Indian drive cycle is estimated properly by the Multivariate Markov Chain procedure with quantization interval $l_q = 0.5$ [km/h].

Table 2.4: WLTP Driving cycle with 3258 data values (First 350 are used as initialization procedure). One step prediction, computed with Multivariate Markov Chain strategy, is exploited in this table.

Algorithm	Interval Length	MSE	MAE	Computation Time
Markov Chain	0.5 [km/h]	5.9343	1.5347	3.27 [s] 0.001 [s] per data
Multivariate Markov Chain	0.5 [km/h]	5.2948	1.3634	257 [s] 0.0789 [s] per data
Markov Chain	1 [km/h]	5.5142	1.5181	1.34 [s] 0.0004 [s] per data
Multivariate Markov Chain	1 [km/h]	5.7465	1.5072	234 [s] 0.07 [s] per data
Markov Chain	2 [km/h]	4.9878	1.4901	0.65 [s] 0.0002 [s] per data
Multivariate Markov Chain	2 [km/h]	5.7816	1.5640	208 [s] 0.06 [s] per data

Table 2.4 allows us to make some considerations about which algorithm works better for our purpose:

- Depending on the MSE and MAE values, the best results are obtained for $l_q = 0.5$ [km/h] as quantization interval and Multivariate Markov Chain as algorithm. The Markov Chain method, with $l_q = 2$ [km/h], has a better MSE but a worse MAE; this means the prediction computed via the Multivariate strategy is generally closer to the target profile; nevertheless a larger MSE points out that the prediction profile can have some outliers, which might be far from the targets.

- If we increase l_q, the accuracy of the prediction decays when computed with the Multivariate method, while the one obtained with simple Markov Chain improves a bit; their performances become more or less the same when the l_q is equal to 1 [km/h], and Multivariate procedure becomes useless with a $l_q = 2$ [km/h]. In this case, data are associated to a larger interval, and transitions between states are less common, that is, the probability that the future state of speed is the same as the current one becomes predominant, and other data sequences become useless. So the one step prediction profile obtained with Markov Chain method is closer to the target and it does not have outliers which increase the MSE error. Nevertheless, in this case the prediction becomes a simple delayed version of the original driving cycle, therefore the application of this strategy does not provide any particular benefit.

- The higher computation time, 0.07 [s], registered with Multivariate Markov Chain procedure, is due to linear programming computation, which takes a lot of time to provide the right weights values; nevertheless, 0.07 [s] is a time well below the time step of 1 [s], so the procedure can be exploited for real time estimation without any problem. With such a huge number of data, benefits of using Multivariate Markov Chain are not so relevant, and the Markov Chain strategy with l_q equal to 0.5 can be still used.

Let us see now the differences between predictions obtained with the two methods in different occasions. Fig. 2.23 shows the one step predictions obtained with $l_q = 0.5$ as interval length for both method. As we can see, the Multivariate method provides a prediction closer to the target with respect to the prediction obtained with just the speed profile. MSE and MAE values in tables 2.3 and 2.4 confirm the previous claim.
Fig. 2.24 illustrates the difference between the one step prediction obtained with Markov Chain strategy, computed with $l_q = 2$ [km/h], and the same prediction computed with Multivariate Strategy, with $l_q = 0.5$ [km/h]. The interval of time chosen is significant because it represents the speed at the beginning of the second repetition of the drive cycle, when transition matrices have been updated with 1804 data. The main differences between the two profiles are the following:

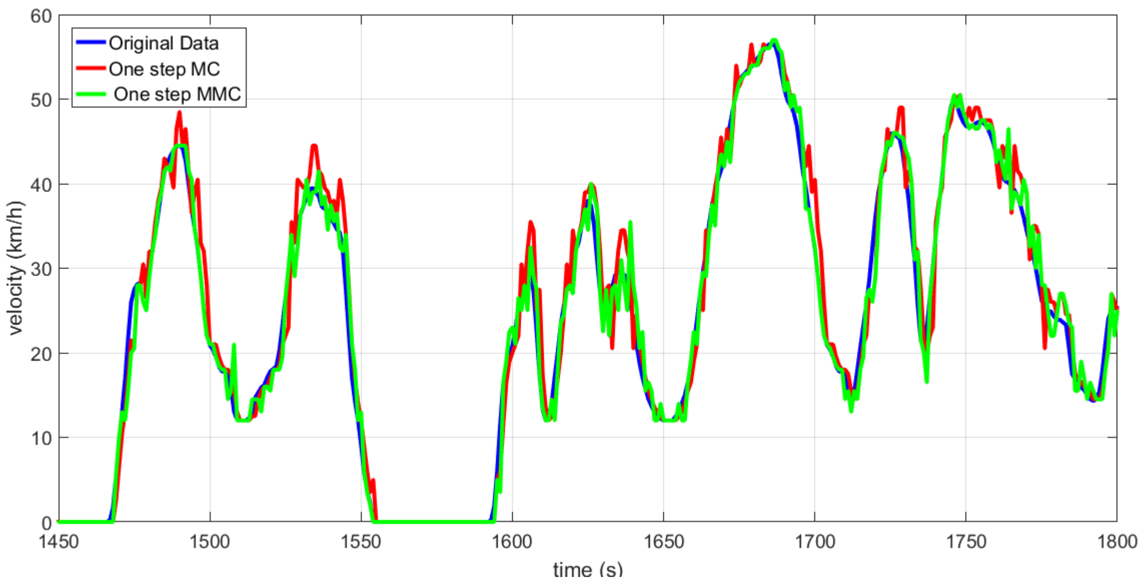


Figure 2.23: Comparison between original data of WLTP (blue), one step prediction computed with Markov Chain method (red), and one step prediction computed with multivariate strategy (green). Time interval: 1450-1800, Interval Length 0.5 [km/h]

the one obtained with Multivariate Markov Chain fits better the original profile, but it has some outliers which spoil accuracy, while the other does not have significant outliers, because a larger interval means less possibilities of transition; nevertheless the profile looks delayed with respect to the other one, therefore it does not fit precisely the original behaviour.

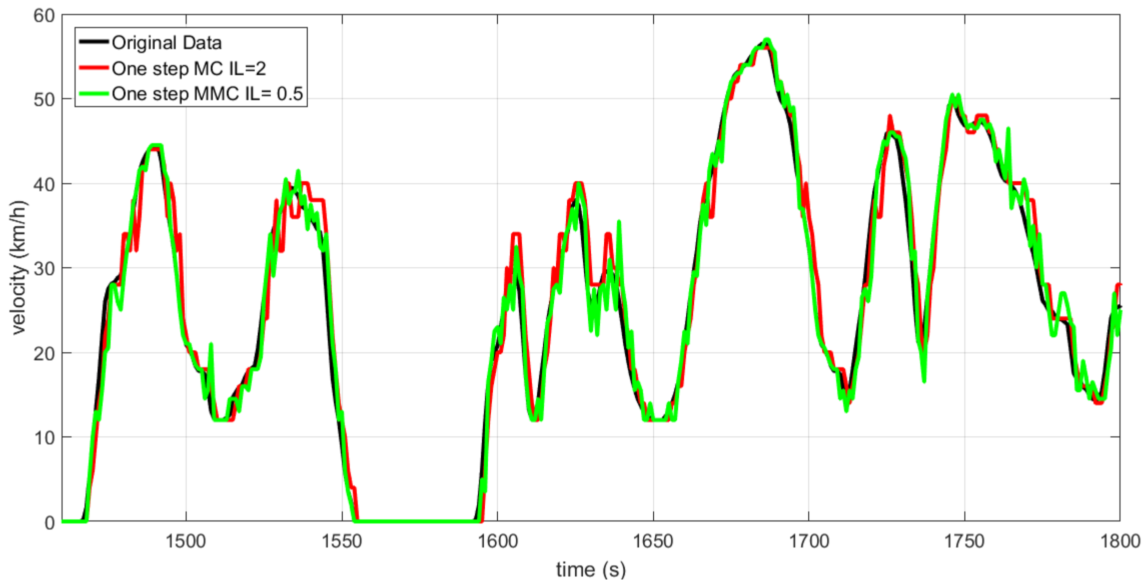


Figure 2.24: Comparison between original data of WLTP (blue), one step prediction computed with Markov Chain method with 2 as interval length (red), and one step prediction computed with multivariate strategy with 0.5 as interval length (green)

So after all these consideration, we can claim that:

- All strategies adopted can be used to compute a prediction of the desired data sequence with good results in terms of accuracy and computation time.
- Multivariate Markov Chain provides prediction results better than Markov Chain general strategy for small driving cycles like Indian one, characterized by few transitions associated with high probabilities in transition matrices;
- With long driving cycles, as WLTP, Multivariate Markov Chain procedure still works better with $l_q = 0.5$ [km/h] generally improves with the increasing in number of data, as we saw in table 2.3; simple Markov chain strategy with $l_q = 2$ [km/h] has similar errors, due to lower number of transitions available. Nevertheless, the application of the latter procedure does not provide any benefit, because the prediction tends to become a delayed version of the original driving cycle.

Finally we observed that the prediction procedure which builds the tree does not improve the quality of two or three steps prediction provided by general Markov chain method; actually, it provides worse results. Hence, we decided not to spend more time on the strategy which builds trees.

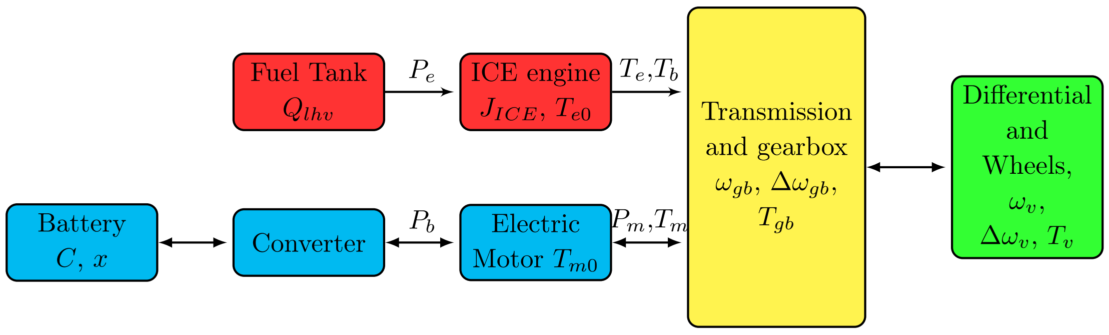
3 HEV Model

3.1 Parallel Architecture

The vehicle involved in our project [3] is characterized by a parallel architecture, which can be described by the following discrete-time, mathematical, single state model:

$$x_{k+1} = f(x_k, u_k, v_k, a_k, i_k) + x_k \qquad\qquad k = 0, 1, ..., N-1 \tag{3.1}$$

where x_k is the state of charge of the battery, u_k represents the torque split factor, the ratio between the motor and the total torque, which is the input to optimize, v_k is the speed[1], measured in $[m/s]$, a_k is the acceleration, $[m/s^2]$, and i_k is the gear number.
The powertrain of a Parallel Hybrid Electric Vehicle (P-HEV) is made up of three different branches, as illustrated in Fig. 3.1:



Fuel Tank Q_{lhv}
P_e
ICE engine J_{ICE}, T_{e0}
T_e,T_b
Transmission and gearbox ω_{gb}, $\Delta\omega_{gb}$, T_{gb}
Differential and Wheels, ω_v, $\Delta\omega_v$, T_v
Battery C, x
Converter
P_b
Electric Motor T_{m0}
P_m,T_m

Figure 3.1: Parallel Architecture

The first branch is the red one, related to the fuel tank and ICE engine; then battery, converter and electric motor realize the second branch in blue, while the third in green is made up of differential and wheels, which have a radius $r = 0.3$ [m]. Finally, the yellow box, where all other branches are connected, represents the transmission and the gearbox.

3.2 Vehicle

The vehicle is characterized by a mass m_v of 1800 [kg], which is the result of the summation of four components:

- vehicle mass m_0;

[1]In the previous chapter, speed was measured in [km/h]; nevertheless this particular model requires a value measured in [m/s], therefore when a value of speed is acquired in [km/h], it has to be converted in [m/s] by dividing the value by 3.6

- combustion engine mass m_{ICE};
- motor mass m_{em};
- battery mass m_b.

As we saw in equation (3.1), the model receives as input three quantities, speed v, acceleration a and gear i, which contribute in computing the following quantities:

- wheel rotational speed $\omega_v = v/r$;
- wheel rotational acceleration $\Delta\omega_v = a/r$;
- wheel torque T_v.

The wheel torque is the product of wheel radius r with a combination of three forces: the rolling friction F_f, estimated in 144 [N], the aerodynamic drag force $F_a = \rho v^2$, where $\rho = 0.48$ $[Ns^2/m^2]$ is the aerodynamic coefficient, and the inertial force $F_i = m_v a$. So:

$$T_v = (F_f + F_a + F_i)r. \tag{3.2}$$

This vehicle model does not consider wheel slip.

3.3 Transmission and GearBox

The gearbox is a six gear manual transmission associated to fixed gear ratios, each one characterized by a constant efficiency $\eta_{gb} = 0.95$. Table 3.1 stores the six possible values:

Gear i	1	2	3	4	5	6
Gear ratio $\gamma(i)$	17	9.6	6.3	4.6	3.7	3.5

Table 3.1: Gear ratio

The inputs of the gearbox are the wheel speed ω_v, wheel acceleration $\Delta\omega_v$, wheel torque T_v, and gear number i. These values are useful to compute the following quantities:

- crankshaft rotational speed $\omega_{gb} = \gamma(i)\omega_v$ [rad/s];
- crankshaft rotational acceleration $\Delta\omega_{gb} = \gamma(i)\Delta\omega_v$ [rad/s^2];
- crankshaft torque T_{gb} [Nm]:

$$T_{gb} = \begin{cases} \frac{T_v}{\gamma(i)\eta_{gb}} & T_v > 0 \\ \\ \frac{T_v\eta_{gb}}{\gamma(i)} & T_v \leq 0 \end{cases} \tag{3.3}$$

The gearbox is characterized by a speed dependent shifting strategy, where gear values are associated to a sub-interval of speed. An up-shift happens when the speed value exceeds the upper limits of the interval, while a down-shift happens when the lower limit is exceeded. Of course, these intervals depend on the different type of vehicle involved. No energy losses are then considered during the gear shifting procedure. Further explanations are provided in the following chapter.
The total demand of torque T_{tot} is the result of the summation of two or three terms, depending on the value of u:

$$T_{tot} = \begin{cases} T_{m0} + T_{gb} & u = 1 \\ T_{e0} + T_{m0} + T_{gb} & \text{otherwise} \end{cases} \tag{3.4}$$

where:

- T_{m0} is the electric motor drag torque, [Nm]: $T_{m0} = I_m \Delta\omega_{gb}$, where $I_m = 0.03$ $[kg\frac{m}{s^2}]$ is the motor inertia;

- T_{e0} is the engine drag torque, [Nm]: $T_{e0} = I_e \Delta\omega_{gb} + T_{diss}(\omega_{gb})$, where $I_e = 0.14$ $[kg\frac{m}{s^2}]$ is the engine inertia, while $T_{diss}(\omega_{gb})$ is the dissipation torque, which is a function of the crankshaft rotational speed.

$T_{diss}(\omega_{gb})$ is computed through a linear interpolation based on current value of ω_{gb}. Vectors involved are stored in table 3.2: vector ω contains the sample points of the crankshaft speed, while T_{diss} contains the corresponding values of dissipated torque.

T_{diss} list	22.24	22.24	22.47	23.37	24.65	27.27	29.85	29.77
ω vector	112	168	224	280	336	392	447	503

Table 3.2: T_{diss}

The torque provided by the engine is the following:

$$T_e = (1-u)T_{tot}; \qquad \omega_{gb} > 0 \text{ and } T_{tot} > 0; \tag{3.5}$$

$$T_b = (1-u)T_{tot}; \qquad \omega_{gb} > 0 \text{ and } T_{tot} \leq 0; \tag{3.6}$$

where T_e is the acceleration torque, while T_b the brake one; finally the torque provided by the electric motor is:

$$T_m = uT_{tot} \qquad \omega_{gb} > 0. \tag{3.7}$$

The required torque is provided by the electric motor and the ICE engine. This demand is ruled by a control input $u = \frac{T_m}{T_{tot}}$, the torque split, computed as the ratio between the motor and the total torque, that is a continuous variable which can take values in the interval $[-L, 1]$. It determines how to split the torque demand between the two sources:

- negative values $[-L, 0)$ mean that more torque than is demanded is provided by the internal combustion engine to recharge the battery;

- $u = 0$ occurs when torque is only provided by the internal combustion engine ($T_m = 0$);

- positive values $(0, 1)$ are chosen when torque has to be provided from both internal combustion engine and electric motor (u establishes the weight of the contribution of the electric motor, while 1-u is the weight related to ICE);

- finally, with $u = 1$ all torque is provided by the electric motor ($T_e = 0$).

The lower value $-L$ can be defined according to the engine size: in our simulations we will consider $L = 1$ or $L = 2$. More details are explained in Appendix A.
Acceleration phases contribute in discharging the battery, while deceleration in recovering its state of charge; when the vehicle is not moving, the battery level remains the same, because the torque required is 0.
If $T_e > 0$ and contemporary $\omega_{gb} < 0$, or $T_{tot} < 0$ and $T_m > 0$, the current input u is infeasible and could damage the overall structure.

3.4 Internal Combustion Engine (ICE)

Let us now define how the combustion branch works; the first thing to determine is the engine efficiency $\eta_{ICE} = \eta_{ICE}(\omega_{gb})$, which is computed as a function of the crankshaft speed ω_{gb} through a linear interpolation procedure. ω are again the samples, while η_{ICE} (stored in Table 3.3) are the corresponding values of the efficiency.

