POLITECNICO DI TORINO

Corso di Laurea Magistrale

in Mechatronic Engineering

Master's degree thesis

Energy Management Strategy of Fuel Cell Hybrid Electric Vehicles based on Dynamic Programming and Neural Networks

Politecnico di Torino

Supervisor: Chiar.mo Prof. Ing. Andrea Tonoli

Co-supervisor: Ing. Sara Luciani

Candidate: Giorgio Romano

Academic Year 2020/2021

To my family and friends.

Abstract

Increasing concern about climate change, air pollution and petroleum resource depletion has led regulators to impose more stringent standards in the automotive industry, which accounts for 22% of global CO2 emissions. Battery Electric Vehicles (BEVs) are the most popular among the different alternatives proposed, but Fuel Cell Hybrid Electric Vehicles (FCHEVs) are regaining attention after a setback during the last decade.

HEVs have two or more power sources that propel the vehicle. Consequently, Energy Management Strategies (EMS) play a key role in the performance of such vehicles because they seek to optimize power split between those sources to minimize fuel consumption. Modern EMSs consider additional criteria, such as increasing lifecycle of components to minimize Well-to-Wheel (WTW) emissions, thus leading to multi-objective optimization problems.

In this context, the present work aims at implementing a real time controller that enables to concurrently optimize fuel consumption, lifetime of the stack and energy utilization rate, while guaranteeing a Charge-Sustaining working mode of the vehicle. For this purpose, an offline analysis is first conducted applying Dynamic Programming (DP) to different drive cycles (among which WLTP and FTP75 are the most realistic ones), to obtain optimal power split policies by minimizing an adequate cost function while meeting constraints associated to the dynamics of the system. Specifically, DP is implemented in Matlab using the DPM function developed at ETH Zurich and a quasi-static model of the vehicle based on the architecture and components sizing of the Toyota Mirai 2021, the state of the art of FCHEVs. The vehicle is modeled through its load parameters, considering only the longitudinal dynamics. Powertrain components such as electric motor, battery and fuel cell stack are modeled using an efficiency map obtained through a data-driven approach.

Then, the Matlab Deep Learning Toolbox is used to design and train a Feedforward Neural Network (FFNN) using the data obtained from the previous stage, to approximate the DP behavior and enable on-board implementation.

To test its effectiveness, the NN controller obtained was implemented on a more realistic HEV model developed in Simulink, using Simscape libraries. Results show that the NN controller outperformed a simple PID in terms of overall cost (fuel consumption and battery electric power), computed as Gallon Equivalent. It was also observed a slightly different behavior between the higher fidelity model and the quasi-static one.

Table of Contents

1 Introduction

1.1 Motivation

In recent years, climate change has become a central topic of discussion both for scientists and government leaders. A decarbonization policy is being requested to different sectors, specially to the transport sector which currently relies almost completely on fossil fuels [1]. The main issue is that over 90% of the vehicles in use today are equipped with Internal Combustion Engines (ICE), which generate combustion wastes such as nitrogen oxides (NOx), carbon monoxides (CO) and unburned hydrocarbons (HC), all of which are toxic to human health. According to the International Energy Agency (IEA), the transport sector accounts for 22% of global CO2 emissions responsible for climate change, which is only behind the electricity and heat sector. In particular, passenger cars and trucks represent the 74% of the total transportation emissions. Figure 1.1(a) illustrates the contribution of different economic sectors to the global CO2 emissions, while Figure 1.1(b) shows such contribution by transport sector.

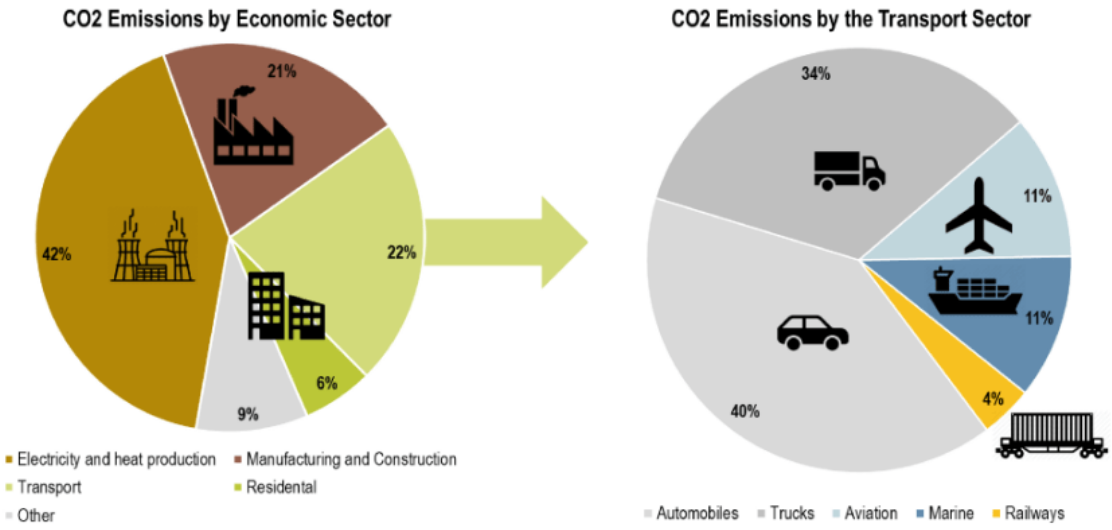

Figure 1.1 Contribution to the global CO2 emissions by: (a) economic sector and (b)transport sector. International Energy Agency. IEA and IPCC (2014) Summary for Policymakers

As a consequence, regulators are increasing the pressure to reduce emissions from road vehicles. For example, EU legislation sets targets to cut CO2 emissions from cars by 37.5 % and vans by 31 % by 2030 (EEA, 2019; EU, 2019). Therefore, it is undeniable that it is necessary to find sustainable alternatives for road transportation that pollute less and are less dependent on oil. Table 1.1 includes a classification of these alternatives [2].

Energy Resources	Energy Storage Device	Energy Converter	Hybrid Vehicles
Solar/ Wind/ Hydroelectric/ Nuclear	Battery/ Ultracapacitor	Motor/ Generator	HEV, Plugin HEV and Full EV
Hydrogen	Hydrogen Tank	Fuel Cell, Motor/Generator	FCHV
N/A	Hydraulic Accumulator	Hydraulic Motor/Pump	HHV
N/A	Air Tank	Air Motor/Compressor	Compressed air HV
N/A	Flywheel	N/A	FHEV

Table 1.1 Alternatives Vehicle – Source [2]

Among all these alternatives, the most promising ones are Battery Electric Vehicle (BEV) and Fuel Cell Hybrid Vehicle (FCHEV), both having zero Tank-to-Wheel emissions. At this point it is important to highlight the difference with the Well-to-Wheel emissions, which are strongly dependent on the methods used to produce Electricity/Hydrogen. Indeed, both are energy carriers and not primary energy sources as oil.

Although BEVs are more popular at this time, FCHEVs enable longer driving range and require less refueling time. On the other hand, FCHEVs face barriers such as fuel cells costs and the need of H2 transportation and distribution infrastructures, still at the initial development stage [3].

BEVs and FCHEVs have a relatively higher manufacturing carbon footprint than ICEVs due to production of batteries, fuel cells and alternative powertrain components at the beginning of their life. However, the total GHG emissions produced along their life cycle fall below those produced by ICEVs with increasing lifetime mileage [4]. Consequently, Energy Management Strategies (EMS) play a fundamental role, focusing not only on fuel consumption minimization, but also guaranteeing a longer lifecycle for components such as Fuel Cell and Battery. An EMS refers to a high-level control strategy which determines the power split between different sources to meet the total load request.

1.2 Aims and Objectives

In this context, the present work aims at designing and implementing a multi-objective EMS controller, considering both fuel consumption and Battery/FC lifetime in the optimization routine. The latter is achieved ensuring smooth variations of the currents provided by both components, to reduce stress and expand component life, and consequently decrease WTW emissions of FCHEVs.

The following sequence of activities will be carried out to achieve this objective:

1. Describe a quasi-static vehicle model, based on the 2021 Toyota Mirai
2. Define an adequate cost function to consider the multi-objective problem of interest
3. Generate a set of optimal policies using the Dynamic Programming (DP) algorithm as an offline optimization technique for different drive cycles
4. Create a real-time NN controller using the data generated in the previous activity
5. Test the effectiveness of the real-time NN controller implementing it on a Simscape dynamic model

1.3 Thesis Outline

The thesis is structured as follows. This chapter presented the formulation and motivation of the problem of interest, and also states the aims and objectives of the work. Chapter 2 presents a review of some concepts relevant to the development of the work, namely the architecture and modeling approaches of HEV powertrains, the operation of fuel cells, the dynamic programming (DP) algorithm and the use of feedforward neural networks to model dynamic systems. On the other hand, chapter 3 addresses the modeling of the vehicle and its components, and chapter 4 discusses Energy Management Strategies and its implementation in this work using a FFNN-based controller. At last, chapter 5 presents and analyzes the results of the work, specifically the optimal power split policy resulting from the DPM algorithm and the performance of the NN controller implemented on the Simscape model, while chapter 6 presents the conclusions and possible future works.

2 Literature Review

2.1 HEV Powertrain Architectures

According to their powertrain, vehicles may be classified as single power source vehicles, such as conventional ICE vehicles, and hybrid ones, which have two or more power sources that propel the vehicle [5]. ICE vehicles have the advantage of delivering good performance and long operating ranges, but they have poor fuel economy, dissipate kinetic energy when braking and pollute the environment. In contrast, BEVs have high energy efficiency, recovering energy in braking phases, and are non-pollutant; however, they allow smaller driving ranges. Now, the two technologies may be combined to exploit the advantages of both and overcome the individual limitations.

Since the complexity of the architecture grows as the number of power sources increases, hybrid vehicles generally consist of only two sources, namely Primary Power Source (PPS) and Secondary Power Source (SPS). According to the couplings between the different power components, the following architectures may be defined:

a. Series (electrical coupling)
b. Parallel (mechanical coupling)
c. Compound (Series-Parallel)
d. Complex

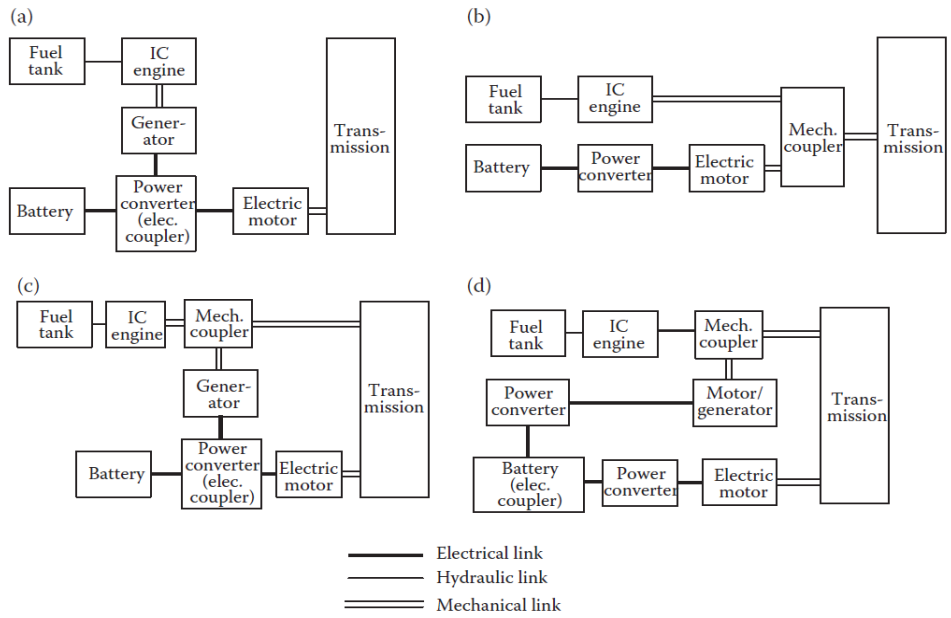

Figure 2.1 HEV Architectures: (a) Series, (b) Parallel, (c) Compound and (d) Complex - Source [6]

Figure 2.1 presents a graphical description of each of the HEV architectures.

In a Hybrid Electric Vehicle (HEV) an electrical powertrain, such as a battery, is present, and this power train allows a bidirectional flow of energy to capture regenerative braking energy that otherwise would be dissipated.

The series architecture is considered in this work, because it is the most common one for FCHEV; this architecture has a Fuel Cell as PPS, instead of an ICE. In the series architecture, two electric power sources are connected through a DC bus to an Electric Motor (EM) that propels the vehicle. The fuel tank represents the unidirectional energy source, and the FC is the unidirectional energy converter. On the other hand, the battery pack is a bidirectional energy source connected to a DC-DC converter and works as an energy bumper. The Electric Motor is the only component with coupled mechanically to the transmission, and can operate both as a motor or as a generator when braking. A vehicle controller is necessary to control operation and power flows, specifically to determine the split from the two power sources to fulfill the load requirements of the EM. This topic will be addressed in chapter 4.

In the parallel architecture, two mechanical sources of power are coupled together through a mechanical coupler. In this architecture, the connection is achieved using a "torque bus" [7].

The compound architecture is also referred to as Series-Parallel because it has features from the previous configurations, with the two sources connected both mechanically through the "torque bus" and electrically through the DC bus.

2.2 Modeling Approaches

As stated in Chapter 1, the objective of this work is to design a multi-objective EMS strategy. A fundamental step towards accomplishing such objective, is to build an appropriate model of different vehicle components. This could result in a complex task due to the multiple interconnected physical subsystems and the multiple scales of the different dynamics involved [7]. A basic modeling technique that only involves longitudinal dynamics can adequately represent the vehicle for the purpose of EMS design, avoiding to increase the order of complexity of the system to be controlled. In longitudinal dynamics the drive cycle given is divided into time steps, and the state of components is computed after each time interval.

A class of models called "Backward-Facing" are often employed in powertrain optimization [8]. These models do not use a driver model; a speed trajectory is rather imposed on the vehicle model and the torque required at wheels is computed. Following a cascade, the requirements for each component are determined backwards (hence its name) using efficiency maps that are obtained through steady-state tests, thus leading to quasi-static models that ignore transient behaviors of components like inductance and inertia. Backward facing models are feasible for a first approximation when computing fuel consumption, which is faster compared to other approaches because it has a relatively low computational load due to the use of larger time steps. However, in backward facing models the power information flow is unidirectional, i.e., effort (torque) and flow (speed) have the same direction, and thus the system is noncausal. Figure 2.2 illustrates a backward facing model.

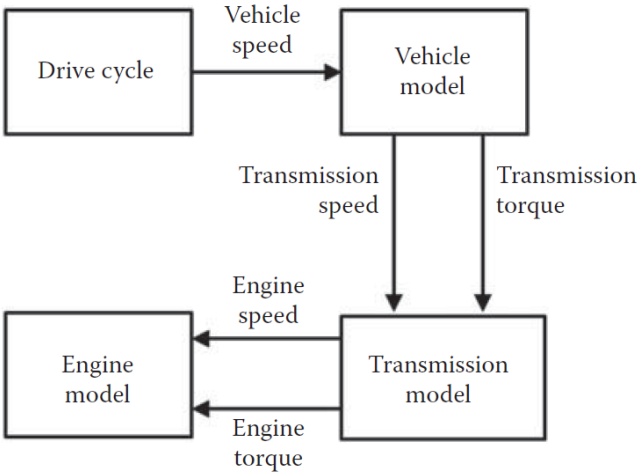

Figure 2.2 Backward-Facing Model - Source [6]

The backward facing approach is not suitable when a more realistic model is needed, as in Hardware in the Loop HIL tests. In this case it is preferrable to use the "Forward-Facing" approach, in which a driver model is introduced, usually as a PI controller, and the speed trajectory is no longer imposed. The driver translates the desired speed and acceleration into pedal commands, which further generates a request in terms of the motor torque required to track the desired speed trajectory. As opposed to backward facing, since the speed is not imposed, there may be a small error between desired and actual velocity, and the PI controller is in charge of diminishing this error. The forward-facing model is illustrated in Figure 2.3.

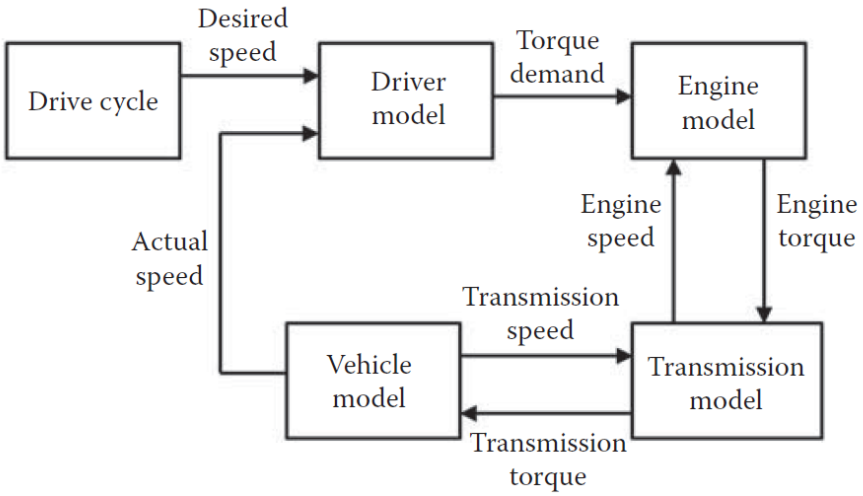

Figure 2.3 Forward-Facing Model - Source [6]

Note that the forward-facing use dynamic models, as opposed to the backward facing approach which was defined using quasi-static models. In the forward-facing approach the information flow is bidirectional, i.e., the actual output is fed back. Therefore, they give a better overview of the physical system for its use in a real application, also capturing the transient states and making it suitable for designing control systems and for implementing HIL tests. These more

realistic results come at the cost of a more complex model due to the presence of several state equations, which involves a more complex computation and thus a slower simulation because smaller time steps are required for solving this model numerically.

A backward-facing model is used in the present work, first to carry out a preliminary analysis of fuel consumption, and then for running the dynamic programming (DP) algorithm, since its quasi-static nature is better suited for this purpose. Then, a forward-facing model is used for implementing the feedback controller, to obtain a better representation of its hypothetical application on a real vehicle.

2.3 Fuel Cell Operating Principles

Fuel Cells transform into electrical energy the chemical energy generated by a reaction between hydrogen (H2) coming from the fuel tank, and oxygen taken from the external air. The behavior of fuel cells is similar to the behavior of batteries, but battery capacity is determined by the quantity of chemicals that it holds. In contrast, in fuel cells chemicals are supplied from external reservoirs, and their capacity depends only on the availability of reactants, i.e., the fuel tank capacity. The operating principle of a fuel cell is illustrated in Figure 2.4.