η_{ICE} list	0.423	0.420	0.446	0.445	0.446	0.445	0.440	0.423
T_{max} list	129	163	190	194	197	199	198	196
ω vector	112	168	224	280	336	392	447	503

Table 3.3: η_{ICE} and T_{max}

Once the efficiency is computed, we can get the instantaneous fuel consumption, which is a function of power and efficiency:

$$\dot{m}_{fuel} = \frac{T_e \omega_{gb}}{\eta_{ICE} Q_{lhv}} \tag{3.8}$$

where $Q_{lhv} = 42500000$ [J/kg] is the lower heating value of gasoline. The power of the fuel consumption is then the following:

$$P_e = \dot{m}_{fuel} Q_{lhv}. \tag{3.9}$$

Finally, the torque T_e computed in equation (3.6), must not be higher than the maximum allowed $Te_{max} = Te_{max}(\omega_{gb})$ which is computed with another linear interpolation, where the sample values are stored in Table 3.3.

3.5 Motor and Battery Branch

Now we introduce the equations which rule the behaviour of the electric motor branch. We start computing the electric power consumption of the battery:

$$P_m = \begin{cases} \omega_{gb} T_m \eta_m & T_m < 0; \\ \frac{\omega_{gb} T_m}{\eta_m} & T_m \geq 0; \end{cases} \tag{3.10}$$

where η_m is the electric motor efficiency, which is a function of the motor rotational speed ω_m and the electric motor torque T_m. The efficiency is computed through a 2D linear interpolation, where $\omega_m \in [0, 600]$ [RPM] and $T_m \in [0, 160]$ [Nm] are coordinate samples, while η_m is an efficiency map with the shape of a matrix, which contains the corresponding function values at each sample point. The absolute value of T_m, computed in equation (3.7), must not exceed the maximum value for the motor torque $T_{m_{max}} = T_{m_{max}}(\omega_m)$, which is a function of the motor rotational speed ω_m, and is computed with a linear interpolation:

$T_{m_{max}}$ [Nm]	130	130	130	110	105	95	80	70	59	50	40	32	28
ω_m [RPM]	0	50	100	150	200	250	300	350	400	450	500	550	600

Table 3.4: $T_{m_{max}}$ Motor Torque table of samples

Now we introduce the equations that describe the behaviour of the battery. The main component of the battery is the internal resistance

$$r_{int} = r_{int}(x, P_m),$$

a function of the state of charge x and the motor power P_m. The motor power is important to determine the current I_b of the battery, too. Table 3.5 stores the sample value for the resistance during the discharging and charging phases of the battery and the corresponding SOC values. If $P_m > 0$, the vehicle is accelerating while battery is discharging, so the value of samples for the linear interpolation with the SOC are stored in vector $\mathbf{R}_{discharge}$; otherwise, if $P_m \leq 0$, we have to consider the vector $\mathbf{R}_{charge}$, because the battery is charging.
The efficiency of the battery η_b is:

$$\eta_b = \begin{cases} 1 & P_m > 0; \\ 0.9 & P_m \leq 0. \end{cases} \tag{3.11}$$

$R_{discharge}$ [ohm]	1.75	0.60	0.40	0.30	0.30	0.30
R_{charge} [ohm]	0.35	0.50	0.85	1.00	2.00	5.00
SOC	0	0.2	0.4	0.6	0.8	1
V_{oc} [Volt]	230	240	245	250	255	257

Table 3.5: R and V_{oc}, function of battery state-of-charge

The open circuit voltage $v_{oc} = v_{oc}(x)$ of the battery is a function of the state of charge and is computed with linear interpolation, too. Vector V_{oc} contains the sample for linear interpolation; it is stored in the previous table.
Once the internal resistance, the efficiency and the voltage of the battery are computed, we can calculate the battery current I_b. The motor power P_m is obtained from to the following relation:

$$-P_m = r_{int} I_b^2 + v_{oc} I_b \tag{3.12}$$

So, the current I_b can be found by solving the quadratic equation $r_{int} I_b^2 + v_{oc} I_b + P_m = 0$:

$$I_b = \eta_b \frac{v_{oc} - \sqrt{v_{oc}^2 - 4 r_{int} P_m}}{2 r_{int}} \tag{3.13}$$

The battery current must not exceed the maximum discharging current[2] of 225 [A] and the maximum charging current, 200 [A], not to damage the battery.
Furthermore, v_{oc}^2 must be bigger than $4 r_{int} P_m$. Finally, the state of charge of the battery is updated as follows:

$$x_{k+1} = -\frac{I_b}{3600C} + x_k, \tag{3.14}$$

where $C = 6$ [Ah] is the battery capacity. The battery power consumption is given by:

$$P_b = I_b v_{oc}. \tag{3.15}$$

The scheme of the battery circuit is the following:

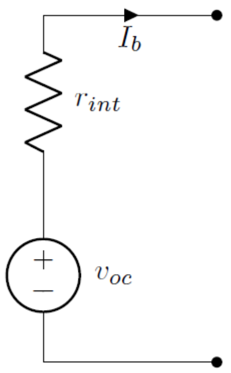


Figure 3.2: Battery circuit

[2]absolute value

4

Dynamic Programming

Dynamic Programming is a powerful method to solve complex problems, because it manages to split them into simpler sub-problems which may be recursively solved. This procedure usually deals with energy management problems, where the goal is to find optimal control inputs which minimize a mathematical cost function. In this chapter, we are going to exploit this strategy in order to minimize the fuel consumption of the hybrid electric vehicle, described by the dynamical model in Chapter 3, in different journeys. Speed values predicted by Markov Chains Algorithms, illustrated in Chapter 2, are here used as input for the real time procedure.

4.1 Dynamic Programming algorithm

According to [9] and [14], a generic continuous optimization problems has the following form:

$$\begin{aligned} &\min_{u(t)\in\mathcal{U}} J(u(t)) \\ \textbf{s.t.:}\ &\dot{x}(t) = F(x(t), u(t), t) \\ &x(0) = x_0, \quad x(t_f) \in [x_{f,\min}, x_{f,\max}] \\ &x(t_f) \in \mathcal{X}(t) \subset \mathcal{R}^n, \quad \mathcal{U} \subset \mathcal{R}^m, \end{aligned}$$

where $J(u(t))$ is a mathematical cost function, $u(t)$ is the decision variable, and U is its set, while $\dot{x}(t)$ is the continuous dynamical model. The cost function can be written as follows:

$$J(u(t)) = G(x(t_f)) + \int_0^{t_f} G(x(t), u(t), t)dt. \tag{4.1}$$

$G(x(t_f))$ is the final cost at time t_f, while the second term is the cost-to-go of applying the control input u to state x. Let us now discretize in time the procedure, so the continuous model F becomes:

$$x_{k+1} = f(x_k, u_k) \qquad k = 0...N-1, \tag{4.2}$$

where k is the time index, x_k is the state at time k, u_k is the control policy that has to be applied at time k, N is the time horizon and f is a function that describes the system and in particular how the state is updated. The continuous total cost (4.1) becomes a discrete additive cost, i.e, the cost obtained at time k, denoted by $g(x_k, u_k)$, accumulates over time:

$$J(u_k) = g_N(x_N) + \phi_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k). \tag{4.3}$$

$g_N(x_N)$ is the terminal cost. The penalty function $\phi_N(x_N)$, can be used to enforce the constraint on the final state.
The dynamic programming algorithm is based on Bellman's principle of optimality [14]:

Definition 4.1 Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. In other words, let us consider the optimal control policy $\pi^* = \{\mu_0^*, \mu_1^*, \mu_2^*, \dots, \mu_{N-1}^*\}$, where μ_k is a function which maps the state x_k in the input u_k. Then we assume that, when π^* is used, a given state x_l is reached at time l with positive probability. If we consider the sub-problem starting from x_l, which aims to minimize the following cost-to-go from time l to final time N,

$$J(u_k) = g_N(x_N) + \phi_N(x_N) + \sum_{k=l}^{N-1} g_k(x_k, u_k)$$

the truncated policy $\pi^* = \{\mu_l^*, \mu_{l+1}^*, \mu_{l+2}^*, \dots, \mu_{l+N-1}^*\}$ is the optimal one for the sub interval $[l, N]$.

Let us consider x_0 as the initial state. The discretized cost, given a control policy $\pi = \{\mu_0, \mu_1, \mu_2, \dots, \mu_{N-1}\}$, is the following:

$$J_\pi(x_0) = g_N(x_N) + \phi_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k)) + \phi_k(x_k) \text{ for } k = 0, ..., N-1.$$

The optimal control policy π^* minimizes J_π, where Π is the set of all admissible policies:

$$J^*(x_0) = \min_{\pi \in \Pi} J_\pi(x_0).$$

Let us define x_k^i, the discretized state variable at the node with time index k and state index i; then, the cost function $J_k(x_k^i)$ is computed by the procedure at every node in the discretized state-time space by proceeding *backward* in time, as explained in Algorithm 3.

Algorithm 3 Deterministic dynamic programming algorithm

Initialization phase: In this phase only the final cost is computed:

$$J_N(x^i) = g_N(x^i) + \phi_N(x^i). \tag{4.4}$$

Backward phase: here the intermediate computation steps are performed, starting from $k = N-1$, and going backward to $k = 0$:

$$J_k(x^i) = \min_{u_k \in \mathcal{U}_k} \{g_k(x_k^i, u_k) + \phi_k(x^i) + J_{k+1}(f_k(x_k^i, u_k))\}. \tag{4.5}$$

The definition above is not sufficient for our real time purposes; in chapter 2 we modelled speed as a stochastic process w_k, which have to be considered as an input in the following discrete time model:

$$x_{k+1} = f(x_k, u_k, w_k) \qquad k = 0, ..., N-1. \tag{4.6}$$

The cost function, because of the presence of w_k, is generally a random variable and cannot be meaningfully optimized. Therefore we need to reformulate the control problem in order to optimize the expected cost, where the expectation is taken with respect to the probability distribution of the random variable involved:

$$J(u_k) = \mathbb{E}\left[g_N(x_N) + \phi_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) \right]. \tag{4.7}$$

Stochastic dynamic programming works as Algorithm 3, nevertheless equation (4.5) has to be replaced by the following equation:

$$J_k(x^i) = \min_{u_k \in \mathcal{U}_k} \mathbb{E}[g_k(x_k^i, u_k, w_k) + \phi_k(x^i) + J_{k+1}(f_k(x_k^i, u_k, w_k))], \tag{4.8}$$

where the expectation is computed with respect to the probability distribution of w_k. Moreover, $u_k^* = \mu_k^*(x_k)$ is the optimal control policy at time k if it minimizes the right hand side of equations (4.5) and (4.8) for each x_k^i.

4.2 DPM Matlab function

The so-called *dpm* function, provided in [9], solves the discretized version of the control problem discussed in section 4.1. It requires as input the mathematical model of the vehicle (*fun*), a problem structure (*prb*) defined by the user, a grid (*grd*) which contains informations and constraints on the input and the state, and an option structure. The output of the algorithm is an optimal control signal map containing the results of the dynamic programming algorithm. This map is then exploited to find the optimal state and cost trajectory through a forward simulation of the model, starting from the given initial state x_0. The control signal in the map is provided only for discrete points in the state-space grid, therefore it must be interpolated when actual state does not coincide with the points in the state grid.
Problem structure contains information about the time step T_s (in our case $T_s = 1$ [s]), the length of the problem N, that counts how many time steps the procedure must consider, and a vector $\mathbf{w}$ of length N containing the input profiles for the model. If we consider the model described in chapter 3, vector $\mathbf{w}$ needs to contain three different profiles of length N describing speed, acceleration and gear.
The second input discussed, (*grd*), is fundamental to realize a grid structure. The latter is made up of cell arrays, associated with possible discrete values about the state x and the input u. This structure has the following fields:

- the number of grid points $N_x\{\cdot\}$ and $N_u\{\cdot\}$, for state and input, respectively;
- the lower and the upper limits for each state, $X_n\{\cdot\}.lo$, $X_n\{\cdot\}.hi$, respectively;
- the lower and the upper limits for the final state constraints, $X_N\{\cdot\}.lo$, $X_N\{\cdot\}.hi$, respectively;
- the initial value for the state $X_0\{\cdot\}$, only necessary for forward simulation;
- the lower and the upper limits for each input, $U_n\{\cdot\}.lo$, $U_n\{\cdot\}.hi$ respectively.

Finally, the option structure defines how to use the algorithm; an important option is the boundary line method, introduced in [15] which allows to increase the accuracy of dynamic programming procedure. More details are given in Appendix B.

4.3 Application of the procedure with given drive cycles

Now we see how the deterministic dynamic programming procedure works with the vehicle model described in chapter 3, which can be summarized by the following equation:

$$x_{k+1} = f(x_k, u_k, v_k, a_k, i_k) + x_k. \tag{4.9}$$

With the deterministic approach, we assume to know in advance the speed, acceleration and gear profile v_k, a_k, i_k and, as a consequence, the number of data involved in our simulations. So, we can write:

$$x_{k+1} = f(x_k, u_k) + x_k \qquad k = 0, 1, ..., N-1 \tag{4.10}$$

Our aim is to optimize the total fuel mass consumed by the vehicle during the driving cycle, therefore our cost function is the sum of all fuel masses contributions at each time step:

$$J = \sum_{k=0}^{N-1} \Delta m_f(u_k, k) T_s, \tag{4.11}$$

where T_s is equal to one. The optimization problem becomes:

$$\min_{u_k \in \mathcal{U}_k} \sum_{k=0}^{N-1} \Delta m_f(u_k, k) \tag{4.12}$$

$$\textbf{s.t.:} \tag{4.13}$$

$$x_{k+1} = f_k(x_k, u_k) + x_k \tag{4.14}$$

$$x_0 = x_S \tag{4.15}$$

$$x_N = x_0 \tag{4.16}$$

$$x_k \in [x_{min}, x_{max}] \tag{4.17}$$

The state of charge x_k assumes a value between 0 and 1; x_S is the initial state of charge, while x_N is the desired final state of charge, which usually has the same value set as the initial one. Moreover the state of charge must be constrained in the interval $[x_{min}, x_{max}]$, where x_{min} represents the minimum possible value, and x_{max} the maximum value. These values are set according to battery specifications; a state of charge which does not respect these constraints might damage the battery and the whole system.
For our first simulation we set $x_S = 0.65$ as the initial state, and $x_{min} = 0.4$ $x_{max} = 0.7$. The final state of charge has the same value as the initial one, that is, whatever path is covered by the vehicle and whatever actions are taken by the driver, the battery state of charge must be the same at the beginning and at the end.
The algorithm requires to set also the number of grid points in the state grid N_x and in the input grid N_u: for the first simulation we set $N_x = 61$ and $N_u = 21$. Larger values lead to better accuracy but higher computation time. Finally u_k can assume values in the interval $[-1, 1]$.
The first driving cycle considered in our simulation is called Japanese 10-15, and it contains three different profiles, for speed (Figure 4.1), acceleration and gear.

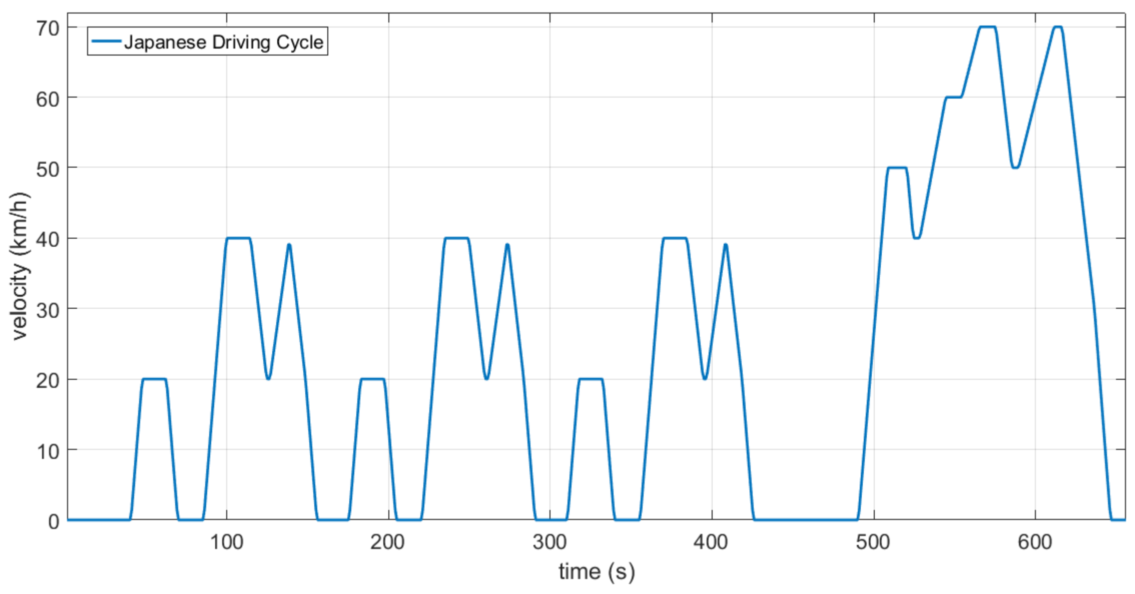


Figure 4.1: Japanese 10-15 speed profile; maximum speed value is 70 [Km/h]

Figure 4.2 illustrates the trajectory of the state of charge: we can see that the final constraint is satisfied, and the state of charge is always within the desired interval.

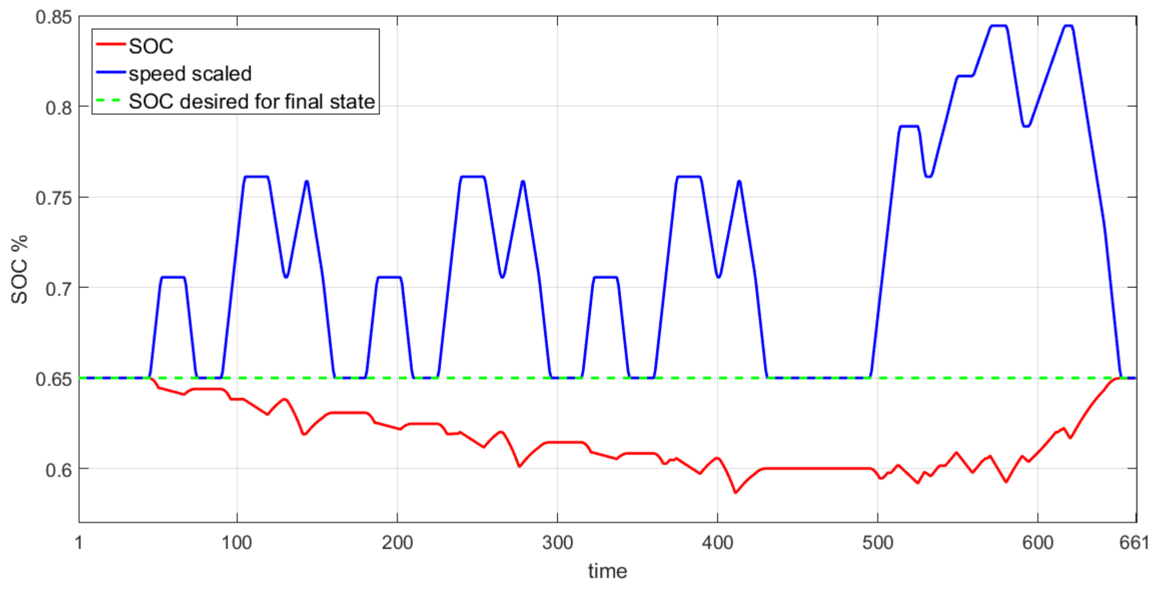


Figure 4.2: Trajectory of the SOC (red) computed with deterministic dynamic programming applied to Japanese driving cycle (blue). Initial and final SOC are equal to 0.65.