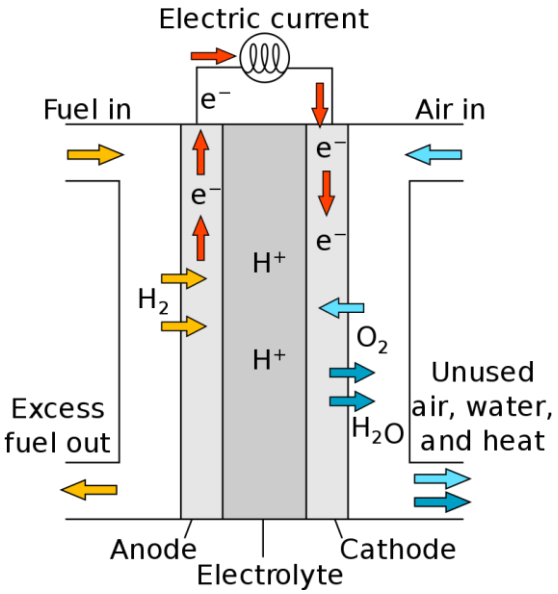

Figure 2.4 Diagram of a PEM Fuel Cell

The H2 is oxidized at the anode, realizing electrons that pass through the load and reach the cathode, where the oxygen reduction takes place. This behavior is represented as

$$H_2 \rightarrow 2H^+ + 2e^- \qquad \frac{1}{2}O_2 + 2H^+ + 2e^- \rightarrow H_20 \qquad 2.1$$

When there is no current flow, the cell has an open-circuit voltage defined by the ratio of the free energy of the cell reaction and Faraday constant

$$V_{oc} = -\Delta G/nF \qquad 2.2$$

where n is the number of electrons involved. Under standard conditions and using hydrogen as fuel, $V_{oc} = 1.229\,V$, while when drawing current from the cell voltage V_{oc} is typically $0.6 - 0.8\,V$ depending on current density [9]. For this reason, it is necessary to use several cells connected in series, which are known as fuel cell stack. The polarization curve shown in Figure 2.5 is used to represent the relationship between the DC voltage and the current density, measured in A/cm^2.

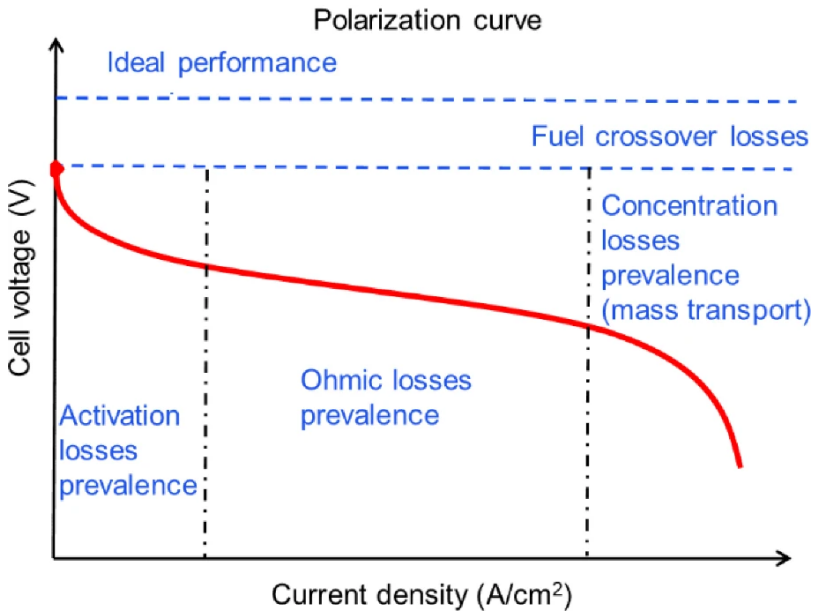

Figure 2.5 Typical polarization curve for PEM fuel cells – Source [10]

The losses in the voltage-current curve are divided in different regions. In the initial voltage drop, the losses are due to activation limitations at electrodes. Then the ohmic losses, which are due to the resistance of the electrodes to the flow of electrons, have a major role in the central quasi-linear region. Finally, concentration losses due to the change in reactant concentration are dominant in the high current density region.

To work properly, a fuel cell stack needs the following auxiliary sub-systems and components:

- Air circulating pump
- Coolant circulating pump
- Ventilation fan
- Hydrogen circulating pump
- Controller

The air circulating pump is the component with the greatest impact on the power drawn from the fuel cell and can account for 10% of the total output. Then the efficiency of the overall resulting system can be computed as

$$\eta_{stack} = \frac{P_{stack} - P_{aux}}{m_{H2}\, LHV} \qquad 2.3$$

where m_{H2} represents the mass of hydrogen consumed and LHV is the Lower Heating Value. The Higher Heating Value, which also includes the vaporization heat, can be used instead depending on the convention.

Table 2.1 shows many different types of fuel cell that are available depending on the typical operating temperature and the electrolyte type [11].

Cell System	Operating Temperature °C	Electrolyte
PEMFC	60-100	Solid
AFC	100	Liquid
PAFC	60-200	Liquid
MCFC	500-800	Liquid
SOFC	1000-1200	Solid
DMFC	100	Solid

Table 2.1 Different Fuel Cell Technologies – Source [11]

All major types use hydrogen as fuel, except the DMFC which uses methanol. In the automotive sector, there is a shared consensus that Proton Exchange Membrane Fuel Cells PEMFCs are the most suitable technology for road transport applications. Consequently, only this type is considered in the present work.

A PEMFC uses a solid polymer membrane, also referred to as Nafion (Dupont®), as the electrolyte. The membrane is acidic, and thus the ions transported are hydrogen ions H^+ or protons. The typical structure of a PEMFC is shown in Figure 2.6. The membrane is coated with a catalyst on a carbon support. The catalyst, which constitutes the electrode, is in direct contact with the diffusion layer and the electrolyte to maximize the interface. The ensemble of electrolyte, catalyst layers and gas diffusion layers are known as Membrane-Electrode Assembly (MEA).

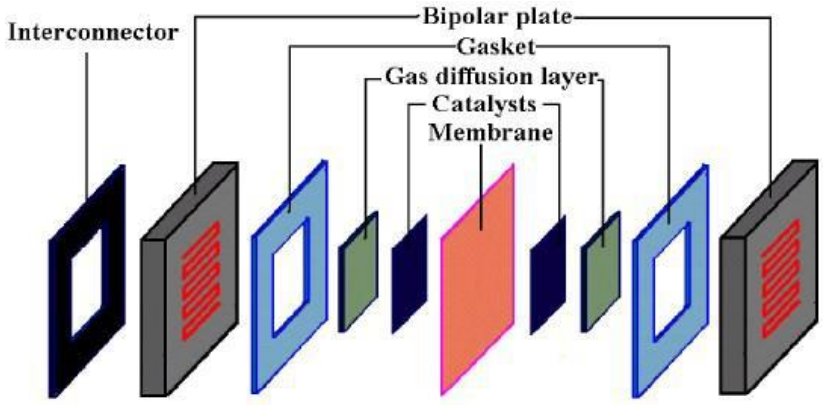

Figure 2.6 An exploded view of a PEMFC

There are three critical issues with a PEMFC. The first is the realization of the catalyst, because noble metals are required due to the low operating temperature and the acidic nature of the electrolyte. The second problem concerns water management, because the polymer membrane must be kept humid in order to function properly: if it is excessively wet the pores of the diffusion layers are blocked and the reactant cannot reach the catalyst, and if it is excessively dry there are not enough acid ions to transport the protons; an external humidifier is usually employed to keep correct humidity levels, running an excess of air. The last critical point concerns poisoning by the platinum catalyst, which has high performance at the expense of a stronger affinity for CO and sulfur products than for oxygen. Poisoning affects fuel cell performance by binding to the catalyst and preventing hydrogen and oxygen from reaching it.

2.4 Fuel Cell Vehicles

The use of PEMFCs in EV and HEV applications has some distinct advantages, namely

- Low temperature operation (60-100 °C) which ensures fast start-up suitable for HEVs
- Highest power density (0.35-0.6 W/cm^2) compared to other fuel cell technologies
- The solid electrolyte does not change and does not evaporate
- Resistant to corrosion as the only liquid is water
- Oxidant is usually air from outside

All these advantages come at a cost in terms of expensive metals and membranes, and a high poisoning probability.

Fuel cell vehicles are seen by many as the ultimate solution to increasing environmental problems. However, running solely on fuel cells has some disadvantages, such as a heavy power unit due to the low power density of the overall system and slow response.

Hybridization is used to overcome these problems, by coupling the fuel cell system primary power source with a high-voltage battery that supports the system during power peaks and heavy acceleration, and stores energy from regenerative braking. Due to its use in peaks, it is also called Peaking Power Source. The battery can also store the excess energy when the fuel cell power is higher than the power demanded by the load. Thus, if an appropriate control strategy is used there is no need to charge the battery externally in this type of configuration. A typical FCHEV configuration is shown in Figure 2.7 [12].

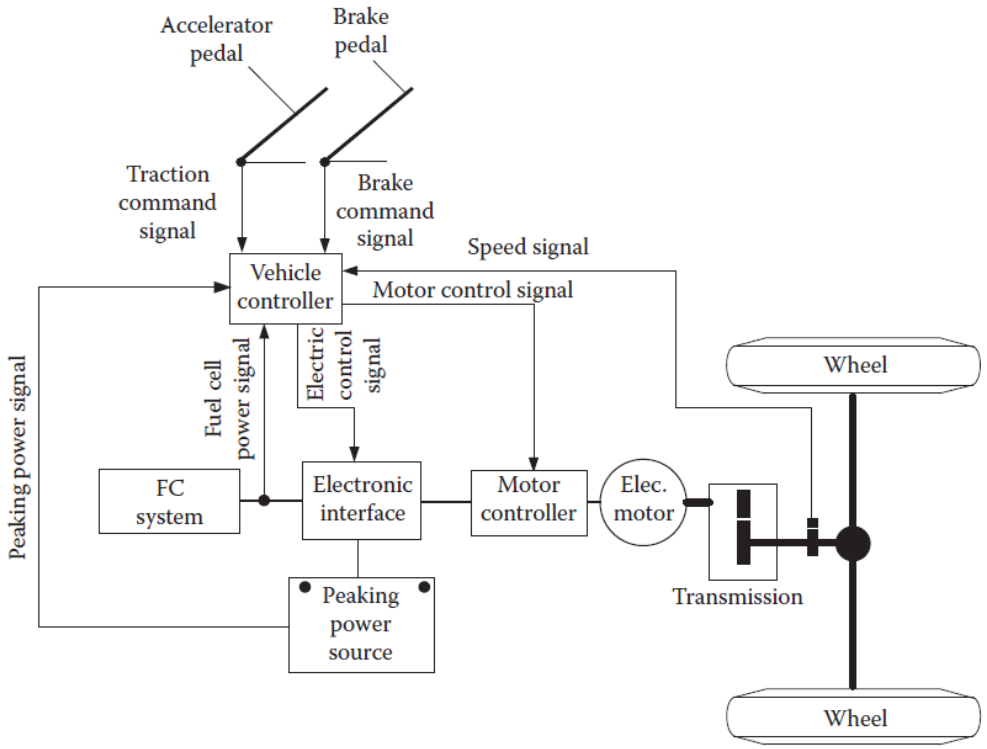

Figure 2.7 Typical FCHEV configuration

At present stage, PEMFC are much more expensive than ICEs, and many improvements need to be made before having a competitive price for this technology. Another issue regards the hydrogen fueling infrastructure: although USA and Europe have made significant investments in recent years, there is still very little infrastructure available worldwide. In theory, hydrocarbon fuel could be stored in a tank and then converted to hydrogen in an on-board reformer. However, many tests have been conducted and it was found that it is easier and more efficient to feed the required hydrogen directly from an external source into a pressurized tank in the vehicle to reduce power losses. This method is known as the non-reforming method.