Let us now introduce in the procedure the driving cycles described in Chapter 2: repeated Indian and WLTP.

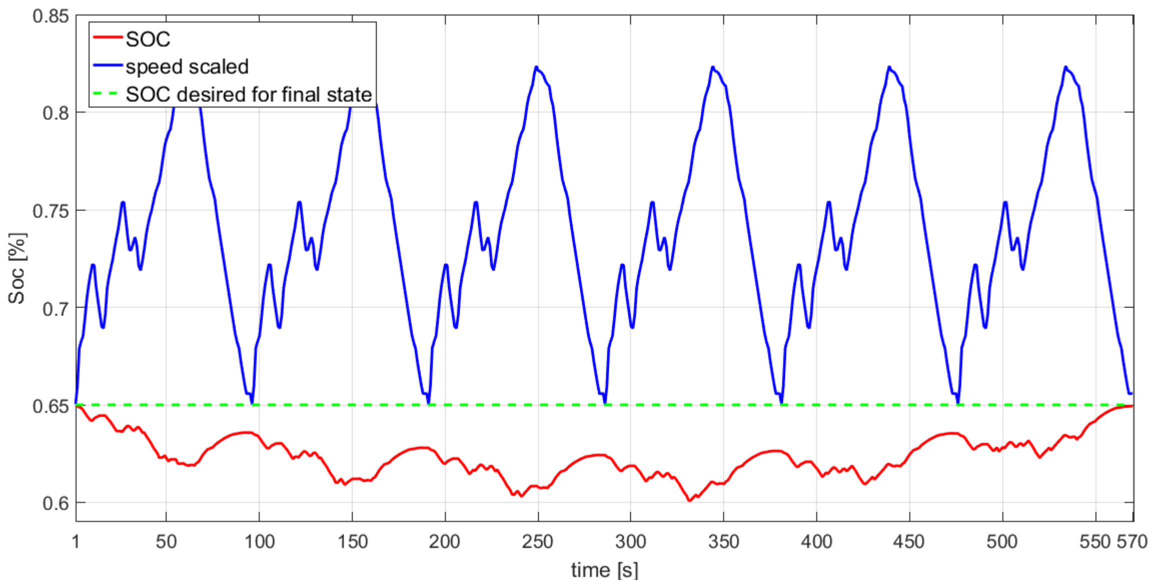


Figure 4.3: Trajectory of the SOC (red) computed with deterministic dynamic programming applied to repeated Indian driving cycle (blue). Initial and final SOC are equal to 0.65.

The behaviour of the state-of-charge SOC is similar in all simulations: during the initial part of the journey, the battery is discharged to attain minimal fuel consumption target, while the second part is exploited to restore the same state of charge value as the beginning. Of course, the state of charge computed with WLTP drive cycle reaches lower values, due to higher values of speed.

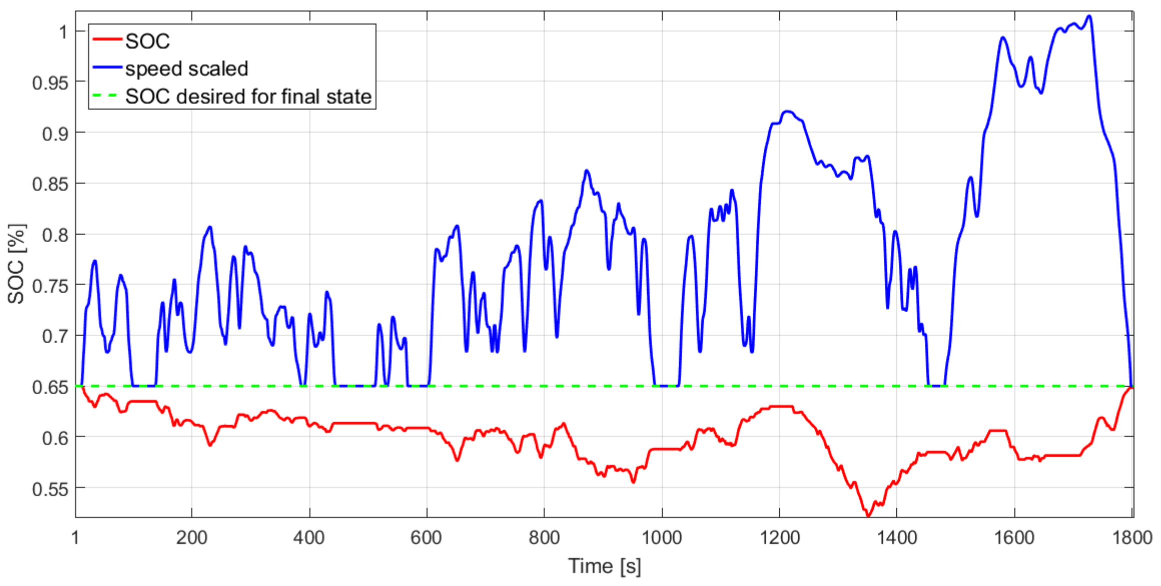


Figure 4.4: Trajectory of the SOC(red) computed with deterministic dynamic programming applied to WLTP driving cycle (blue). Initial and final SOC are still equal to 0.65.

The torque splits associated to the three driving cycles are illustrated in Figures 4.5, 4.6, 4.7. We can easily see that negative values for the torque split are principally applied during the last part of the driving cycle: during the first part the battery branch is mostly exploited to produce the desired torque. On the contrary, during the final part of the journey, negative values are applied in order to attain the final state battery constraint.

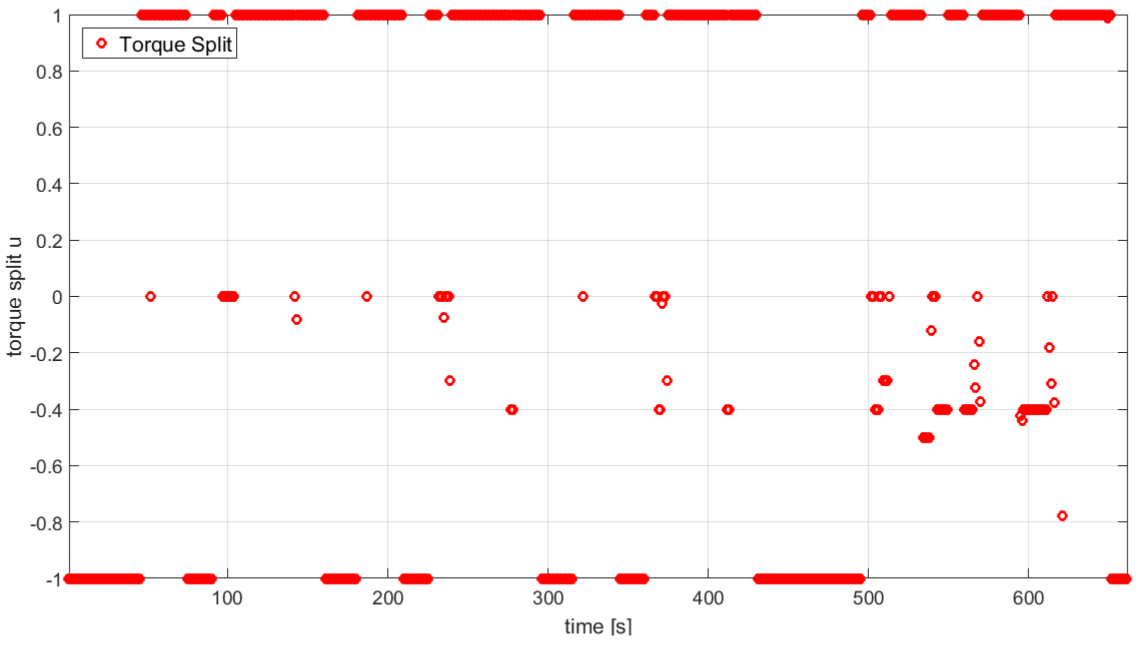


Figure 4.5: Torque split computed with deterministic dynamic programming applied to Japanese driving cycle.

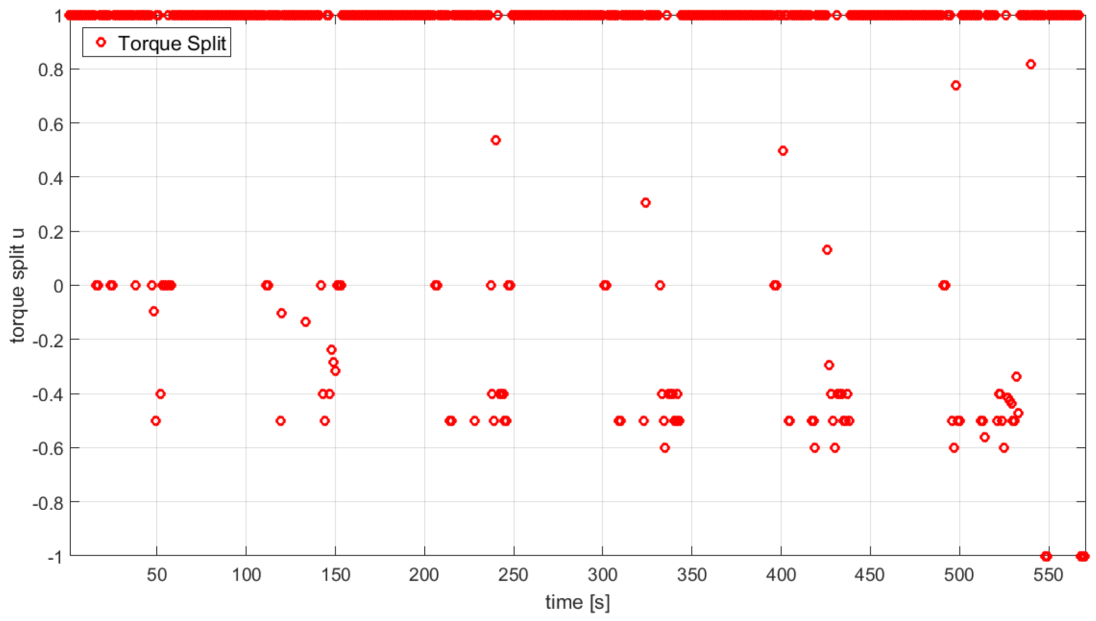


Figure 4.6: Torque split computed with deterministic dynamic programming applied to Indian driving cycle.

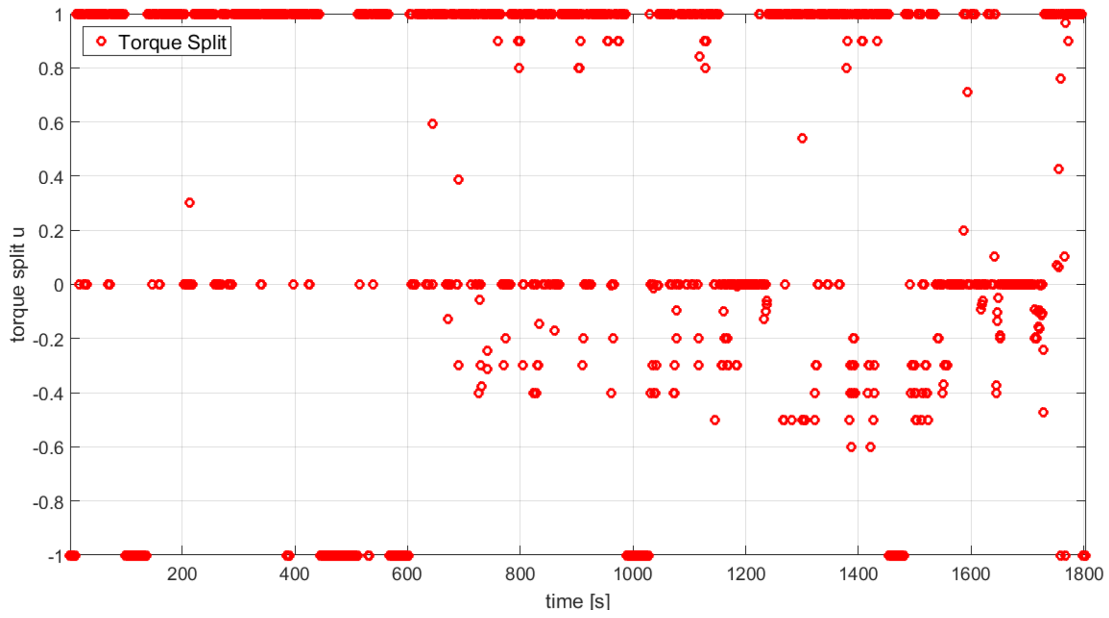


Figure 4.7: Torque split computed with deterministic dynamic programming applied to WLTP driving cycle.

Table 4.1 stores the details of the three different simulations computed, with particular attention to fuel consumption value. Of course, WLTP consumes more fuel, because of the larger data profiles and the wide variety of driver actions.

Table 4.1: This table stores the fuel consumed during each journey after the application of deterministic dynamic programming on the whole horizon of data

Drive cycle	Number of Data	Fuel Consumption [Kg]	Horizon length
Japan	661	0.1205 [Kg]	661
Indian	571	0.1180 [Kg]	571
WLTP	1804	0.8702 [Kg]	1804

Finally, Figures 4.8 and 4.9 illustrate the comparison between the engine torque, the motor torque and the total torque computed respectively with Japanese and WLTP driving cycles.

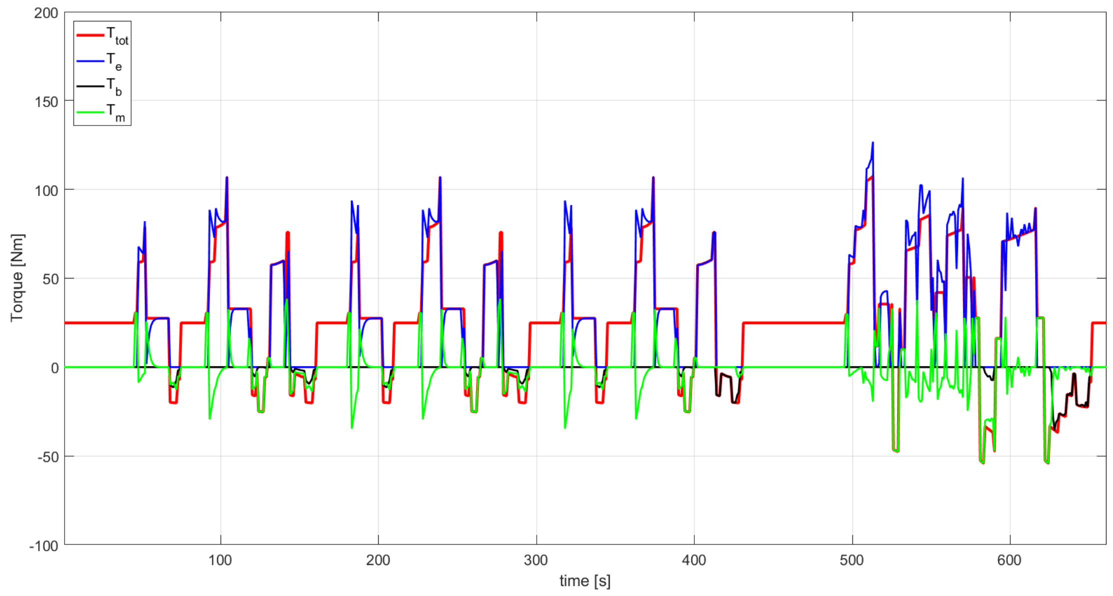


Figure 4.8: Comparison between total torque T_{tot}, engine torque T_e, T_b, and Motor Torque T_m computed with deterministic dynamic programming applied to Japanese driving cycle.