If FCHEVs are compared with HEVs that use ICE as their primary energy source, the first thing to note is that the former produces no carbon emissions or other pollutants, it is recalled that the analysis here focuses only on TTW emissions. On the other hand, fuel cells also have better efficiency; indeed, the PEMFC efficiency is in the 32-38% range considering the hydrogen HHV and shows the greatest efficiency at low power levels, while ICEs have a low efficiency

on the road, about 20-25%, which is even smaller considering total efficiency of the process. It should also be noted that there are many ways to produce hydrogen from renewable resources that could lead to zero emissions, while petroleum resources are running out.

Compared to pure BEVs, both have zero emissions, but fuel cells offer additional advantages such as a longer range and a much faster refueling time compared to the time necessary to recharge the battery, which may take hours. In addition, the fuel cell can warm up faster in cold weather conditions and reach full power in less time.

A comparative analysis of the costs, that still constitute the biggest obstacle, was carried out in [13] and it is summarized in Table 2.2.

Total Cost	2010	2030 Optimistic	2030 Pessimistic	2030 Average
ICE	2200$	2400$	2530$	2465$
FCEV	47400$	7000$	14060$	10530$
BEV	26700$	6200$	9530$	7865$
FCHEV	19700$	4000$	7330$	5665$

Table 2.2 Costs for different powertrains – Source [13]

2.5 Optimization Algorithm: Dynamic Programming

2.5.1 Philosophy of the method

Dynamic programming (DP) was developed by R. E. Bellman at the end of the 1950s. Dynamic programming is based on Bellman principle of optimality, which states that:

"An optimal policy has the property that regardless of the initial states and decisions that drove the system to the current state, the remaining decisions must constitute an optimal policy with respect to such current state" [14].

This principle limits the number of potentially optimal policies that should be taken into consideration, and also implies that optimal control policies must be determined by moving backward from the final state [15].

Dynamic Programming may be applied to solve optimal control problems involving continuous or discrete nonlinear time-varying systems, with time-varying constraints on the inputs and states. The application of the principle of optimality to solve an optimal control problem is now illustrated [15].

Consider the discrete nonlinear time-varying system described by

$$x_{k+1} = F_k(x_k, u_k) \tag{2.4}$$

and the associated performance index

$$J_0(x_0) = \phi_N(N, x_N) + \sum_{k=0}^{N-1} h_k(x_k, u_k) \tag{2.5}$$

where $[0, N]$ is the time interval of interest. Note that $J_{\mathrm{i}}(x_i)$ depends on the initial time and state.

It is desired to find the control sequence $\{u_0^*, u_1^*, \cdots, u_{N-1}^*\}$ using the principle of optimality. Suppose that both the optimal control sequence $\{u_{k+1}^*, \cdots, u_{N-1}^*\}$ and the optimal cost $J_{k+1}^*(x_{k+1})$ from time $k+1$ to final time N have been computed, for all possible states x_{k+1}. If the arbitrary control input u_k is now applied at time k and taking into account the known optimal control sequence $\{u_{k+1}^*, \cdots, u_{N-1}^*\}$, the resulting cost from time k to time N is given by

$$h_k(x_k, u_k) + J_{k+1}^*(x_{k+1}) \tag{2.6}$$

According to Bellman principle of optimality, the optimal cost from time k to time N is calculated as

$$J_k^*(x_k) = \min_{u_k}\left[h_k(x_k, u_k) + J_{k+1}^*\left(F_k(x_k, u_k)\right)\right] \tag{2.7}$$

and the optimal control u_k^* is the one that minimizes (2.7). Equation (2.7) is the principle of optimality for discrete time systems and enables to optimize over only one control vector at a time by going backward from N. This equation is known as functional equation of dynamic programming and constitutes the basis for the computer implementation of the DP algorithm [15]. Additional constraints may be included, such as requiring that the states and inputs belong to particular admissible sets.

2.5.2 The dpm function: A Matlab implementation of the DP algorithm.

The dpm function developed by Sundström is an efficient Matlab implementation of the DP algorithm used in this work. The type of optimal control problem that can be solved using the dpm function is now formulated [16].

Consider the discrete time-varying nonlinear system described by the state equation

$$x_{k+1} = F_k(x_k, u_k), k = 0,1, \cdots, N-1 \tag{2.8}$$

where $x_k \in \mathcal{X}_k$ is the state variable and $u_k \in \mathcal{U}_k$ is the control signal. In addition, define the control policy $\pi \epsilon \Pi$

$$\pi = \{\lambda_0, \lambda_1, \cdots, \lambda_{N-1}\} \quad 2.9$$

where Π is the set of all admissible policies, and also define the discretized cost function

$$J_\pi(x_0) = g_N(x_N) + \phi_N(x_N) + \sum_{k=0}^{N-1} \left[h_k\left(x_k, \lambda_k(x_k)\right) + \phi_k(x_k)\right] \quad 2.10$$

given π and the initial state $x(0) = x_0$, where $g_N(x_N)$ is the final cost, $h_k\left(x_k, \lambda_k(x_k)\right)$ is the cost of applying the control signal $\lambda_k(x_k)$ at state x_k, and $\phi_N(x_N)$ and $\phi_k(x_k)$ are penalty functions to guarantee that the final state x_N and the intermediate states $x_k, k = 0,1, \cdots, N-1$, respectively, fulfil the corresponding constraints. The optimal control policy π^O is the one that minimizes $J_\pi(x_0)$.

According to the principle of optimality, dpm proceeds backward in time to evaluate the cost function. In order to reduce the computational cost, equally spaced input and state grids are defined and a linear interpolation scheme is used to avoid evaluating the model when proceeding backwards. In other words, in the backward pass the following cost-to-go functions are evaluated in the discretized state-time

i. Step for calculating the end cost

$$\mathcal{J}_\mathrm{N}(x_0) = g_N(x_N) + \phi_N(x_N) \quad 2.11$$

ii. Step for calculating intermediate costs for k=N-1, ⋯,0

$$\mathcal{J}_\mathrm{k}(x^i) = \min_{u_k \in \mathcal{U}_k} \left\{h_k(x^i, u_k) + \phi_k(x^i) + \mathcal{J}_\mathrm{k}\left(F_k(x^i, u_k)\right)\right\} \quad 2.12$$

The optimal control is determined as the argument that minimizes the right-hand side of (2.12) for each x^i at time index k of the discretized state-time space [16].

The result of the backward pass is an optimal control signal map, which is further used in the forward simulation of model (2.8) starting from the initial state x_0, to find the optimal control signal and the optimal state trajectory. In the optimal control signal map, the control signal is only given at discrete points of the state-space grid; therefore, an interpolation must be carried out when the state does not coincide with the points in the grid. The complexity of the DP algorithm is exponential in the number of state and input variables [16].

- Syntax

The syntax and commands for using the dpm function to solve the optimal control problem stated above are described hereafter. The dpm function is called by means of the instruction [16].

```
[res dyn] = dpm(fun,par,grd,prb,options);
```

where `fun` is a handle to the model function, `par` is a user defined parameter structure passed to the model, `grd` is the grid structure, `prb` is the problem structure, and `options` is the option structure. In general, the output of the dpm function are two structures representing the output of the dynamic programming algorithm and the signals from forward simulation of the model using the optimal control input map.

- Inputs

Some of the structures required as inputs by the dpm function are:

1. `prb` structure: Gives the parameters necessary to define the problem. The parameters included in this structure are:

`Ts`	time step (is passed to the model function)
`N`	number of time steps in problem (integer that defines the problem length)
`N0`	(optional) start time index (only used in forward simulation)
`W{.}`	(optional) vectors with length N containing time varying data for the model

2. `grd` structure: Contains all the information about the state and input grids and constraints.

`Nx{.}`	number of grid points in state grid
`Xn{.}.lo`	lower limits for each state (vector for time-variant or scalar for fixed)
`Xn{.}.hi`	upper limits for each state (vector for time-variant or scalar for fixed)
`XN{.}.lo`	final state lower constraints
`XN{.}.hi`	final state upper constraints
`X0{.}`	initial value (only used in forward sim)
`Nu{.}`	number of grid points in input grid
`Un{.}.lo`	(optional) upper limits for each input (vector for time-varying or scalar for fixed)
`Un{.}.hi`	(optional) upper limits for each input (vector for time-varying or scalar for fixed)

3. `options` structure: Defines how to use the algorithm.

`HideWaitbar`	hide waitbars (0/1)
`Warnings`	show warnings (0/1)
`SaveMap`	save cost-to-go map (0/1)
`UseLine`	use boundary line method (0/1)
`FixedGrid`	(used if `UseLine=1`) using the original grid as specified in grd or adjust the grid to the boundary lines (0/1)
`Iter`	(used if `UseLine=1`) maximum number of iterations when inverting model
`Tol`	(used if `UseLine=1`) minimum tolerance when inverting model
`InfCost`	a large cost for infeasible states (`I=1`)
`Minimize`	(optional) minimizing (or maximizing) cost function (0/1) default is minimizing
`InputType`	(optional) string with the same number of characters as number of inputs. Contains the character 'c' if input is continuous or 'd' if discrete (default is all continuous).
`gN{1}`	(optional) Cost matrix at the final time (must be of `size(options.gN{1}) = [grd.Nx{1} grd.Nx{2}... grd.Nx{.}])`

- Outputs

On the other hand, the outputs of the dpm function are two structures, namely res and dyn.