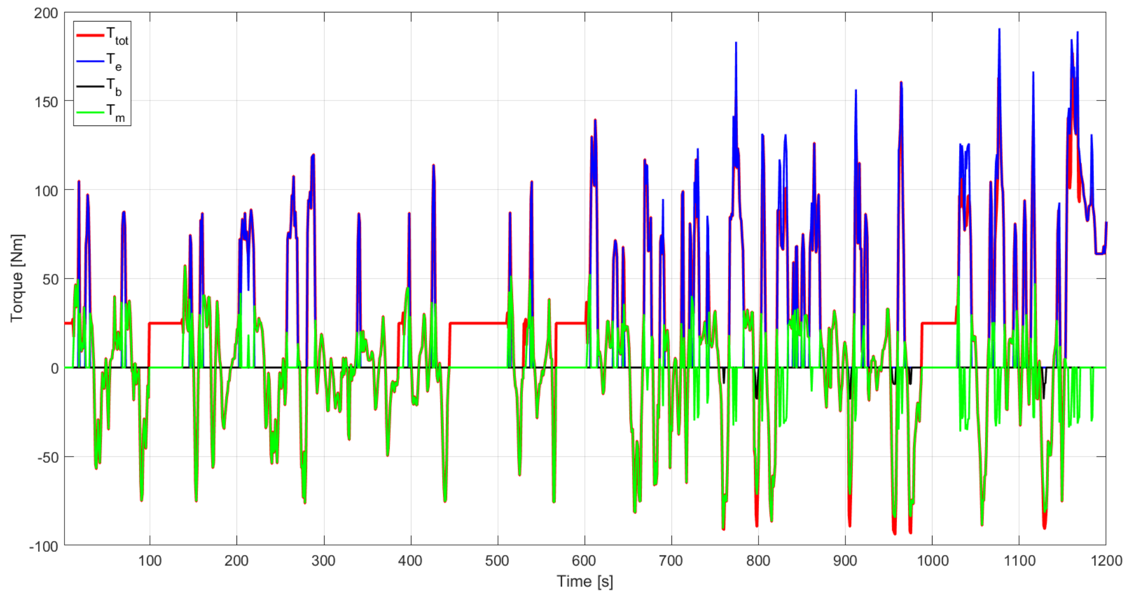


Figure 4.9: Comparison between total torque T_{tot}, engine torque T_e, T_b, and Motor Torque T_m computed with deterministic dynamic programming applied to WLTP driving cycle. Interval: 1-1200

We can see that the engine branch (T_e positive values, T_b negative) is mainly exploited when the total torque demand is positive, that is, the vehicle is accelerating. On the contrary, when the vehicle is braking, motor branch supply the total torque required to drive the vehicle. Motor branch may also supply torque in order to help engine branch in fulfilling the torque demand in accelerating phases. If necessary, engine branch may also supply torque when the vehicle is breaking. With Japanese driving cycles the latter possibility is exploited, with WLTP instead is quite rare.

4.4 Real Time Algorithm and Stochastic Dynamic Programming

Dynamic programming algorithm determines an optimal policy for the torque split between two torque sources (engine and motor) so that the fuel consumption is minimized. The knowledge of the whole finite horizon allows the procedure to choose the optimal control policy which reaches the global minimum for the cost function along the whole journey. In the meantime, the algorithm manages to satisfy vehicle driveability and to maintain battery SOC within the given constraints. Now let us see what happens with a real time procedure, when speed, acceleration and gear need to be estimated at every instant, without knowing how long the journey is, and, therefore, without constraints on the final state of charge.
We now start by formulating a proper infinite horizon problem, where the system dynamics and the cost are time-invariant and we cannot define a final time for the journey or a terminal constraint for the state of charge. A key benefit of this strategy is that also the generated control policy is time-invariant and thus the whole procedure can be easily implemented in a real vehicle.
Markov Chain strategies are here exploited to predict future speed values: at every instant, the vehicle acquires the current speed through some sensors and estimates future next sample of speed (sampling time is one second). Acceleration and gear, which are functions of the predicted speed, are computed according to the following rules. The former is generally calculated through this relation between two speed values:

$$a = \frac{v(t_2) - v(t_1)}{t_2 - t_1}.$$

where $v(t_2)$ is the speed at time t_2, while $v(t_1)$ is the velocity at time t_1. In our case, we have the predicted future speed $\tilde{v}(k+1)$ and the current speed value $v(k)$, and the difference of time between the two values is one second, so the relation above would become $a_{k+1} = (\tilde{v}(k+1) - v(k))/(1[s])$. Nevertheless, this relation involves a real value and an estimated one, which is characterized by quantization and estimation errors. The latter cause issues when computing the acceleration: for example, nonnegative values for the acceleration might be obtained when the vehicle is decelerating. because of these, the acceleration vector stores oscillations between positive and negative values which do not allow the procedure to discriminate properly the two phases, so the optimization task could fail.
We fixed this issue by replacing the current value of speed $v(k)$ with $\bar{v}(k)$, the result of a mean filter which computes the arithmetic mean between the current prediction value $w(k+1)$, and the two previous predictions $w(k)$ and $w(k-1)$:

$$\tilde{a}_{k+1} = \frac{w(k+1) - \bar{v}(k)}{1[s]}$$

In Chapter 3, we said that gear i is computed according to a gear shifting strategy, given by a speed dependent shifting policy. This simple strategy splits the whole interval of speed in six different sub-intervals, each associated to a particular gear ratio: the higher is the speed, the lower is the gear ratio associated. Intervals have to be chosen according to the typology of the vehicle and of the associated driving cycles. The following scheme shows some examples of velocity intervals associated with a proper gear value:

$$\tilde{i}_{k+1} = \begin{cases} 1 & w(k+1) \in [0,5] & [m/s] \\ 2 & w(k+1) \in [5,10] & [m/s] \\ 3 & w(k+1) \in (10,18] & [m/s] \\ 4 & w(k+1) \in (18,25] & [m/s] \\ 5 & w(k+1) \in (25,30] & [m/s] \\ 6 & w(k+1) \in (30,\text{MAX}] & [m/s] \end{cases} \tag{4.18}$$

When the vehicle speed exceeds the speed of the current interval there is an up-shift; on the contrary, when the speed is lower, then a down-shift happens.

Once future speed, acceleration and gear value are obtained, the optimization problem with dynamic programming function is reformulated so that it can be applied to a window of one second, in order to calculate the optimal policy of powertrain energy management which represents the local minimum for the fuel consumption in that small window. In the meanwhile, the vehicle follows the predicted speed. This procedure is repeated every instant until the vehicle journey is complete in a receding horizon manner.
The optimization problem applied to every small window is the following one for each instant k:

$$\min_{u_k \in \mathcal{U}_k} E[\Delta m_f(u_k, w_k, k)] \tag{4.19}$$

$$\textbf{s.t.:} \tag{4.20}$$

$$x_{k+1} = f_k(x_k, u_k, w_k) + x_k \tag{4.21}$$

$$x_0 = x_k \tag{4.22}$$

$$x_k, x_{k+1} \in [x_{min}, x_{max}] \tag{4.23}$$

It differs from the original problem by small but significant details:

- a_k and i_k, respectively acceleration and gear, depend on the speed value predicted with Markov Chain Procedure; because of this, they are not included in the model (4.21);
- the cost function that has to be minimized is the expectation of the instantaneous fuel consumption at time instant k; in fact process $\mathbf{w}$ of predicted speed, modelled as a Markov Chain, is stochastic.
- the initial state-of-charge x_0 must be initialized with the value of the current state-of-charge, computed by applying the dynamic programming procedure to the previous interval;
- the summation has been replaced because in this case the length of the horizon applied to the procedure is one second. With a larger horizon and more samples, it has to be re-introduced;
- the final state of charge is constrained only by the lower and upper limits; the limited horizon does not allow to define a desired final state of charge.

The application of such an optimization problem to the Japanese drive cycle does not provide useful results, as we can see in Fig. 4.10, where the trajectory of the state of charge computed in real time is shown.
The value of the state of charge is updated in real time with the previous method: it starts from the given initial value, 0.65, then gradually decreases, until it reaches the lower value available for the SOC; finally, it remains constant. The decreasing in SOC level is due to the massive exploitation of the battery branch to provide the necessary power for the vehicle at the beginning of the journey. By doing this, the procedure never chooses a negative value for the torque split which would enable the recharging procedure; nevertheless the battery cannot be infinitely discharged, and at some point, the lower safety value for the SOC is reached. In this case, the torque split factor is chosen equal to 0 in order to keep constant the value of the state of charge, that is, the battery branch is no more considered and the whole amount of power is only provided by the ICE engine branch. Thus, benefits provided by the battery in minimizing fuel consumption would quickly expire.

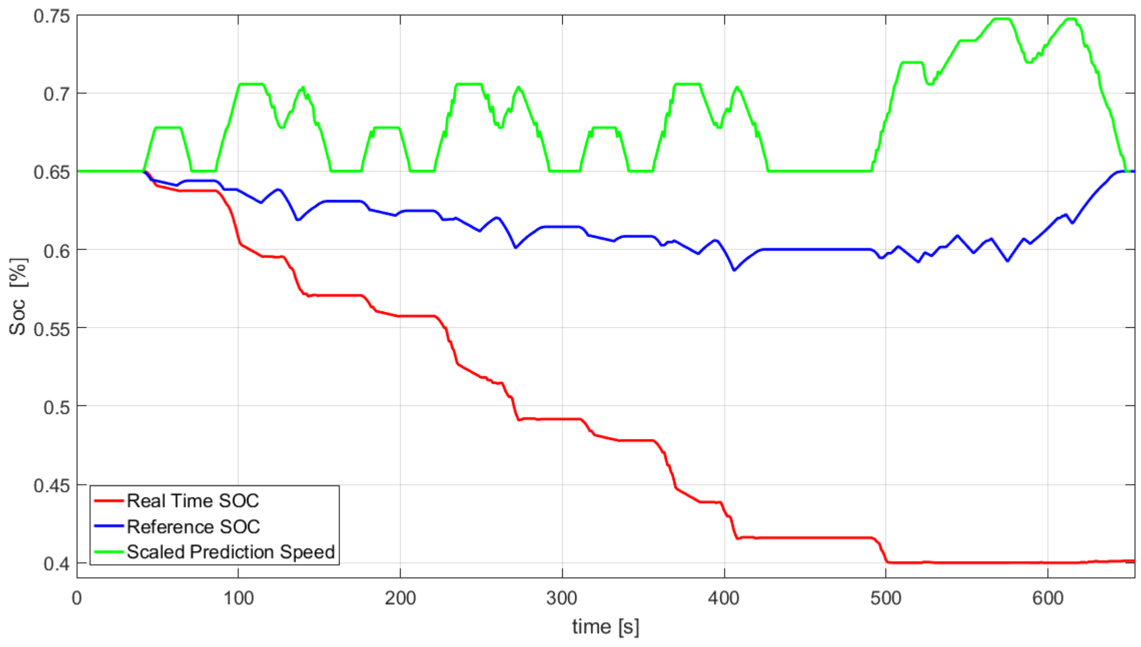


Figure 4.10: Comparison between the SOC computed with deterministic dynamic programming applied to the whole horizon available and the SOC obtained with the current real time procedure, characterized by stochastic dynamic programming method

Figure 4.11 confirms the previous statements: at the beginning, the real time torque split is usually equal to one, that is, the power is fully provided by the battery branch, and the behaviour looks similar to the optimal one; when instead the state of charge reaches the lower possible value, the torque split mostly assumes a value equal to 0, that is, the total necessary torque is computed only by the engine branch.

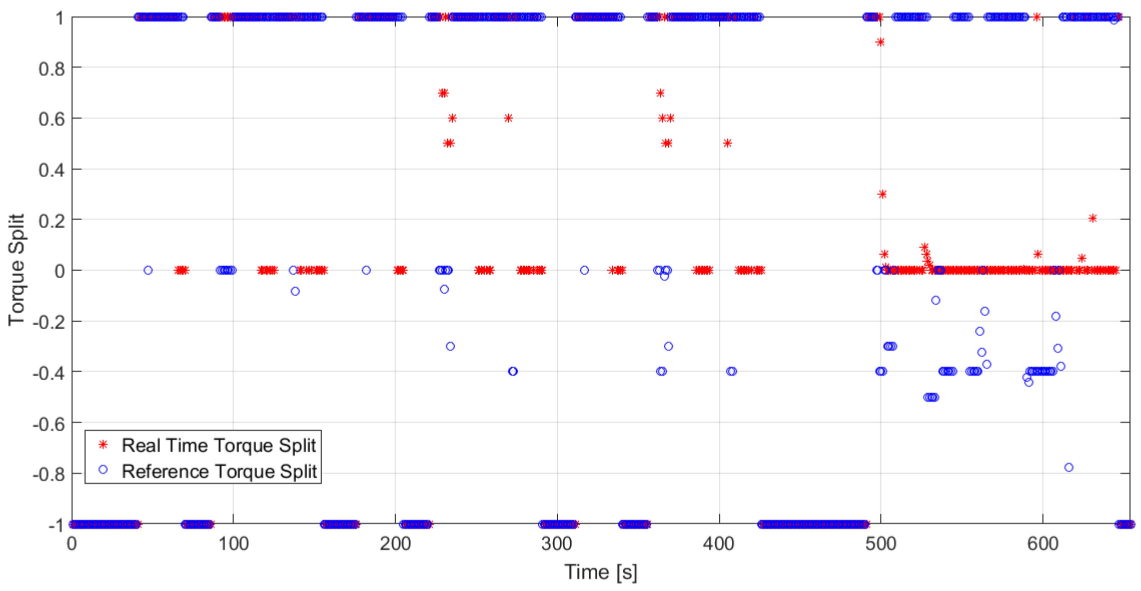


Figure 4.11: Torque split computed with the whole horizon known is compared with the optimal torque split computed with real time procedure

We can see that this particular real time procedure never chooses a negative value for the torque

split[1], because in such a small horizon a recharging phase would require more fuel than what 0 or a positive value would need, and, therefore, a negative value is always more expensive than a positive one. When instead the horizon is larger and known, the possibilities to compute a global minimum and impose a constraint on the final state allow the procedure to select both discharging and recharging phases in a proper way, in order to minimize the whole fuel consumption and restore the initial battery state of charge.
Behaviour illustrated in Fig. 4.10 is not acceptable; so let us now introduce a new modified optimization cost procedure that takes into account both fuel consumption and level of the state-of-charge, in order to overcome issues caused by the small dimension of the horizon and the unconstrained final value for the SOC.

4.4.1 Cost Function update

The above problem formulation does not have any constraint on terminal state of charge SOC, therefore, the optimization algorithm tends to deplete the battery in order to achieve minimal fuel consumption. Hence, we impose a new functional of cost to constrain the SOC current value:

$$\alpha(SOC(k) - SOC_f)^2 \tag{4.24}$$

Function (4.24) is a quadratic penalty factor, also called regularization term, which is made up of a parameter, α, multiplied by the squared difference between the current SOC and the desired final value for it.
In an estimation setting, the extra term, penalizing large deviations between current and final state of charge, can be interpreted as our prior knowledge that the current state of charge is not too far from the desired final value. In an optimal design setting, this quantity increases the cost of missing the target specifications for inputs which contribute in depleting the battery; as a consequence, it allows to keep the state of charge close to the desired value SOC_f. The control input on every small horizon is calculated by solving the following new optimization problem, which minimizes the weighted sum of the two costs:

$$\min_{u_k \in \mathcal{U}_k} E[\Delta m_f(u_k, w_k, k) + \alpha(SOC(k) - SOC_f)^2] \tag{4.25}$$

$$\textbf{s.t.:} \tag{4.26}$$

$$x_{k+1} = f_k(x_k, u_k, w_k) + x_k \tag{4.27}$$

$$x_0 = x_k \tag{4.28}$$

$$x_k, x_{k+1} \in [x_{min}, x_{max}] \tag{4.29}$$

Although the introduction of regularization terms can control the state of charge value, this raises the question of how to determine a suitable value for the regularization coefficient α. Of course, looking for the solution which minimizes the cost function with respect to both the weights involved, lead to the un-regularized solution $\alpha = 0$, that is not desirable.
According to [16], operation of *Bias variance decomposition* (a brief explanation is discussed in Appendix C) splits the cost function into the sum of a bias and a variance, and the model with the optimal predictive capability is the one that leads to the best balance between them. Parameter α plays an important role in this task: in fact high values of α limit the variance but produce an increase in bias. On the contrary, low values lead to a small bias but also to a big variance.
The similarity with our cost function can be here exploited: fuel consumption term can be considered as the bias we want to minimize, while the squared difference between the State-of-charge values (the regularized term (4.24) without α) is the variance. Our main targets are the minimization of the fuel consumption and the restoration of the SOC value at the end of the journey of the vehicle; so, let us see what happens with different values of α. We are going to exploit the

[1]when -1 happens, the vehicle is not moving, and the battery is not recharging

Japanese driving cycle with only real data, without considering predictions procedure, in order to see what happens to the state-of-charge and the consumed fuel when α changes.
Figure 4.12 illustrates these behaviours: a large value of α pulls down the variance and the regularization factor (4.24) towards 0, leading to larger values for the fuel consumption; conversely, small values for α cause a reduction in fuel consumption leading to a larger variance. A small variance reduces in fact the possible contribution offered by the battery in supplying torque to the vehicle, attaining to a larger use of the ICE branch, and, as a consequence, to a larger fuel consumption. According to these consideration, we should impose a value for α as small as possible. Nevertheless, the final state of charge must be completely restored to the initial value, and a big deviation of the current state of charge might prevent it. Fig. 4.12 shows that for Japanese driving cycle, we cannot set α to a value smaller than 6, which is the best solution, otherwise the procedure would fail in restoring the initial state of charge.

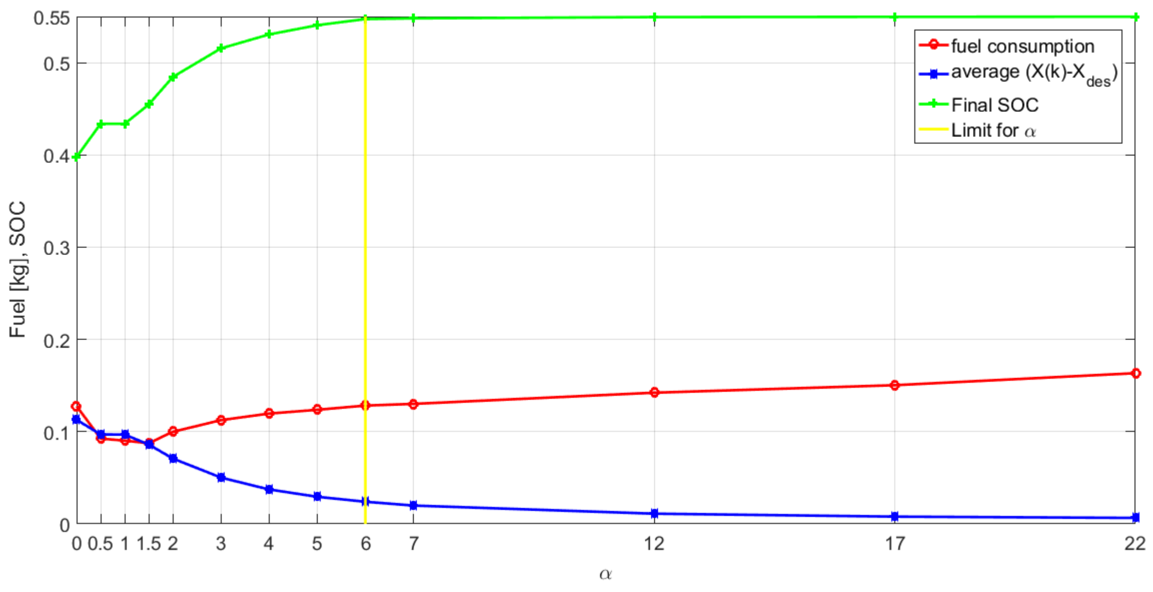


Figure 4.12: Comparison between fuel consumption, final state and average difference between current state of charge and desired final value computed with different α values applied to Stochastic Dynamic Programming Procedure.