1. `res` structure: Contains the results from the forward simulation of the model when applying the optimal control input map.

`X{.}`	state trajectories
`C{.}`	cost trajectory
`I`	infeasible vector (problem is not solved if there are nonzero elements)
`signals`	structure containing all the signals that were saved in the model function

2. `dyn` structure: Associated with the DP algorithm, the optimal cost-to-go, and the optimal control input map. When the boundary line method is used the `dyn` structure also contains the boundary lines (with the states, inputs, and costs).

`B.hi`	`Xo,Uo{.},Jo` contains the cost, input, and state for the upper boundary line
`B.lo`	`Xo,Uo{.},Jo` contains the cost, input, and state for the lower boundary line
`Jo{.,.}`	optimal cost-to-go (indexed by input number and time index)
`Uo{.,.}`	optimal control input (indexed by input number and time index)

The `dpm` function can be also used only for forward simulation, when the DP output structure `dyn` is precalculated. This can be very useful when changing the initial condition or when increasing the starting time `N0` of the problem. To call the `dpm` function when the DP output structure is already calculated use

```
res = dpm(dyn,fun,par,grd,prb,options);
```

- Model definition

In general, the model function should have the format:

```
function [X, C, I, signals] = mymodel(inp,par)
```

Its inputs are the

1. `inp` structure

`X{.}`	current states (`n+m` dimensional matrix form depending on the number of inputs and state variables)
`U{.}`	current inputs (`n+m` dimensional matrix form depending on the number of inputs and state variables)
`W{.}`	current time-varying data (scalar)
`Ts`	time step

2. `par` structure, which can contain any parameters defined by the user, which are necessary in the model function.

On the other hand, the outputs of the model are

`X{.}`	resulting states after applying `inp.U{.}` at `inp.X{.}` (same size as `inp.X{.}`)
`C{.}`	resulting cost of applying `inp.U{.}` at `inp.X{.}` (same size as `inp.X{.}`)
`I`	set with infeasible combinations (feasible=0, infeasible=1) (same size as `inp.X{.}`)
`signals`	structure with user defined signals (same size as `inp.X{.}`)

2.6 Modeling using Neural Networks

Neural networks (NN) have demonstrated to be an excellent alternative for modeling dynamic systems, due to their capability of approximating any function with arbitrary degree of accuracy [17] and to the excellent performance they exhibit for analyzing, extracting information, and identifying models from large amounts of data.

2.6.1 Definition and elements of a neural network

Nowadays, neural networks are part of the Machine Learning discipline. They were initially developed during the 1940s, trying to mimic the behavior of the human brain. Different definitions of neural networks have been given in the literature ([16],[19],[20]). A neural network consists of various nodes (also known as units or neurons) arranged in layers, which are connected through weights. Neural networks are capable of learning from data through a training process, during which the weights are adjusted in an ordered manner to successfully achieve a desired behavior. Among other tasks, neural networks have been successfully used to identify models of processes with complex dynamics; in general, this complexity is related to features such as nonlinearity, large number of variables, high dimension, uncertainty, among others, or it may simply be impractical to determine models based on physical principles. Indeed, neural networks are capable of approximating any functional relationship between variables, with arbitrary degree of accuracy[17].

A neural network consists of the following elements:

a) Architecture: specifies the number of layers, the size of each layer and how these layers are connected, thus defining how the information flows through the network. Various architectures have been developed, namely feedforward, radial basis functions, support vector machines and recurrent neural networks, among others. Only feedforward neural networks are described in this work, since that is the architecture chosen here for modeling the dynamic systems of interest.

b) Transfer function: specifies how to calculate the output values of the NN as a function of its input values.

c) Training or learning algorithm specifies how the weights are adjusted during the training process. The learning algorithms may be classified in (i) supervised, in which the target values are known, or (ii) unsupervised, in which the target values are not given, and the NN seeks to define clusters or extract probabilistic features from the data.

2.6.2 Feedforward neural network (FFNN)

a) Architecture of the FFNN

It consists of three types of layers of nodes connected in sequence: (i) one input layer, which receives the inputs to the NN, (ii) one output layer, which gives the output of the NN and (iii) one or more hidden layers, located between the input and output layers. The information flows unidirectionally, from input to output layer. Note that the size of the input and output layer are established by the configuration of the data vectors. On the other hand, the number and size of the hidden layers should be chosen by the designer. Regarding this issue, a theorem by Kolmogorov [21] states that one hidden layer is enough to achieve arbitrary precision accuracy; it is recommended to use various hidden layers when it is necessary to reduce the size of such layers for implementation purposes.

Figure 2.8, below, shows a FFNN with n input nodes, m output nodes, and $p + 2$ layers, namely one input layer, p hidden layers and one output layer. Each node of a layer is connected to all nodes of the subsequent layer, and each connection has associated a weight which is adjusted during the training process. Since the input nodes perform no processing on the input values (they just distribute them to the first hidden layer), the input layer will be denoted with the superscript 0. A bias is included for hidden and output nodes. Let us define as $\boldsymbol{W}^{(i)}$ $(i = 0, \cdots, p + 1)$ the matrix that contains the weights between layers $i - 1$ and i, which have J and K nodes, respectively. Then

$$\boldsymbol{W}^{(i)} = \begin{bmatrix} w_{11}^{(i)} & w_{12}^{(i)} & \cdots & w_{1J}^{(i)} \\ w_{21}^{(i)} & w_{22}^{(i)} & \cdots & w_{2J}^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K1}^{(i)} & w_{K2}^{(i)} & \cdots & w_{KJ}^{(i)} \end{bmatrix} \in \mathbb{R}^{K \times J} \tag{2.13}$$

where $w_{kj}^{(i)}$ represents the weight that connects node j in layer $i - 1$ and node k in layer i. On the other hand, let us define as

$$\boldsymbol{b}^{(i)} = \begin{bmatrix} b_1^{(i)} \\ b_2^{(i)} \\ \vdots \\ b_K^{(i)} \end{bmatrix} \tag{2.14}$$

the vector containing the biases of units in layer i.

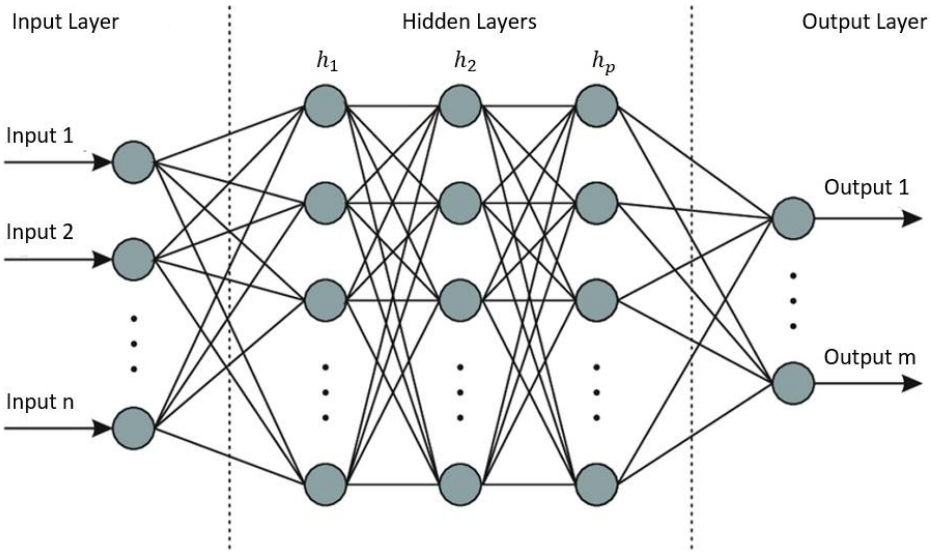

Figure 2.8 FFNN with $p + 2$ layers: one input layer, p hidden layers and one output layer.

b) Transfer function of the FFNN

As stated before, the input units do not carry out any processing, they only receive the inputs and distribute it to the units in the hidden layer. On the other hand, the unit shown in Figure 2.9 is a representation of the processing performed by hidden and output units, whose output y is calculated as

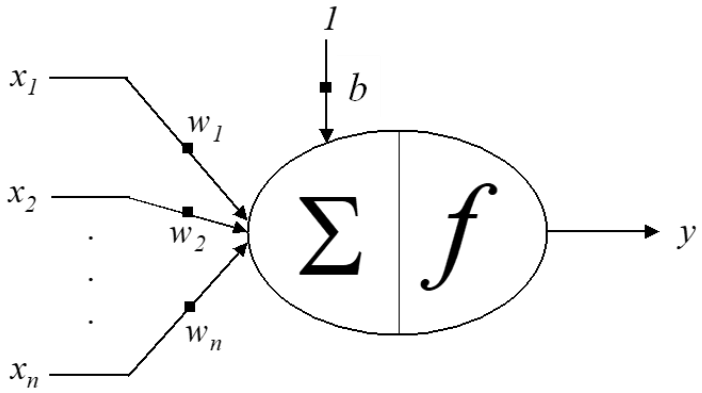

Figure 2.9 Hidden and output units of the FFNN of Figure 2.8

$$y = f(b + x_1 w_1 + x_2 w_2 + \cdots + x_n w_n) \qquad 2.15$$

where x_i $(i = 1, \cdots, n)$ are the inputs, w_i $(i = 1, \cdots, n)$ are the corresponding weights, b is the bias of the unit and f is the activation function of the unit. Note that the bias is represented as a weight whose input is fixed at 1. Therefore, defining the input vector $\boldsymbol{x} = [1 \quad x_1 \quad x_2 \quad \cdots \quad x_n]^T$ and the weight vector $\boldsymbol{w} = [w_0 \quad w_1 \quad w_2 \quad \cdots \quad w_n]^T$, equation (2.15) may be rewritten as

$$y = f(\boldsymbol{x}^T \boldsymbol{w}) \qquad 2.16$$

Common activation functions for the output units are the rectified linear unit (ReLU), the hyperbolic tangent, the sigmoid and the identity. On the other hand, the activation function of the hidden units is required to be nonlinear, and therefore the identity function cannot be chosen. The mathematical definition of these functions, and their corresponding derivatives are shown in Table 2.3.

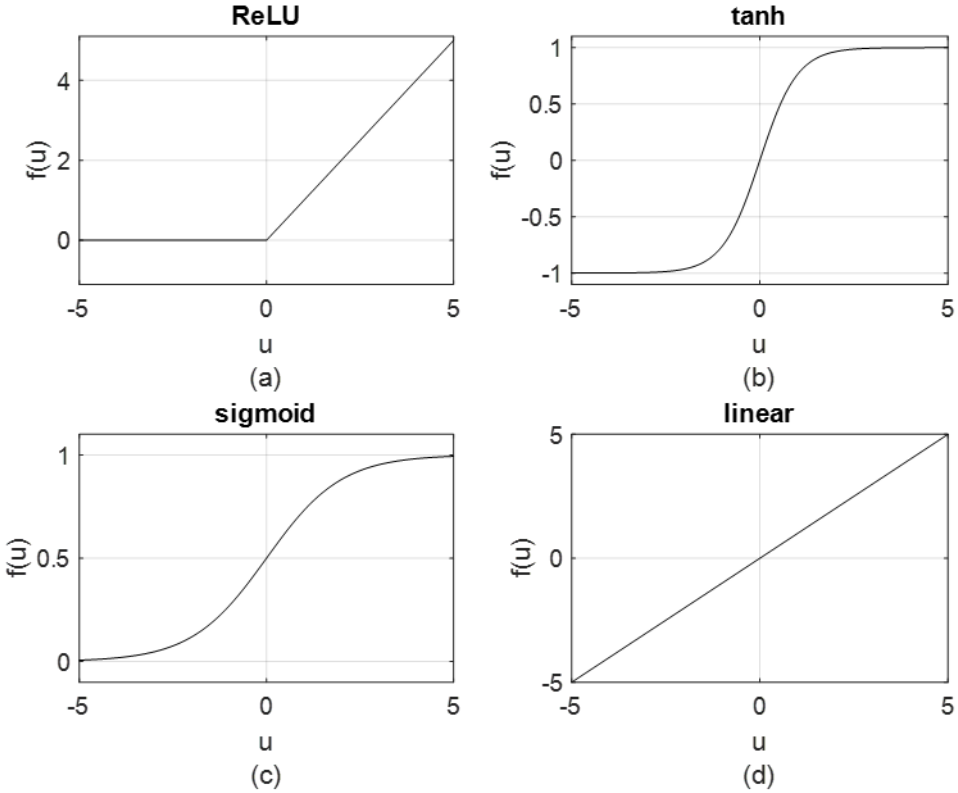

Figure 2.10 Common activation functions: (a) ReLU, (b) hyperbolic tangent, (c) sigmoid and (d) identity

Activation function	Mathematical definition	Derivative
ReLU	$f(u) = \begin{cases} 0 & for\ u < 0 \\ u & for\ u > 0 \end{cases}$	$f(u) = \begin{cases} 0 & for\ u < 0 \\ 1 & for\ u > 0 \end{cases}$
hyperbolic tangent	$f(u) = tanh(u)$	$f'(u) = sech^2(u)$
sigmoid	$f(u) = \frac{1}{1+e^{-u}}$	$f'(u) = \frac{e^{-u}}{(1+e^{-u})^2}$
identity	$f(u) = u$	$f'(u) = 1$

Table 2.3 Mathematical definition and derivative of the activation functions

Then, if the input vector

$$\boldsymbol{X} = [X_1 \quad X_2 \quad X_3 \quad \cdots \quad X_n]^T \tag{2.17}$$

is applied to the FFNN of Figure 2.8, its output vector

$$\widehat{\boldsymbol{Y}}^{(p+1)} = \left[\hat{Y}_1^{(p+1)} \quad \hat{Y}_2^{(p+1)} \quad \cdots \quad \hat{Y}_m^{(p+1)}\right] \tag{2.18}$$

can be calculated by means of the recursion

$$\begin{aligned} \widehat{\boldsymbol{Y}}^{(1)} &= f_1\left(\boldsymbol{W}^{(1)}\boldsymbol{X} + \boldsymbol{b}^{(1)}\right) \\ \widehat{\boldsymbol{Y}}^{(2)} &= f_2\left(\boldsymbol{W}^{(2)}\widehat{\boldsymbol{Y}}^{(1)} + \boldsymbol{b}^{(2)}\right) \\ &\vdots \\ \widehat{\boldsymbol{Y}}^{(p)} &= f_p\left(\boldsymbol{W}^{(p)}\widehat{\boldsymbol{Y}}^{(p-1)} + \boldsymbol{b}^{(p)}\right) \\ \widehat{\boldsymbol{Y}}^{(p+1)} &= f_{p+1}\left(\boldsymbol{W}^{(p+1)}\widehat{\boldsymbol{Y}}^{(p)} + \boldsymbol{b}^{(p+1)}\right) \end{aligned} \tag{2.19}$$

where f_i represents the activation function of the units of layer i and $\widehat{\boldsymbol{Y}}^{(i)}$ is the output vector of the same layer.

c) Training algorithm

Various algorithms have been developed for training FFNNs. The backpropagation training algorithm, developed by Rummelhart et al. in 1986, constituted a significant step that boosted the research and practical applications of feedforward neural networks. The backpropagation algorithm uses the steepest descent method, one of the most ancient minimization techniques, to adjust the NN weights to minimize the mean square estimation error over the target values of the outputs in the training set. The steepest descent method is an iterative procedure whose principle is very simple: standing at any point a of the function being minimized, the subsequent point b is determined moving a small step in the direction opposite to the gradient at a, which represents the direction along which the function decreases at a faster rate.

The backpropagation algorithm has two versions, namely off-line and on-line. In the former the weights are updated after a complete pass through all training vectors, while in the latter the weights are updated after each training vector is presented. In this work, the off-line version was used for training all neural networks; hence, only this version is described here.

Given a set of S training vectors

Input				**Output**			
X_1^1	X_2^1	$\cdots$	X_n^1	Y_1^1	Y_2^1	$\cdots$	Y_m^1
X_1^2	X_2^2	$\cdots$	X_n^2	Y_1^2	Y_2^2	$\cdots$	Y_m^2
	$\vdots$				$\vdots$		
X_1^S	X_2^S	$\cdots$	X_n^S	Y_1^S	Y_2^S	$\cdots$	Y_m^S

the objective function to be minimized in the off-line version of the backpropagation algorithm is given by

$$E = \frac{1}{2}\sum_{s=1}^{S}\sum_{j=1}^{m}\left(Y_j^s - \hat{Y}_j^s\right)^2 \qquad 2.20$$

where superscript s identifies the training vector and subscript j refers to the output unit, Y_j^s is the target value of the jth output in the sth training vector, and $\hat{Y}_j^s$ is the network estimate for Y_j^s.

For the FFNN of Figure 2.8, the procedure to implement the backpropagation algorithm is the following: Set matrix $\boldsymbol{\Delta}^{(i)} = \mathbf{0}$, where the term $\Delta_{ij}^{(i)}$ will be further used to update weight $w_{kj}^{(i)}$ that connects node j in layer $i-1$ and node k in layer i. Note that $\boldsymbol{\Delta}^{(i)}$ has the same size as $\boldsymbol{W}^{(i)}$. For $s = 1, \cdots, S$

- Set $\widehat{\boldsymbol{Y}}^{(0,s)} = \boldsymbol{X}^{(s)} = [X_1^s \quad X_2^s \quad \cdots \quad X_n^s]^{\boldsymbol{T}}$
- Perform forward propagation according to equations (2.19), to compute $\widehat{\boldsymbol{Y}}^{(i,s)}$ for $i = 1, \cdots, p+1$
- For the output layer, compute the error term $\boldsymbol{\delta}^{(p+1,s)} = \widehat{\boldsymbol{Y}}^{(p+1,s)}$ corresponding to the s-th training vector
- Perform back propagation of the error terms corresponding to the s-th training vector, computing $\boldsymbol{\delta}^{(p,s)}, \boldsymbol{\delta}^{(p-1,s)}, \cdots, \boldsymbol{\delta}^{(1,s)}$ given by the following equation (2.21) where $\odot$ is the pointwise product and f_i' is the derivative of the activation function of layer i

$$\boldsymbol{\delta}^{(i,s)} = \left[\boldsymbol{W}^{(i)}\right]^T \boldsymbol{\delta}^{(i+1,s)} \odot f_i'\left(\widehat{\boldsymbol{Y}}^{(i,s)}\right) \quad \text{for } i = 1, \cdots, p \qquad 2.21$$

- Compute

$$\boldsymbol{\Delta}^{(i)} = \boldsymbol{\Delta}^{(i)} + \boldsymbol{\delta}^{(i+1,s)}\left[\widehat{\boldsymbol{Y}}^{(i,s)}\right]^T \qquad 2.22$$

- Update weight matrix $\boldsymbol{W}^{(i)}$ between layers $i-1$ and i by means of equation (2.23) where t denotes the epoch number and η is the learning rate. Remember that one epoch is a pass through all vectors in the training set.

$$\boldsymbol{W}^{(i)}(t) = \boldsymbol{W}^{(i)}(t-1) - \eta \cdot \frac{1}{S} \cdot \boldsymbol{\Delta}^{(i)} \qquad 2.23$$

In general, the backpropagation algorithm converges slowly. As an alternative to overcome this drawback, different variants of the backpropagation algorithm [22] have been developed which use more efficient minimization approaches.