Another interesting thing to analyse in Fig. 4.12 is the comparison between the fuel consumption computed with $\alpha = 0$ and the contribution obtained with small values of α: the quantity of fuel consumed with $\alpha = 0$ is more or less the same than the quantity computed with $\alpha = 6$, but now the final state of charge is completely restored, and can be used for another journey. In fact, when the battery is depleted, it does not provide any more contribution to torque and power, and the fuel consumption becomes bigger and bigger.
The regularization factor introduces the possibility to recharge the battery: this term penalizes a little number of intervals with respect to the whole horizon, by leading the procedure in selecting a control input which consumes more fuel, in order to continue to have benefits from the battery branch along the whole journey.

4.4.2 Real Time Optimal Control Strategy

Fig. 4.13 describes the whole real time control strategy. The driver actions are estimated by the Multivariate Markov Chain procedure described in Chapter 2; speed, acceleration and gear are then passed to the Energy Management Controller, which computes the optimal torque split factor for the Hybrid Electric Vehicle Model. The output of the model is the current value of the state of charge; the squared difference between this value and the desired one, multiplied by a proportional term α, is a fundamental part for the optimization problem.

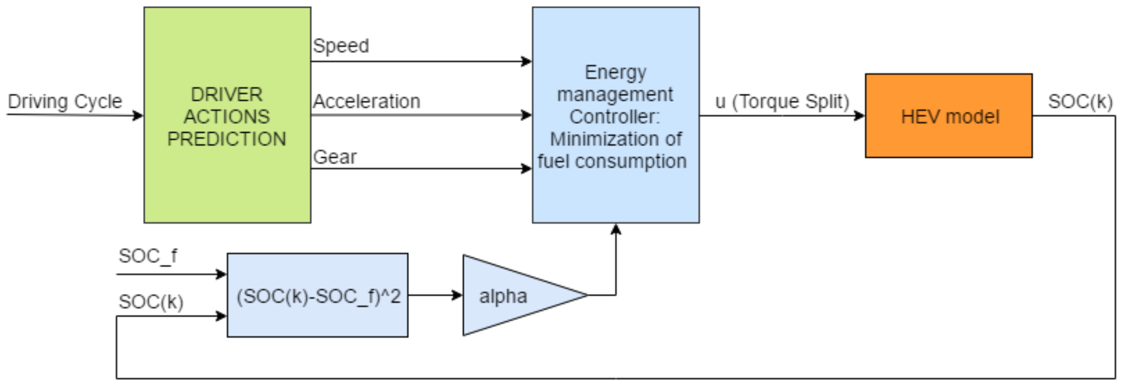


Figure 4.13: Controller for the vehicle

4.5 Simulations with real time strategy

Let us see now what happens to the state-of-charge and fuel consumption when the new real time procedure is applied to the three available driving cycles. We are going to compare the trajectories of the state of charge obtained with the following methods:

(i) Dynamic programming method with driving cycles totally known in advance (blue);

(ii) Real time stochastic dynamic programming procedure with future data estimated by using Multivariate Markov Chain procedure (red);

(iii) Real time stochastic dynamic programming procedure with real future data acquired by available data profiles (cyan).

Japanese is the first driving cycles observed; in this simulation the state of charge is constrained by safety measures inside the interval $[0.4, 0, 7]$, and the initial value is 0.65. Finally, control input u belongs to the interval $[-1, 1]$. Figure 4.14 shows the different trajectories of the state of charge; the red trajectory looks similar to the benchmark computed with dynamic programming procedure (blue), but it is quite closer to the limit of 0.65, due to the introduction of the functional (4.24).
The trajectories computed with real time procedure are quite similar, but the one computed with real speed data looks smoother than the other obtained with predicted data, because of the errors and the oscillations in the prediction of speed. It might happen that the predicted value is bigger than the current one, and the real future value is smaller; for example, the algorithm can confuse an acceleration phase with a breaking one, and as a consequence, the optimal torque split applied discharges the battery instead of recharging it.
This issue might also affect the final value for the SOC computed with estimated speed data, as we can see in Figure 4.15: small oscillations of the predicted speed contribute to have a little displacement between the desired final value and the real one. α value for the functional in this simulation has set to 9 (instead of 6) in order to make the algorithm more robust to these oscillations without increasing too much the fuel consumption mass. However, if the difference is very small as in this case, the state of charge can be easily brought to the desired value.

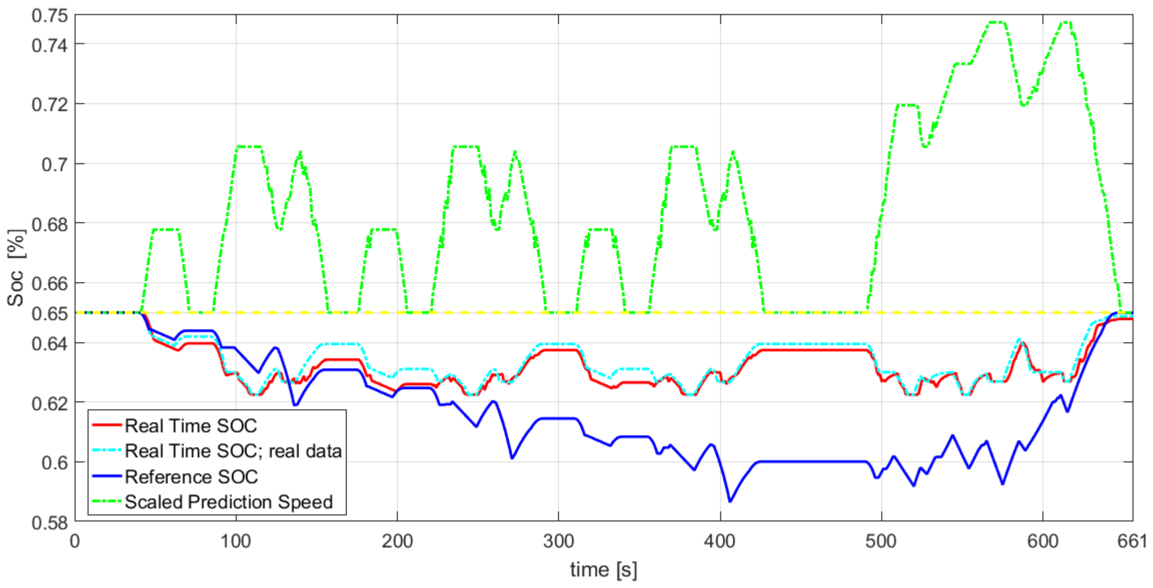


Figure 4.14: Comparison between the different SOC trajectories computed with three different methods and Japanese driving cycle

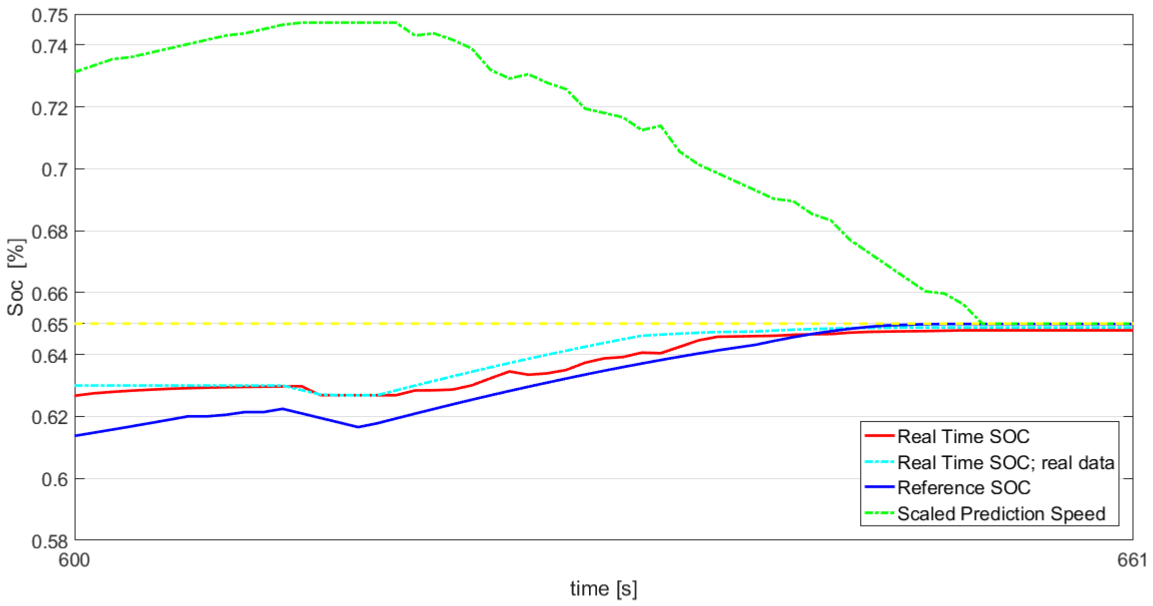


Figure 4.15: Last 60 simulation data values. Oscillations in speed due to prediction errors caused displacement between real final value and desired final value

Let us now see what happens with the Indian driving cycle. Unfortunately some final data of this profile are missing, due to failure in getting correct measures; in fact the last measured speed of the driving cycle is not equal to 0, as we would expect, so the desired final state of charge is not reached by the real time procedure for small α.
According to Fig. 4.16, the minimum value, which respects both minimization and final state of charge constraints, is 12, but, because of the lack of some data of the driving cycle, we might consider also values starting from $\alpha = 10$. However, for the same reason discussed before, we will consider for our simulation $\alpha = 12$.

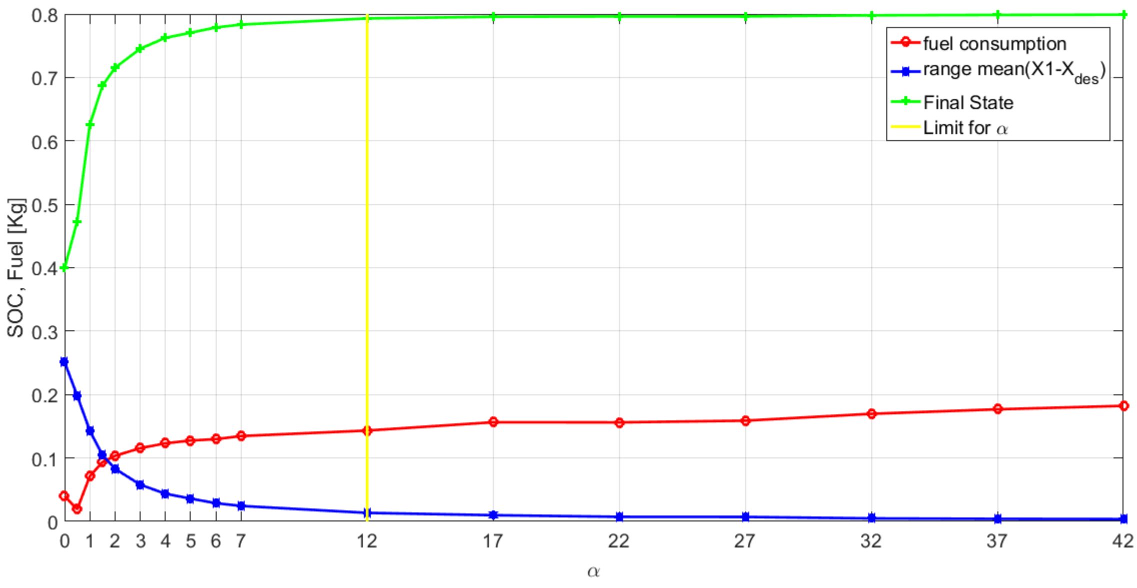


Figure 4.16: This picture shows the different values for fuel consumptions and state of charge computed with different α for Indian driving cycle.

As before, Fig. 4.17 compares the trajectories of the state of charge computed with three different methods. In this case we consider a different value for the initial and desired state of charge, that is, 0.8. The upper limit is replaced with 0.9.

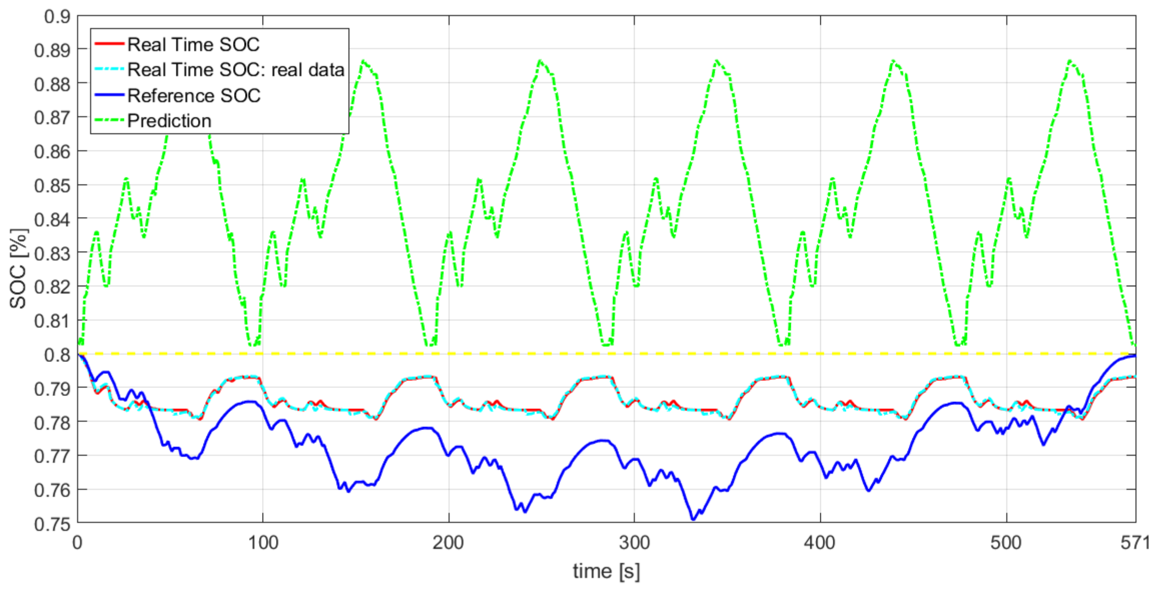


Figure 4.17: Comparison between the different SOC trajectories computed with the three different methods applied to Indian driving cycle

We can note that also in this case the procedure respects the condition which exploits the acceleration phase as a discharging phase for the battery SOC, while the deceleration phase contributes in recharging it. The different initial state does not provide any particular difference in computing the trajectories. The latter computed with real time procedures are very similar, because we have already seen that the speed prediction profile, computed by Multivariate Markov Chain algorithm, is optimal for Indian driving cycle. For the same reason, the fuel consumption mass is similar, too.

Now we try the stochastic dynamic programming algorithm with WLTP driving cycle. Fig. 4.18 shows how to choose the best α to satisfy the constraints. In this case the minimum α is equal to 4, but for our simulation we choose it equal to 5.

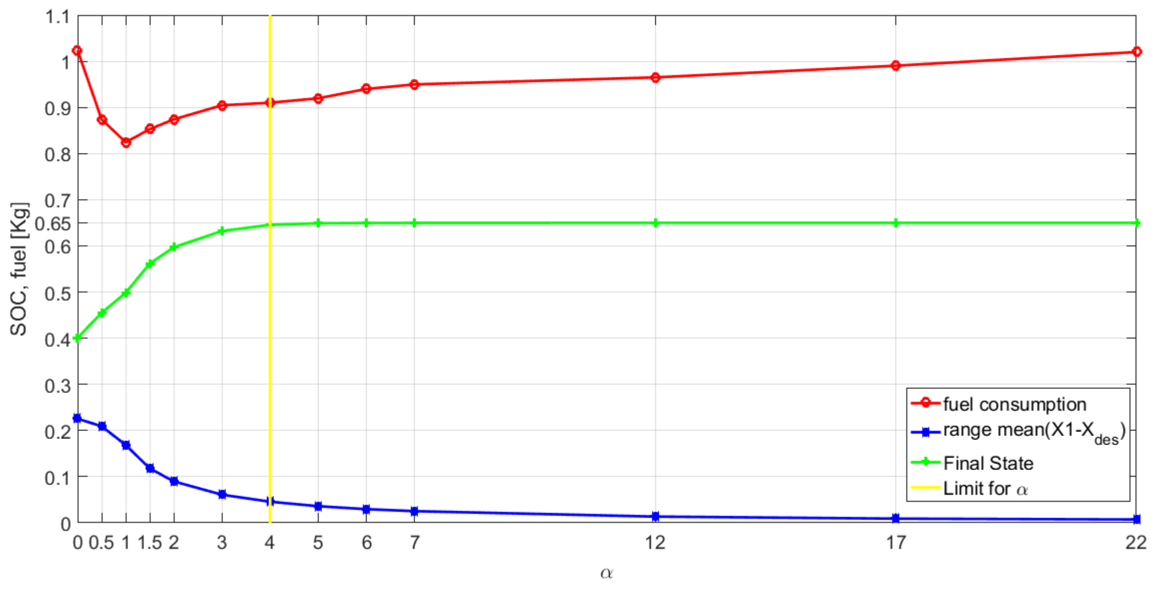


Figure 4.18: This picture shows the different values for fuel consumptions and state of charge computed with different α for WLTP driving cycle.

Fig. 4.19 illustrates the trajectories of the state of charge computed with $\alpha = 5$. Again, the real time behaviour looks very similar, especially in the second part. The final state-of-charge constraint is respected.