One of these variants is the Levenberg-Marquardt algorithm [22], which combines steepest descent and Newton methods. Newton method is another iterative minimization procedure, in which a second order approximation of the error surface is determined at current point a using a Taylor series expansion (truncated to eliminate terms of order greater than 2), and the subsequent point b is equal to the point corresponding to the minimum of such quadratic approximated surface. In the Levenberg-Marquardt iterative algorithm, the update term is a linear combination of the updates of the steepest descent and Newton methods. In particular, the update of the steepest descent method is weighted by a variable term μ that decreases as the search progresses; in other words, the steepest descent method dominates when the search is far from the minimum, while Newton method predominates when the search is approaching the minimum.

All gradient-based methods, such as the backpropagation and Levenberg-Marquardt algorithms, are susceptible of getting trapped in local minima. A way to cope with this issue is to carry out various trainings processes with different initial weights and select the one that exhibits the best performance.

An important aspect that should be taken into account during the NN training process is to avoid overtraining. An overtrained network yields an outstanding performance on the training set, but its performance degrades significantly for data vectors outside such set. In this case it is considered that the NN has "memorized" the training set but has not captured the relationship between input and output variables, which is the ultimate objective. An NN with good "generalization capability" is one that has effectively "learned" the underlying relationship between input and outputs.

A common practice to avoid overtraining involves splitting the available data in three sets:

i. **Train set:** is used to train the NN

ii. **Validation set:** is used to determine when to stop the training process.

iii. **Test set:** is used to measure the generalization capability of the NN.

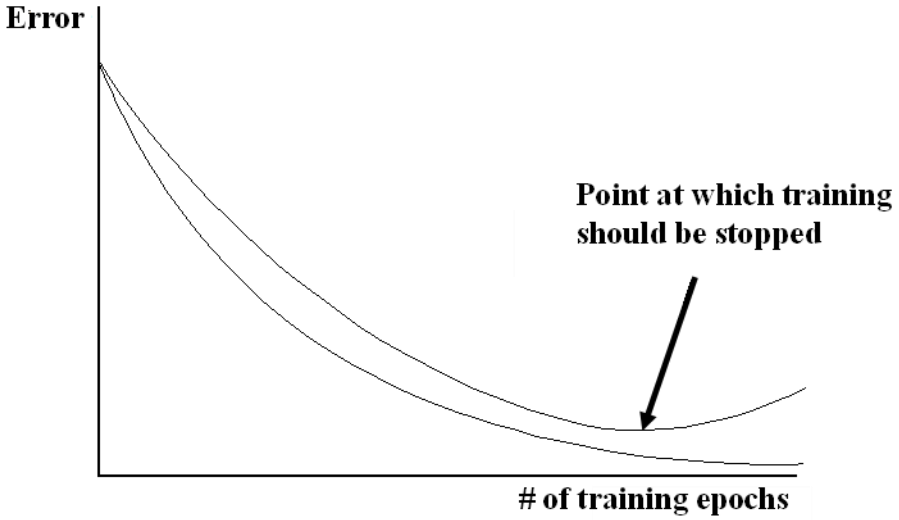

Figure 2.11 Typical behavior of the error on the training (solid) and validation (dotted) sets during a training process

Figure above shows typical error curves on the training and validation sets during a training process. In general, such process starts with a set of random weights; therefore, it is expected that both errors are high during initial training epochs. As training evolves in time, the error of the NN on both sets progressively decreases until a point in which the error on the validation set reaches a minimum: at this epoch the training should be stopped because the NN has attained the best performance on the validation set, and therefore the best generalization capability. Further training beyond this epoch will result in a training error that keeps decreasing, i.e., the network is starting to "memorize" the training set.

Figure 2.11 illustrates the ideal stopping situation of a training process, known as validation stop. However, gradient based algorithms may get stalled in local minima; when this happens, the errors may tend to remain constant or decrease very slowly even after many epochs have elapsed, and the training process may take much longer than desired. To avoid this undesirable situation, additional stopping criterion are established, such as (i) maximum number of epochs, (ii) a minimum value of MSE on the training set and (iii) a minimum variation of the MSE between two consecutive epochs.

Prior to training the NN, all elements in the input and target vectors in the training set are linearly normalized to span the interval $[-1,1]$, using the formula

$$\theta = -1 + 2\frac{\Theta - \Theta_{min}}{\Theta_{max} - \Theta_{min}} \qquad 2.24$$

where Θ and θ represent the original and normalized values, respectively, and Θ_{min} and Θ_{max} are the corresponding lower and upper bounds, respectively. The purpose of normalizing the

data is to have all variables vary in the same range. Then, the corresponding denormalized values can be obtained by means of the formula

$$\Theta = (\theta + 1)\frac{\Theta_{max} - \Theta_{min}}{2} + \Theta_{min} \qquad 2.25$$

2.6.3 Modeling of dynamic systems using feedforward neural networks

Models of dynamic systems are useful for implementing advanced strategies to control such systems. Since FFNNs have universal approximation capabilities, they have been successfully used for identifying models of dynamic systems. It is important to mention that NN-based models constitute a "very flexible" type of nonlinear regression models; they belong to the class of black box models, because they reproduce the input-output behavior of the process, but their parameters (weights and biases) do not have physical meaning.

A structure used very frequently to describe dynamic systems with one output and r inputs is the NN-ARX (Neural Network-based Auto Regressive with eXogenous Inputs) model. This model estimates the output $y(k)$ at time instant k as a function of p previous values of the output and q previous values of each of the inputs, i.e.

$$\hat{y}(k) = NN[y(k-1), \cdots, y(k-p), u_1(k-1), \cdots, u_1(k-q), \cdots u_r(k-1), \cdots, u_r(k-q)] \qquad 2.26$$

where $y(k)$ is the output and $u_l(k)$ $(l = 1, \cdots, r)$ are the inputs, all at time instant k, p and q are the orders of the model and NN represents the nonlinear function implemented by the FFNN. Note that the NN-ARX model (2.26), which is illustrated in Figure 2.12, has one output and $p + r \cdot q$ inputs.

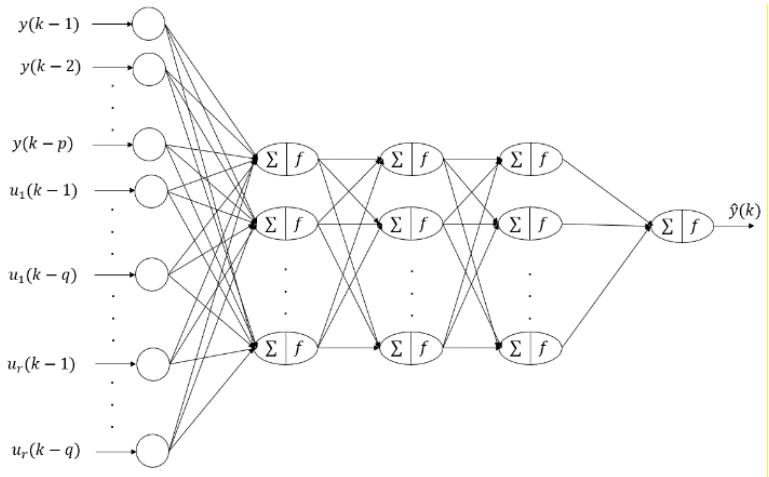

Figure 2.12 NN-ARX model with one output and $p + r \cdot q$ inputs

The procedure for identifying a NN-ARX model of a dynamic system with one output and r inputs is explained hereafter. In this work, this procedure was implemented using Matlab and its Neural Networks Toolbox.

Given a data set containing S vectors

$$\begin{matrix} y(1) & u_1(1) & u_2(1) & \cdots & u_r(1) \\ y(2) & u_1(2) & u_2(2) & \cdots & u_r(2) \\ \vdots & \vdots & \vdots & & \vdots \\ y(S) & u_1(S) & u_2(S) & \cdots & u_r(S) \end{matrix}$$

the procedure for identifying an NN-ARX model (2.26) of the dynamic system comprises the following steps:

1. Normalize the output and input data using formula (2.24). For example, considering variable Y, with maximum and minimum values defined as

   ```
   Y_max = 100;
   Y_min = -100;
   ```

   its normalized values can be calculated as

   ```
   Y_norm = -1+2*(Y-Y_min)/(Y_max-Y_min);
   ```

2. Choose the orders p and q of the model. There is no deterministic procedure established for this selection. A trial-and-error approach is used here, and the model with the best performance is chosen.

   For example, the instructions

   ```
   ndelay_out = 2;
   ```

   ```
   ndelay_inp = 2;
   ```

   are used to configure an NN-ARX model with two previous values of the output (p=2) and two previous values of each of the r inputs (q=2). Since both delays are equal, a single delay

   ```
   ndelay = 2;
   ```

   is defined.

3. Construct the data that will be used for training and testing the NN model, according to the orders chosen. It is important to remember that the original dynamic system has one output and r inputs, but the NN-ARX model has one output and $p + r \cdot q$ inputs. For example, the first vector of this data will have the form

NN Output | NN Inputs

$$y(p) \qquad y(p-1) \cdots y(1)\ u_1(p-1) \cdots u_1(1)\ u_2(p-1) \cdots u_2(1) \cdots u_r(p-1) \cdots u_r(1)$$

For example, for a system with output `Y` and inputs `X1` and `X2`, the following instructions construct the data vectors with the delay previously defined

```
Y_XNN = [];
X1_XNN = [];
X2_XNN = [];
for i=1:ndelay
    Y_XNN = [Y_XNN;Y_norm(1,ndelay+1-i:end-i)];
    X1_XNN = [X1_XNN;X1_norm(1,ndelay+1-i:end-i)];
    X2_XNN = [X2_XNN;X2_norm(1,ndelay+1-i:end-i)];
end
XNN = [Y_XNN;X1_XNN;X2_XNN];
YNN = Y(ndelay+1:end);
```

The data set `[XNN,YNN]` has been constructed.