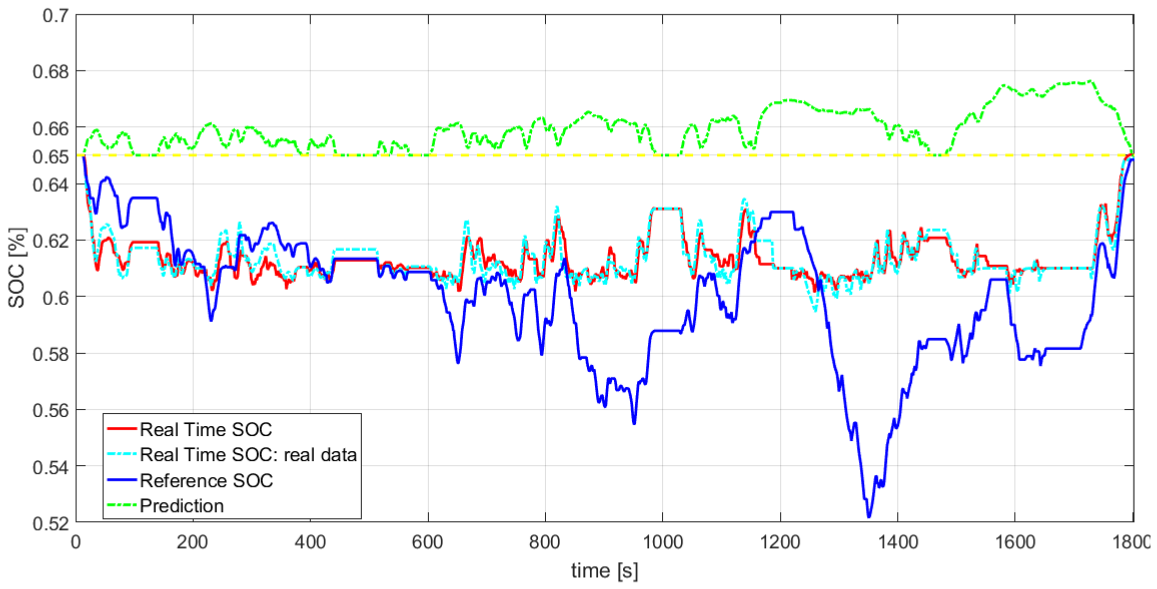


Figure 4.19: State of charge trajectory computed with the three different methods and WLTP driving cycle.

Table 4.2 stores the main results of the previous simulations about fuel consumption, number of data and value of α. Of course, fuel consumption masses are bigger than those stored in table 4.1, because knowledge of the whole horizon allows to exploit optimally the battery branch.

Table 4.2: This table stores the fuel consumed by the different driving cycles when stochastic dynamic programming procedure is applied. α value are also stored, in order to see how these value change according to the driving cycles

Drive cycle	Number of Data	Fuel Consumption [Kg]	Horizon length	α
Japan	661	0.1543	1	9
Japan with real data	661	0.1392	1	9
Indian	571	0.1433	1	12
Indian with real data	571	0.1389	1	12
WLTP	1804	0.9960	1	5
WLTP with real data	1804	0.9196	1	5

4.6 Speed-Dependent Alpha

In the previous section, Table 4.2 shows that for three different possible profiles, the chosen value of α changes significantly, and it basically depends on the prior knowledge we have of the driving cycles, represented in the three figures 4.16, 4.12 and 4.18. If the maximum speed reached is high, the value of α can be set low, because high velocity is related to a long breaking phase, which can easily recover the state of charge. But, of course, in a real situation the vehicle does not know the intentions of the driver, and, if the driver does not reach high speed values, a small α might not help in reaching the desired final state of charge. Then, α should not be kept constant for the whole horizon, but it must adapt based on the different speed required by different types of driving cycles. To confirm these statements, we can look at the SOC trajectory computed with WLTP driving cycle illustrated in Fig. 4.19: in this driving cycle there are some moments when the vehicle is not moving, probably due to traffic conditions, where the related state of charge does not reach the desired final value. Moreover, the whole SOC trajectory, compared with the benchmark computed with the whole horizon known, is limited into a small interval of values centred in 0.62, oscillating between charging and discharging phases. So let us define α as a linear function of speed measured in [km/h]:

$$\alpha = \begin{cases} M - \beta v & v \in [0; M-L] \ \ [Km/h] \\ L & v \in [M-L; \text{MAX}] \ \ [Km/h] \end{cases} \tag{4.30}$$

where M and L are the maximum and the minimum value for α, respectively, while β is the slope of the line and MAX is the maximum value of speed. According to these relations, the higher the speed, the lower α is. So the definition of the constant parameter α is replaced by setting the limits $[L, M]$ that α can reach and the value of the slope β. These parameters can be set according to a prior knowledge of the behaviour of the vehicle with different driving cycles, therefore, we do not need any information on the journey. Fig. 4.20, illustrates a graphical representation of equation (4.30).

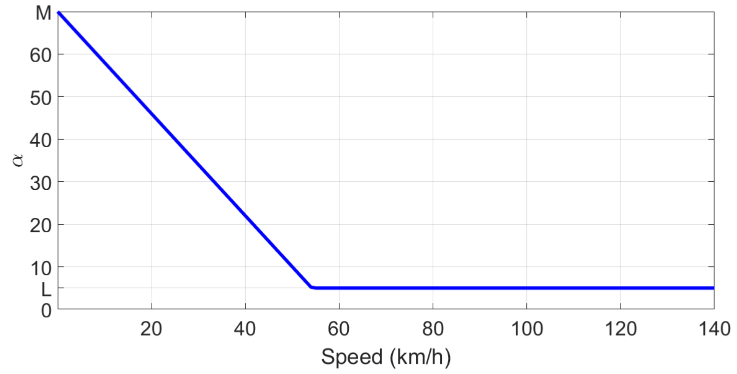


Figure 4.20: Graphical representation of equation 4.30. This figure illustrates α as a function of speed

Now let us test the new procedure with the Japanese driving cycle and the following parameters: $\beta = 1.2$, $M = 70$, $L = 9$. Corresponding to these values, α is high for small speeds, that is, the battery is not really exploited to provide torque, leading to an increase in fuel consumption. This happens because the deceleration from a low speed might not have the possibility to restore the desired value of the state of charge. On the contrary, high speed values are associated to small α, and this enables to exploit the battery in order to save fuel. Fig. 4.21 shows the trajectories of the state of charge obtained with this strategy, and confirms the previous hypothesis: in fact, differently from Fig. 4.14, the battery is significantly discharged only when speed is quite high. Nevertheless, the final state constraint at the end of every small journey is almost always respected; small displacements are due to errors in estimation of the future speed data.

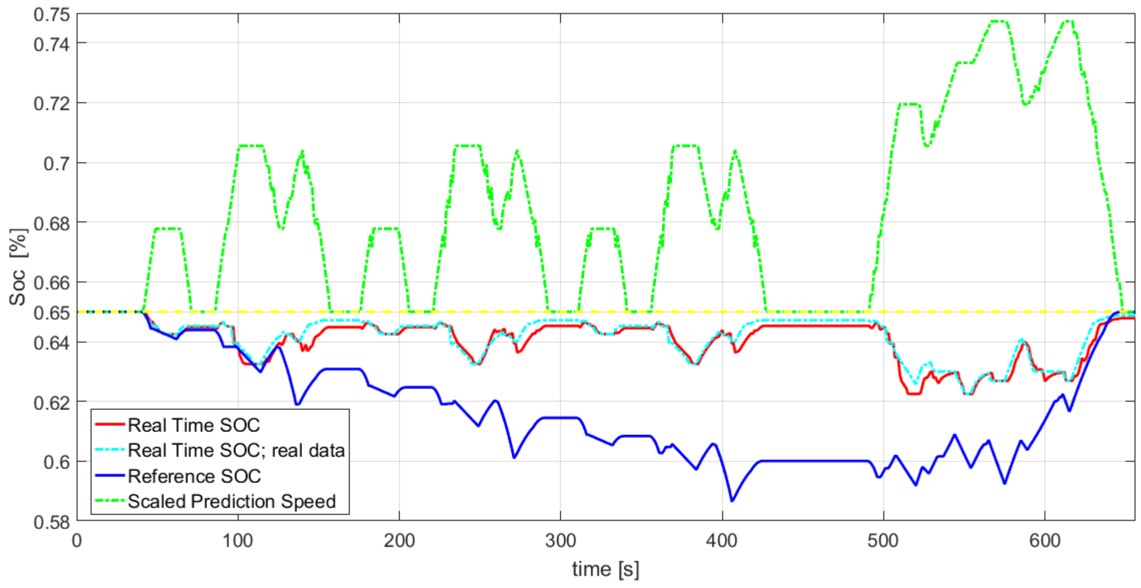


Figure 4.21: State of charge trajectory computed with the three different methods, Japanese driving cycle and α speed-dependent

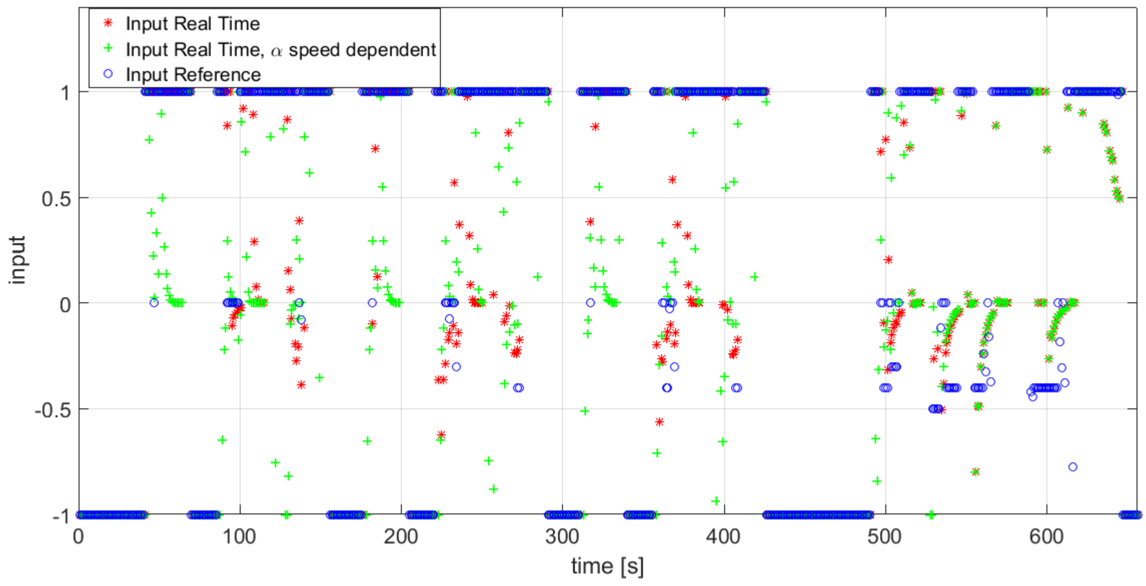


Figure 4.22: Comparison between the benchmark (blue) and two input profiles computed with real time procedure. The second (red) is the profile computed with α speed dependent strategy

Fig. 4.22 compares the optimal input computed with the three different strategies adopted, real time SDP (red), real time SDP with α speed dependent (green) and deterministic dynamic programming (blue), respectively. The motor branch is exploited more by real time procedures, especially the one with speed dependent α. This is normal because the deterministic procedure can compute the optimal input over the whole horizon and can choose the best moments to recharge the battery. Stochastic procedure instead is constrained by the small horizons, so the battery cannot be fully exploited to provide the required torque, especially with speed-dependent α strategy.
Fig. 4.23 shows instead the results obtained with procedure applied to WLTP driving cycle. In this case we set parameters as follows $\beta = 1.4$, $M = 70$, $L = 5$. Here the benefits of the new procedure are clear: every time the vehicle has a limited speed, the battery is not really exploited and the state of charge remains close to the desired value, especially when the vehicle is not moving. On the contrary, when the speed is very high, the battery provides the whole required torque to move the vehicle, saving fuel mass.

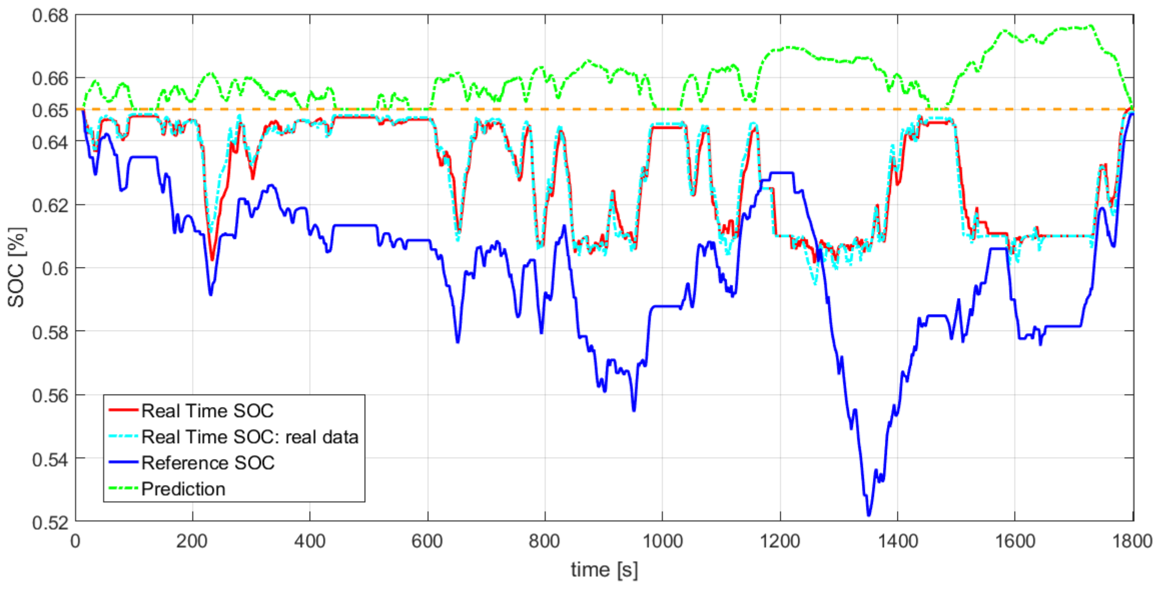


Figure 4.23: State of charge trajectory computed with the three different methods, WLTP driving cycle and α speed-dependent

Table 4.3 stores the fuel consumption values computed with the new method. Of course, the quantities are a bit higher than those computed with the procedure with constant α, because the recharging phases, that often recur to restore completely the state of charge, have to consume more fuel than before, especially when the vehicle is going to stop.

Table 4.3: This table stores the consumed fuel when stochastic dynamic programming with speed-dependent α is applied to the three driving cycles involved.

Driving cycle	Number of Data	Fuel Consumption [Kg]	Horizon length
Japan	661	0.1747	1
Japan with real data	661	0.1603	1
Indian	571	0.1618	1
Indian with real data	571	0.1563	1
WLTP	1804	1.0430	1
WLTP with real data	1804	0.9693	1

The last procedure consumes more fuel than the normal one, but it allows to restore the state of charge in any particular situation of the journey. Moreover, new trajectories of the SOC are more similar to the benchmark obtained when the whole driving cycle is known in advance. Of course, the results obtained with the deterministic dynamic programming method are unattainable, because when the whole horizon is known in advance without estimation errors, the procedure can compute the global minimum for the fuel consumption, by deciding when it is better to exploit the battery branch with respect to the engine one to provide the necessary torque to drive the vehicle. Real time procedure instead can only approach the benchmark, because without knowing the whole horizon, procedure is not able to compute the global minimum, but only local minima over small horizons. Figure 4.24 then shows how the torque is supplied by the different sources during the first 600 samples of WLTP journey. The considerations of section 4.3 for Figure 4.9 are still valid: the only difference is the bigger contribution of torque offered by the engine branch in braking phases of the vehicle.

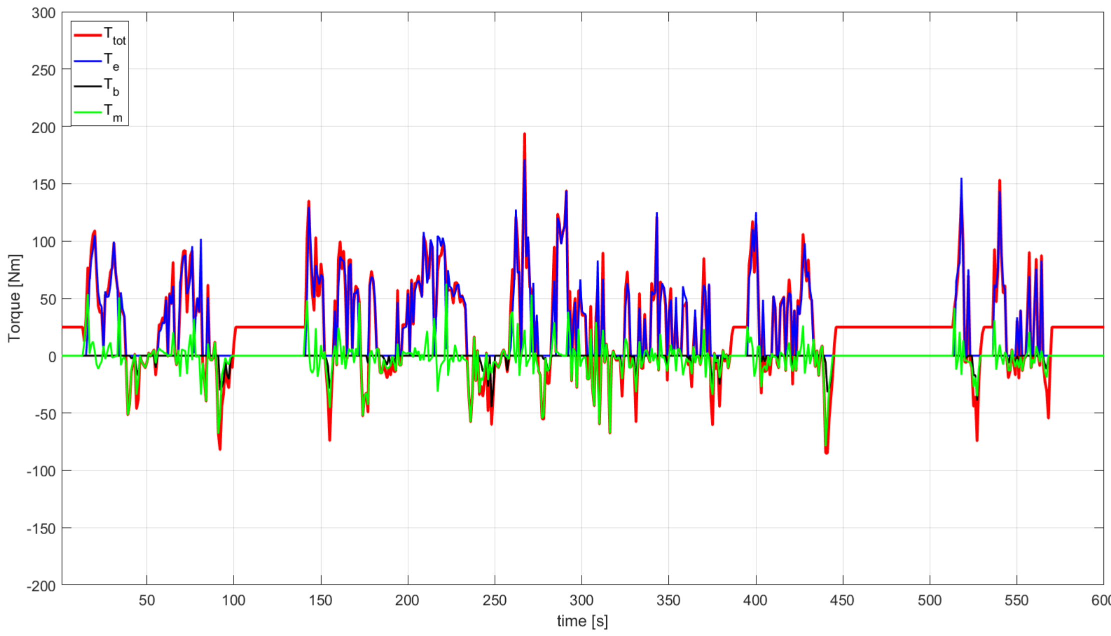


Figure 4.24: Comparison between total torque T_{tot}, engine torque T_e, T_b, and motor torque T_m computed with stochastic dynamic programming applied with speed dependent α strategy to WLTP driving cycle. Interval: 1-600 [s]

Fig. 4.25 illustrates α variations computed with WLTP: small α are correctly associated to big values of speed, while big α values are related to small speed values.

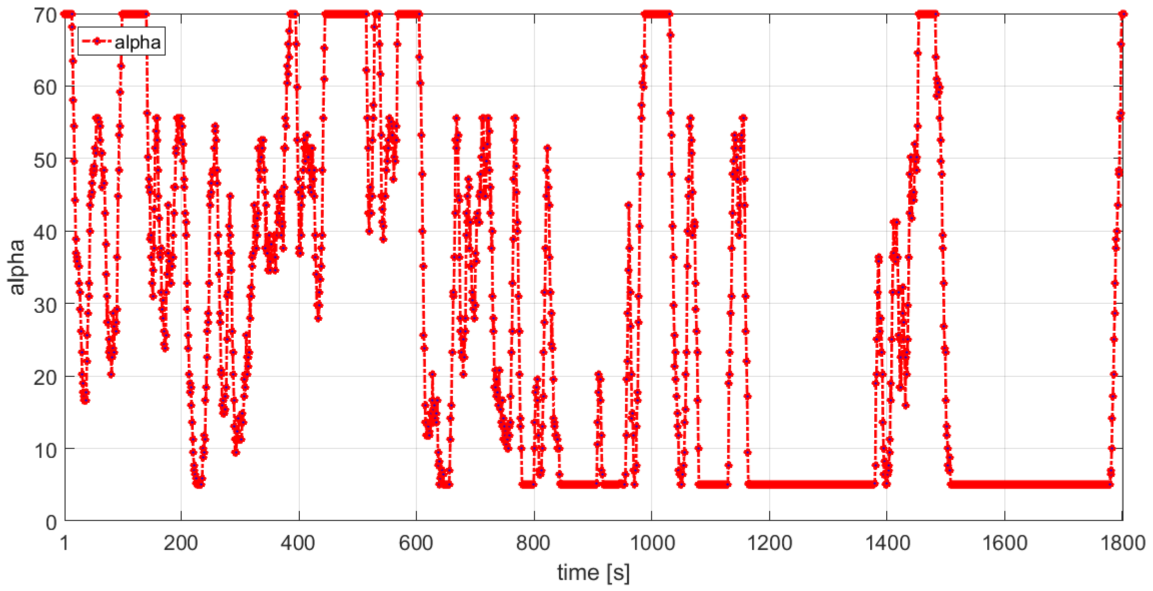


Figure 4.25: α variations with WLTP driving cycle

To conclude this chapter, let us see if battery current and engine torque respect the model constraints along the whole WLTP journey, where speed values are quite high and driver actions are more significant with respect to the other two driving cycles.
Fig. 4.26 shows the applied engine torque (red) compared to the maximum available (blue). The applied torque is always lower than the maximum available. Nevertheless sometimes the two values are quite close; maybe a more powerful engine could be more useful.

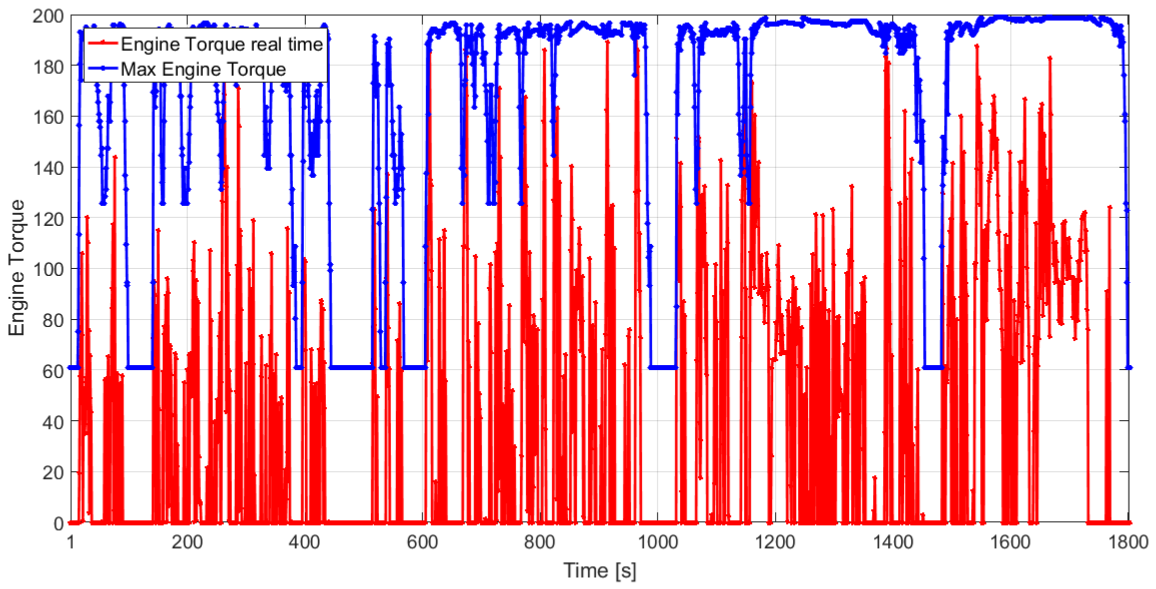


Figure 4.26: Engine torque applied compared to maximum available.

Fig. 4.27 instead shows the required battery current (red) compared to the maximum available (green):

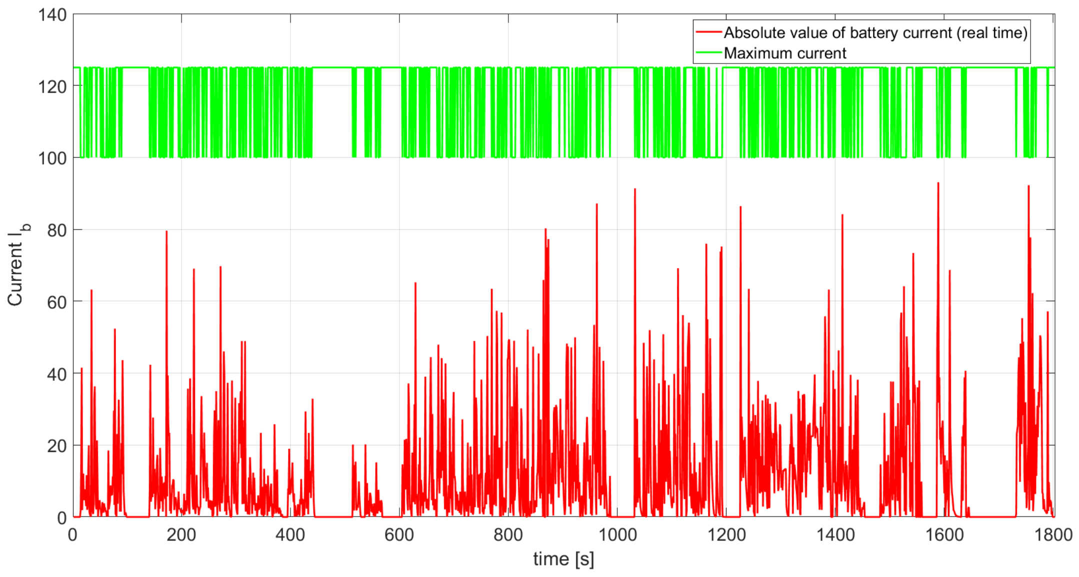


Figure 4.27: Absolute value of battery current applied compared to maximum available: the maximum value for charging current must be 200 [A], for discharging 225 [A].

Model constraints are respected with these parameters. Fig. 4.27 is interesting also because it illustrates the continuous variation in the maximum battery current, due to recurrent oscillations in estimated acceleration and speed. In the previous chapter we discussed the possibility to change the lower limit of the control input $[-L, 1]$. The results obtained for $L = 1$ and $L = 2$ are quite similar, the only difference is a small increase in fuel consumption for $L = 2$, due to a more expensive recharge process.

5 Conclusions

In this project we realized a real time power management control strategy which allows to minimize fuel consumption of a hybrid electric vehicle. Speed profiles of the involved driving cycles are modelled as Markov Chain Processes that represent the future uncertainty of the driver speed request under diverse driving conditions. The Markov Process is built by estimating the transition probabilities from the quantized samples of the data profile, as described in Chapter 2: transition probabilities are stored into a matrix, which maps the current speed demand to the next speed state. The speed demand is highly correlated, because the probabilities are centred around the diagonal axis and this helps estimating the future velocity value for the vehicle. A better estimation can be obtained by exploiting Multivariate Markov Chain theory, which involves more data profiles to estimate future values.
An important parameter for this prediction strategy is the length of the quantization interval, that has to be chosen as a trade off between the accuracy of the prediction and the increase in computation time. At the end, we chose to set it equal to 0.5 [km/h].
Once the future speed is available, we can calculate acceleration and gear as a function of the predicted speed; the acceleration is based on the current prediction and the two previous values, the gear instead is computed according to a gear-shifting strategy.

Then, deterministic optimization with dynamic programming has been applied over a given journey to our vehicle model, characterized by one state (battery state of charge) and one input. The goal was to find an optimal control policy which determines the torque split strategy to govern the engine and battery operations in order to save fuel. The application of such an algorithm allows to compute a global optimal solution, that provides the best possible configuration of torque split for the given driving cycle. Of course, these results can only be considered as a benchmark, because in a real time journey we do not have information on the future driver behaviour and we cannot constrain the final state-of-charge.

To solve this issue, we introduced a new infinite-horizon stochastic dynamic programming methodology, which computes local optimal solutions and provides a time-invariant state-dependent torque split strategy. This procedure requires as inputs the predicted speed, the acceleration and the gear, and computes the optimal torque split value for a limited horizon of one second. Battery depletion led to the addition of a new cost functional to the fuel consumption term in order to constrain the current state-of-charge. This functional is ruled by a parameter α which has to be chosen properly to guarantee fuel consumption minimization and state of charge restoration. According to this, a speed dependent strategy has been designed to accurately compute the parameter α and improve the results: we saw that state of charge trajectory obtained with this new strategy is quite similar to the one computed with the deterministic approach. The algorithm respects battery constraints. Furthermore, the proposed approach provides a directly implementable control design path, which is highly desirable because of its potential for a fully integrated optimal design and control process.

Finally, future works include investigations of a possible extension of the proposed method to more complex models characterized by more inputs and states: the simplicity of our model plays an important role in the real time computation, but a more detailed model could provide more accurate results. Moreover, we should compare the fuel consumption results with others obtained with different horizon lengths and with a heuristic equivalent consumption minimization strategy (ECMS) (see [17]).

A
Operating modes for the hybrid electric vehicle

Depending on the value of the torque split u, four operation modes are possible for the vehicle:

- $u = 1$, namely, the torque is entirely provided by the electric motor;
- $u \in (0, 1)$ corresponds to the case when the torque is provided from both internal combustion engine and electric motor;
- $u = 0$ corresponds to the case when the torque is only provided by the internal combustion engine ($T_m = 0$);
- negative values, i.e., $u \in [-L, 0)$, correspond to the situation when the torque provided by the internal combustion engine is higher than the demanded value, in order to recharge the battery, with $-L$ being full recharge, i.e., this value determines the maximum surplus of torque that can be provided by the engine.

The first mode provides torque only from the battery branch. Picture A.1 shows how the parallel architecture works: the engine branch is excluded, so fuel is not consumed and battery is discharged. This situation usually happens when vehicle is accelerating.

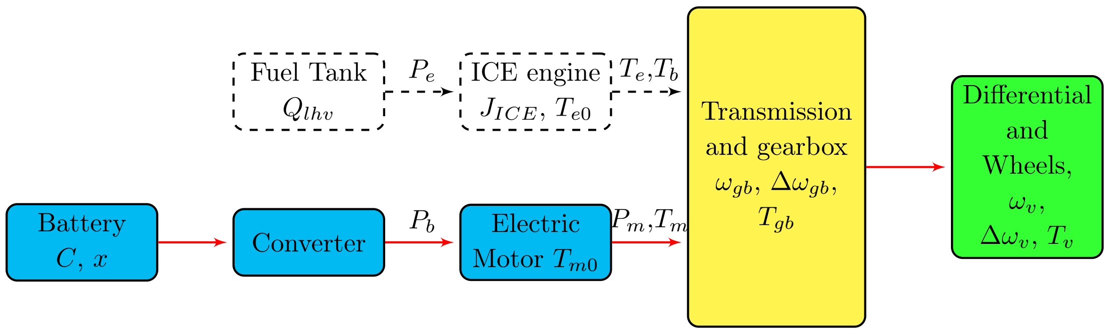

Fuel Tank Q_{lhv}
P_e
ICE engine J_{ICE}, T_{e0}
T_e,T_b
Transmission and gearbox ω_{gb}, $\Delta\omega_{gb}$, T_{gb}
Differential and Wheels, ω_v, $\Delta\omega_v$, T_v
Battery C, x
Converter
P_b
Electric Motor T_{m0}
P_m,T_m

Figure A.1: Torque Split equal to 1

Picture A.2 instead shows how parallel architecture works when the second mode is operating: when $u \in (0, 1)$, part of the torque is provided by engine branch, while the remaining part is supplied by the battery.

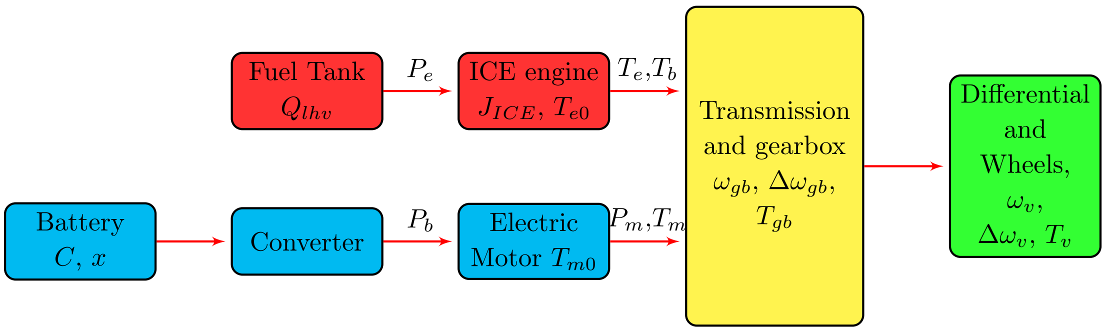

Fuel Tank Q_{lhv}
P_e
ICE engine J_{ICE}, T_{e0}
T_e,T_b
Transmission and gearbox ω_{gb}, $\Delta\omega_{gb}$, T_{gb}
Differential and Wheels, ω_v, $\Delta\omega_v$, T_v
Battery C, x
Converter
P_b
Electric Motor T_{m0}
P_m,T_m

Figure A.2: Torque Split between 0 and 1

Picture A.3 illustrates the third operating mode, when $u = 0$: the whole torque is provided by engine branch, while the battery branch is disengaged.

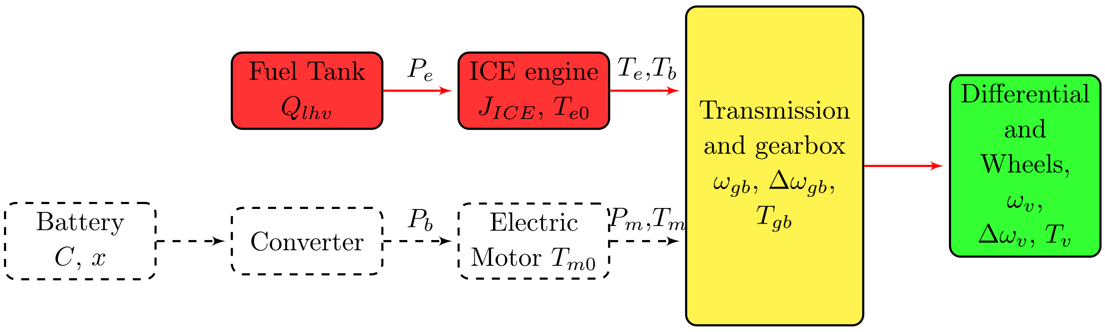

Fuel Tank Q_{lhv}
P_e
ICE engine J_{ICE}, T_{e0}
T_e,T_b
Transmission and gearbox ω_{gb}, $\Delta\omega_{gb}$, T_{gb}
Differential and Wheels, ω_v, $\Delta\omega_v$, T_v
Battery C, x
Converter
P_b
Electric Motor T_{m0}
P_m,T_m

Figure A.3: Torque split equal to 0

Finally, picture A.4 illustrates the recharging mode, when torque split is negative: the surplus of the torque provided by engine branch is used to recharge the battery. This mode usually happens when vehicle is braking.

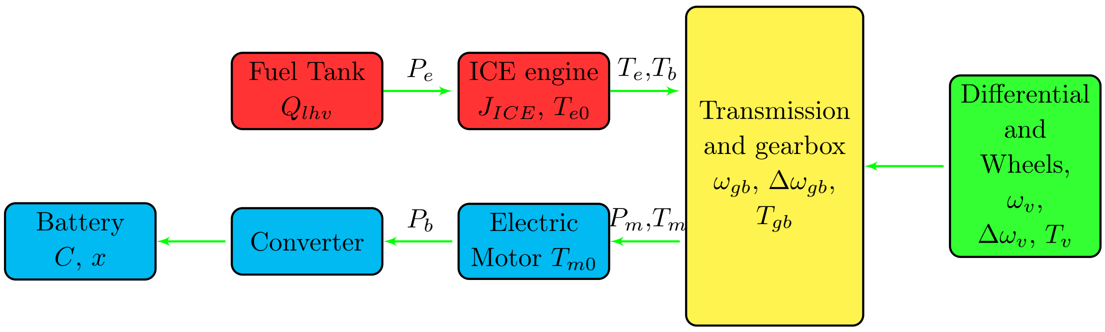

Fuel Tank Q_{lhv}
P_e
ICE engine J_{ICE}, T_{e0}
T_e,T_b
Transmission and gearbox ω_{gb}, $\Delta\omega_{gb}$, T_{gb}
Differential and Wheels, ω_v, $\Delta\omega_v$, T_v
Battery C, x
Converter
P_b
Electric Motor T_{m0}
P_m,T_m

Figure A.4: Negative torque split

$\mathcal{B}$

Dynamic Programming

B.1 Boundary Line Method

This method, described in [15], allows to solve numerical issues related to the implementation of the dynamic programming Matlab function exploited in Chapter 4 that generally occur on the the border of feasible state region. These numerical issues arise when computing the cost function for infeasible states and inputs, which are infinitely expensive and therefore should be associated to an infinite cost. The presence of the infinite cost for some states, which cannot be achieved, creates some numerical issue due to the discretization of time and state space. For example let us define the set of reachable states over one time step $\Omega_k^i = \{x | x = F_k(x^i, u)\ \forall u \in \mathcal{U}\}$, starting from a given state x^i by using all admissible inputs, where $\mathcal{U}$ is the set for the decision variable u: if the dynamic programming function is calculating the optimal cost-to-go for the state x^i at time $k+1$ and an infinite cost is used for infeasible states together with a linear interpolation, the feasible part of Ω_{k+1}^i would use an interpolation between an infinite cost-to-go $J_{k+2}(x^i)$ and a finite cost-to-go $J_{k+2}(x^i)$, therefore, as a result, the cost-to-go for x^i at time $k+1$ becomes infinite, although the grid point (k+1,i) is located within the feasible region. Now consider the algorithm at time k and the step of calculating the cost-to-go for the state x^i. For the same reason as for the time k+1, the cost-to-go $J_k(x^i)$ will be infinite since $J_{k+1}(x^i)$ was calculated before to be infinite. When the algorithm proceeds backwards, these effects might continue and the computed infeasible region will grow into the actual feasible region.
The first step to improve this issue is to replace the infinite cost with a very large but otherwise finite cost (like 1000), greater than the maximum value of the cost function; in this case the propagation of the infeasible region is reduced but not completely removed in the region close to the boundary line.
Moreover, the infinite gradient at boundary line is being blurred (see Figure B.1) because of the interpolation between feasible and infeasible states, that is, the optimal state trajectory cannot approach the boundary line. The new method computes the boundary line before computing the backward procedure of the dynamic programming, in order to avoid the blurring effect.

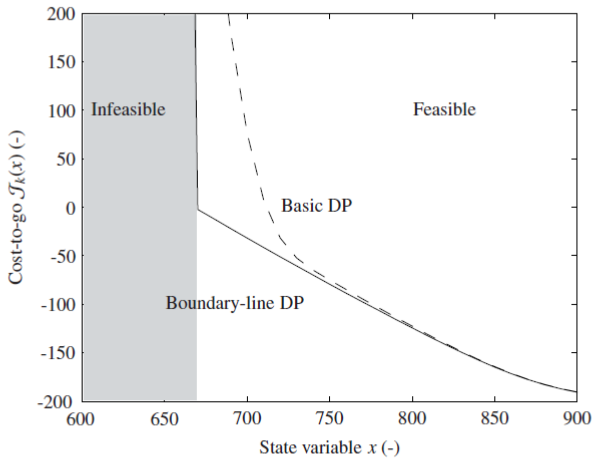


Figure B.1: Section of an example of a cost-to-go function at time index k. Basic DP does not manage to approach the boundary line, because of the blurring effect

There exist infeasible regions in the state-time space of an optimization problem with fixed final time and a partially constrained final state, that is, $x(t_f) \in [x_{f,min}, x_{f,max}]$. When the model is characterized by a single state, there exist only two infeasible regions, named upper and lower because of their position in the region, characterized by two different boundary lines. In particular, the lower boundary line is defined as the lowest state $x_{k,low}$ at time instant k that allows achieving the minimal final state $x_{f,min}$, while the upper computes the state $x_{k,high}$ at time instance k that allows achieving the maximal final state. Figure B.2 shows the two boundaries for the algorithm:

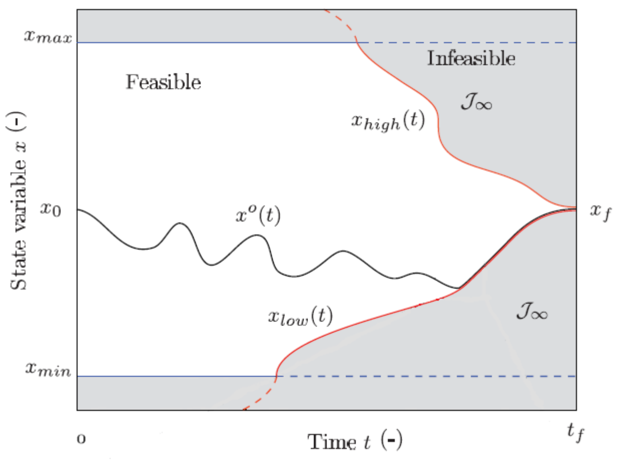


Figure B.2: State variable boundaries for the dynamic programming algorithm for the entire problem domain

The computation of the two boundary lines is analogous for both lines. The lower is computed by sequentially going backward in time from $k = N-1$ to $k = 0$ and solving the following optimization problem:

$$\min_{x_{k,low},u_k} x_{k,low} \tag{B.1}$$

$$\textbf{s.t.:}\ F_k(x_{k,low}, u_k) + x_{k,low} = x_{k+1,low} \tag{B.2}$$

$$F_k(x_k, u_k) = f(x_k, u_k) - x_k \tag{B.3}$$

$$u_k \in \mathcal{U}_k, \qquad x_{k,low} \in \mathcal{X}_k \tag{B.4}$$

$$x_{N,low} = x_{f,min}. \tag{B.5}$$

Then, by solving equation (B.2) for $x_{k,low}$, and inserting the result in (B.1), we can write the following problem:

$$\max_{x_{k,low},u_k} F_k(x_{k,low}, u_k) \tag{B.6}$$

$$\textbf{s.t.:}\ F_k(x_{k,low}, u_k) + x_{k,low} = x_{k+1,low} \tag{B.7}$$

$$F_k(x_k, u_k) = f(x_k, u_k) - x_k \tag{B.8}$$

$$u_k \in \mathcal{U}_k, \qquad x_{k,low} \in \mathcal{X}_k \tag{B.9}$$

$$x_{N,low} = x_{f,min}. \tag{B.10}$$

If the state is unconstrained, we can write:

$$x_{k,low} = x_{k+1,low} - \max_{u_k \in \mathcal{U}_k} F_k(x_{k,low}, u_k) \tag{B.11}$$

So the lower boundary line is found according to this algorithm:

Algorithm 4 Boundary Line Method

Initialization phase: $x_{k,low} = x_{f,min}$

Backward phase from $k = N-1$ **to** 0**:** The unconstrained equation (B.11) is solved as follows:

- $x^{j=0}_{k,low} = x_{k+1,low}$
- Iteration over j until a specified tolerance is achieved

$$x^{j+1}_{k,low} = x_{k+1,low} - \max_{u_k \in \mathcal{U}_k} F_k(x^{j}_{k,low}, u_k) \tag{B.12}$$

- The algorithm converges if

$$\left| \frac{\delta}{\delta x^{j}_{k,low}} \max_{u_k \in \mathcal{U}_k} F_k(x^{j}_{k,low}, u_k) \right| < 1 \tag{B.13}$$

- check if the solutions respect the state constraints; if they are not respected, the general problem (B.6) is solved.
- the solution $x_{k,low}$ with the corresponding control input u_k and the cost-to-go $J_{k,low}$ is stored.

This method presented here improves the dynamic programming results only if the optimal state trajectory is close to the bounds of the feasible region; that is the case of our constrained problem. Unfortunately, this method can only work with a model characterized by a single state.

C Regression and Regularization Term

In Chapter 4 we introduced a new cost functional in order to constrain battery state-of-charge to eliminate battery depletion. According to [18], [19], [16], this operation is called regularization, and deals with a convex optimization problem characterized by two cost functions. There are two possible forms of regularization:

$$\min_{\mathbf{x}\in\mathcal{R}^2_+} \|\mathbf{A}\mathbf{x}-\mathbf{y}\| + \alpha\|\mathbf{x}\| \tag{C.1}$$

$$\min_{\mathbf{x}\in\mathcal{R}^2_+} \|\mathbf{A}\mathbf{x}-\mathbf{y}\|^2 + \alpha\|\mathbf{x}\|^2 \tag{C.2}$$

$\mathbf{A} \in \mathbb{R}^{m\times n}$ represents a map $L : R^n \to R^m$ such that $L(x) = \mathbf{A}x$; then, $\mathbf{y} \in \mathbb{R}^m$ is a vector containing the measurements, $\mathbf{x} \in \mathbb{R}^n$ contains the parameters that have to be estimated, and $\|\mathbf{x}\|$ is the regularization factor. For a correct estimation, $\mathbf{y} = f(\mathbf{x}) + \epsilon \approx \mathbf{A}\mathbf{x}$, where ϵ is an additive zero-mean Gaussian noise. Finally, α is the regularization parameter which can vary in the interval $[0, +\infty)$. The results of equation (C.2), which is the most common, can be computed by using *Thikonov regularization least-squares method*:

$$\min_{\mathbf{x}\in\mathcal{R}^2_+} \|\mathbf{A}\mathbf{x}-\mathbf{y}\|^2 + \alpha\|\mathbf{x}\|^2 = \mathbf{x}^T(\mathbf{A}^T\mathbf{A}+\alpha\mathbf{I})\mathbf{x} - 2\mathbf{y}^T\mathbf{A}\mathbf{x} + \mathbf{y}^T\mathbf{y} \tag{C.3}$$

which has the following solution:

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A}+\alpha\mathbf{I})^{-1}\mathbf{A}^T\mathbf{y}. \tag{C.4}$$

Since $(\mathbf{A}^T\mathbf{A}+\alpha\mathbf{I})^{-1} > 0$ for any $\alpha > 0$, the regularized least-squares solution does not require any rank (or dimension) assumptions on the matrix $\mathbf{A}$. Equation (C.4) represents a simple extension of the least-squares solution. Although the introduction of the regularization term is important to constrain battery SOC, this raises the question of how to choose parameter α. According to [16], *seeking the solution that minimizes the regularized error function with respect to both the weight vector* $\mathbf{x}$ *and the regularization coefficient* α *is clearly not the right approach since this leads to the unregularized solution with* $\alpha = 0$. So the method suggested is called bias-variance decomposition. Let us define the conditional expectation, that is, the optimal solution given $\mathbf{x}$:

$$h(\mathbf{x}) = \mathbb{E}(y|\mathbf{x}) = \int yp(y|\mathbf{x})dy$$

where $p(y|\mathbf{x})$ is the conditional distribution. According to [16], the difference $\epsilon = (f(x) - \mathbf{y})$ is called loss function, and the expected squared loss can be computed with the following method:

$$\int (f(\mathbf{x}) - h(\mathbf{x}))^2 p(\mathbf{x})d\mathbf{x} + \int (h(\mathbf{x}) - \mathbf{y})^2 p(\mathbf{x},\mathbf{y})d\mathbf{x}d\mathbf{y}$$

The first term depends on the choice for the function $f(\mathbf{x})$, and the goal is to find a solution for $f(\mathbf{x})$ which makes this term minimal. Because it is nonnegative, the smallest value that we can obtain for the additive Gaussian noise ϵ is zero. If we had an unlimited supply of data (and unlimited computational resources), we could in principle find the regression function $h(\mathbf{x})$ to any desired degree of accuracy, and this would represent the optimal choice for $f(\mathbf{x})$. However, in practice we have a data set $\mathcal{D}$ containing only a finite number N of data points, and consequently, we do not know the regression function $h(\mathbf{x})$ exactly.
Function $h(\mathbf{x})$ can be modelled by using a parametric function $f(\mathbf{x}, \mathbf{w})$, where $\mathbf{w}$ is a parameter vector based on the data set $\mathcal{D}$. From a Bayesian perspective, the uncertainty in our model is expressed through a posterior distribution over $\mathbf{w}$.
If we consider the integrand function of the first integral, for a given data set we can replace $f(\mathbf{x})$ with $f(\mathbf{x}, \mathbf{w})$. Let us add and subtract to this integrand the expectation over data set $\mathcal{D}$ of $f(\mathbf{x}, \mathbf{w})$; we obtain:

$$\begin{aligned}(f(\mathbf{x},\mathbf{w}) - E_{\mathcal{D}}[f(\mathbf{x},\mathbf{w})] + E_{\mathcal{D}}[f(\mathbf{x},\mathbf{w})] - h(\mathbf{x}))^2 = \\ (f(\mathbf{x},\mathbf{w}) - E_{\mathcal{D}}[f(\mathbf{x},\mathbf{w})])^2 + (E_{\mathcal{D}}[f(\mathbf{x},\mathbf{w})] - h(\mathbf{x}))^2 \\ - 2(f(\mathbf{x},\mathbf{w}) - E_{\mathcal{D}}[f(\mathbf{x},\mathbf{w})])(E_{\mathcal{D}}[f(\mathbf{x},\mathbf{w})] - h(\mathbf{x}))\end{aligned}$$

Then we compute the expectation with respect to $\mathcal{D}$; the result is the bias-variance decomposition:

$$\underbrace{(E_{\mathcal{D}}[f(\mathbf{x},\mathbf{w})] - h(\mathbf{x}))^2}_{\text{Bias}^2} + \underbrace{E_{\mathcal{D}}[(f(\mathbf{x},\mathbf{w}) - E_{\mathcal{D}}[f(\mathbf{x},\mathbf{w})])]^2}_{\text{Variance}^2}$$

The first term, called the squared bias, represents the extent to which the average prediction over all data sets differs from the desired regression function. The second term, called the variance, measures the extent to which the solutions for individual data sets vary around their average, and hence this measures the extent to which the function $f(\mathbf{x}, \mathbf{w})$ is sensitive to the particular choice of data set. In our case, fuel consumption term can be considered as the squared bias that has to be minimized, while the variance is the squared deviation of the battery state-of-charge with the desired value.
So, α can be chosen according to these statements:

- α large reduces the variance term but, conversely, produces a high bias, which is undesirable;
- α small instead reduces the bias term and increases the variance.

So α should be chosen as small as possible; nevertheless the regularization term is added because we do not appreciate a large state-of-charge variance which does not enable the recharging phase for the battery. Hence, α must be set as a trade off to keep significantly low both bias and variance.

Bibliography

[1] R. Lot and S. Evangelou, "Lap time optimization of a sports series hybrid electric vehicle," *Proceedings of the World Congress on Engineering, London, U.K.*, vol. 3, pp. 1711–1716, 3-5 July 2013. p. 7

[2] ——, "Green driving optimization of a series hybrid electric vehicle," *52nd IEEE Conference on decision and control, Florence, Italy*, pp. 2200–2207, December 10-13 2013. pp. 7, 8

[3] O. Sundstrom, L. Guzzella, and P. Soltic, "Optimal hybridization in two parallel hybrid electric vehicles using dynamic programming," *The International Federation of Automatic Control, Proceedings of the 17th World Congress, Seoul, Korea*, pp. 4642–4647, july 6-11 2008. pp. 7, 8, 41

[4] C. Lin, H. Peng, J. Grizzle, and J. Kang, "Power management strategy for a parallel hybrid electric truck," *IEEE Transactions On Control System Technology*, vol. 11, no. 6, pp. 839–849, November 2003. p. 8

[5] C. Lin, H. Peng, and J. Grizzle, "A stochastic control strategy for hybrid electric vehicles," *Proceedings of the 2004 American Control Conference*, vol. 5, pp. 4710–4715, January 2004. p. 8

[6] T. Leroy, J. Malaize, and J. Corde, "Towards real-time optimal energy management of hev powertrains using stochastic dynamic programming," *IEEE Vehicle Power and Propulsion Conference, Seoul, Korea*, pp. 383–388, October 9-12 2012. p. 9

[7] D. P. Filev and I. Kolmanovsky, "Markov chain modeling approaches for on board applications," *Proceedings of the 2010 American Control Conference*, vol. 135, pp. 4139–4145, 2010. pp. 9, 13

[8] S. Di Cairano, D. Bernardini, and A. Bemporad, "Stochastic mpc with learning for driver-predictive vehicle control and its application to hev energy management," *IEEE Transactions on Control System Technology*, vol. 22, no. 3, pp. 1018–1031, June 2014. pp. 9, 12, 13, 23

[9] O. Sundstrom and L. Guzzella, "A generic dynamic programming matlab function," *18th IEEE International Conference on Control Applications (CCA) and 24th IEEE International Symposium on Intelligent Control (ISIC), St Petersburg*, pp. 1625–1630, July 8-10 2009. pp. 9, 47, 49

[10] W. K. Ching, E. Fung, and M. K. Ng, "Higher-order multivariate markov chains and their applications," *Linear Algebra and its Applications*, pp. 492–507, 15 january 2008. pp. 9, 15

[11] D. F. Opila, X. Wang, R. McGee, and J. Grizzle, “Real-time implementation and hardware testing of a hybrid vehicle energy management controller based on stochastyc dynamic programming,” *Journal of Dynamic Systems, Measurement and Control, 135(2), 021002 (Nov 07, 2012) (11 pages) Paper No: DS-10-1320; doi: 10.1115/1.4007238*, vol. 135, no. 2, 2012. p. 9

[12] E. Fornasini, *Appunti di Teoria dei Sistemi.* Edizioni Libreria Progetto, Padova, 2015. p. 11

[13] W. K. Ching and M. K. Ng, *Markov Chains: Models, Algorithms and Applications.* Springer, 2006. pp. 14, 15

[14] D. Bertsekas, *Dynamic Programming and Optimal Control.* Athena Scientific Belmont, 2012. p. 47

[15] O. Sundstrom, D. Ambuhl, and L. Guzzella, “On implementation of dynamic programming for optimal control problems with final state constraints,” *Oil and Gas Science and Technology*, pp. 91–102, 2009. pp. 49, 75

[16] C. Bishop, *Pattern Recognition and Machine Learning.* Springer, Singapore, 2006. pp. 58, 79

[17] A. Gao, X. Deng, M. Zhang, and Z. Fu, “Design and validation of real-time optimal control with ecms to minimize energy consumption for parallel hybrid electric vehicles,” *Mathematical Problems in Engineering, Volume 2017*, pp. 1–13, 2017. p. 72

[18] E. Rasmussen and I. Williams, *Gaussian Processes for Machine Learning.* the MIT Press, 2006. p. 79

[19] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, 2004. p. 79