4. Split the data into training, validation and tests sets. First, the data vectors obtained in the previous step were randomly shuffled, to guarantee that the three sets contain vectors corresponding to all possible operating conditions of the process.

   The following instructions are used to implements the tasks specified in this step

```
S=length(XNN);
Ish=randperm(S);
Xsh=[];
Ysh=[];

for i=1:S
    Xsh=[Xsh XNN(:,Ish(i))];
    Ysh=[Ysh YNN(:,Ish(i))];
end

FNN = 0.8;
Ntrain = floor(FNN*S);
Xtrain = Xsh(:,1:Ntrain);
Ytrain = Ysh(:,1:Ntrain);
```

```
Xtest = Xsh(:,Ntrain+1:S);
Ytest = Ysh(:,Ntrain+1:S);
```

Specifically, the training set `[Xtrain,Ytrain]` and the testing set `[Xtest,Ytest]` are created. Note that in this case, a subset of 80 % of the data vectors is used for training the network; the NN Toolbox will further split this subset into training and validation sets. The remaining 20 % of the data vectors are reserved for testing the network.

5. Define the architecture of the FFNN, which involves choosing the number and size of the hidden layers. The instruction

   ```
   net = feedforwardnet([10 10],'trainlm');
   ```

   creates a FFNN called net with two hidden layers, each with 10 units; the FFNN will be trained using the Levenberg-Marquardt algorithm (`'trainlm'`)

6. Train the FFNN using the Levenberg-Marquardt. Prior to training, set the stopping criteria.

   The piece of code

   ```
   iter = 2e3;
   net.trainParam.epochs = iter;
   net = train(net,Xsh,Ysh);
   ```

   sets the maximum number of iterations in `2000` and trains the `net`.

When the `train` command is executed, the NN Toolbox displays the dialog box shown in Figure 2.13. In the `Neural Network` section on top, the dialog box shows the NN architecture.

Then, in the `Algorithm` section it shows that (i) the data was randomly split into training and validation vectors, (ii) the Levenberg-Marquardt (LM) algorithm was used for training and (iii) the root mean square error (RMSE) was used as performance measure.

The `Progress` section states that the training stopped after 645 iterations (epochs), which means that it did not reach the maximum number of epochs (2000). This is confirmed by the `Validation Stop` green check at the bottom of the dialog box, which indicates that the training stopped when the minimum RMSE in the validation data set was achieved. This section also shows that:

(i) the training took 46 seconds

(ii) the final RMSE was `2.56e-05`

(iii) the final value of gradient was `0.000143`

(iv) the final value of the μ parameter in the LM algorithm was `1e-07`. Remember that this parameter weights the steepest descent update term; hence, this indicates that Newton's method was dominating the search at the end.

(v) `50` validation checks were performed.

At last, the buttons available in the `Progress` section are used to generate and display different performance plots.

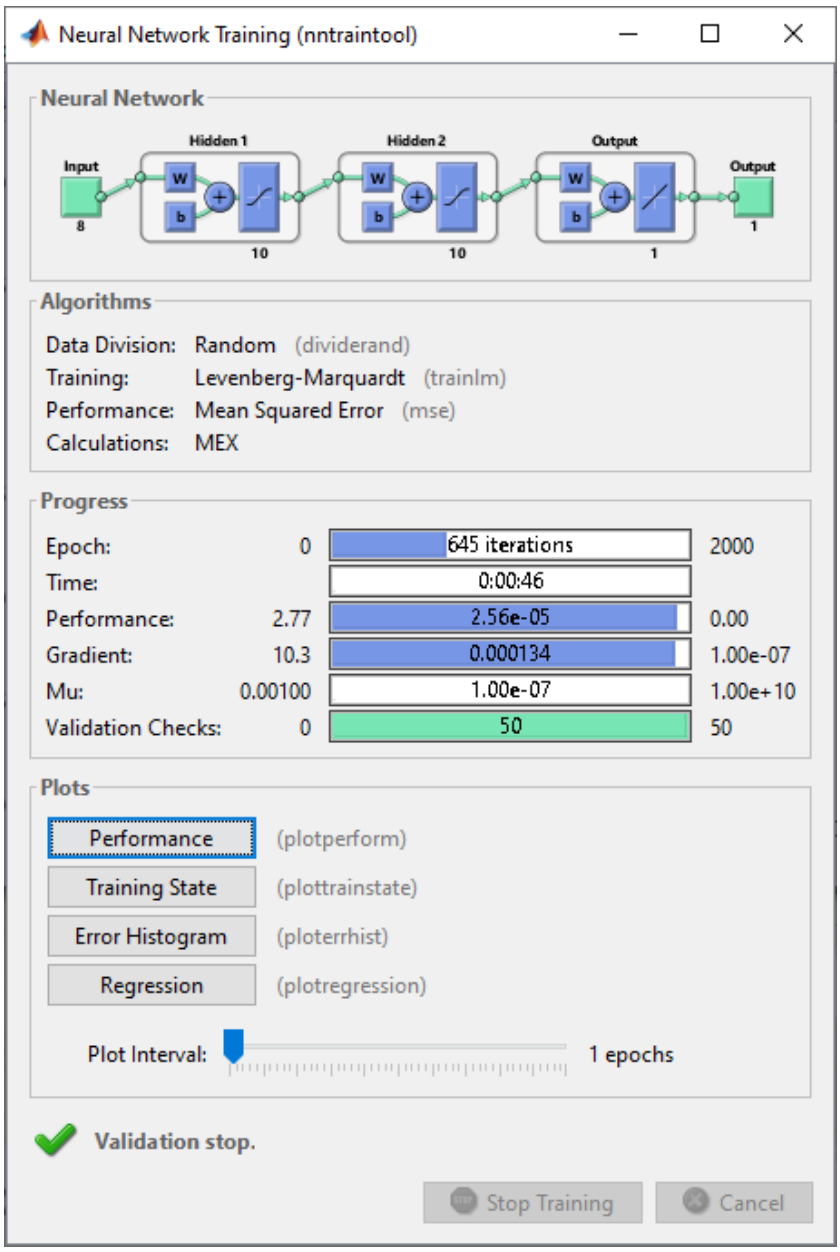

Figure 2.13 Dialog box displayed by the NN Toolbox to illustrate the training process

Finally, test the performance of the FFNN, i.e., its generalization capability. Use the test set for this purpose.

The instructions

```
Ytest_est = net(Xtest);

Y_test = (Ytest+1)*(Y_max-Y_min)/2+Y_min;

Ytest_dnorm = (Ytest_est+1)*(Y_max-Y_min)/2+Y_min;
```

are used to assess the performance of the FFNN on the test set. In particular, the net command is used to determine the response of the network for the test set, and the values determined are further denormalized.

This process is repeated until an NN-ARX model with satisfactory performance is obtained.

3 Modeling of the vehicle and its components

3.1 Drive Cycle Analysis

Legislators worldwide are faced with the problem of defining standard conditions and tests for evaluating vehicle emissions prior to market approval. To this end, standardized driving cycles are used to enable that emissions tests are carried out in a reproducible manner.

As mentioned in Chapter 2, this thesis focuses only on longitudinal dynamics as it adequately represents real-world behavior when considering energy management strategies and fuel consumption evaluation. Thus, the driving cycles used are represented only by their longitudinal speeds and accelerations. Based on the driving cycle formulation, two main categories can be considered: modal driving cycles, which consist of constant speed and acceleration periods, and transient driving cycles, which include a wider range and variations of both speed and acceleration [23]. Modal cycles are usually adopted because they facilitate performing tests using a dynamometer. However, they are not able to adequately represent a real-world scenario. Therefore, transient cycles are preferred when a better representation of fuel consumption is required, because they correspond to the simulation of typical road route.

In Europe, the preferred test was a modal driving cycle known as the NEDC (New European Driving Cycle). But after the Dieselgate scandal, in which it emerged that carmakers were manipulating results, the NEDC was replaced by the WLTP (Worldwide harmonized Light Test Procedure) which offers a more realistic representation of the real world. As of September 1, 2018, WLTP is the only driving cycle accepted for testing vehicle performance in Europe. Depending on the Power/Weight ratio Three different classes of WLTP are acknowledged; the WLTP-Class 3, which is illustrated in Figure 3.1, is used for the FCHEV considered in this work.

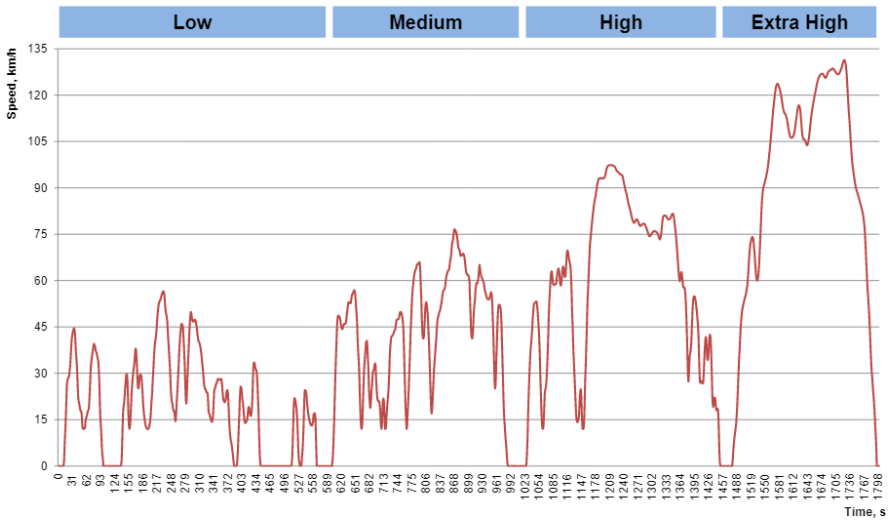

Figure 3.1 WLTP-Class 3 Drive Cycle

On the other hand, the cycles used in the US for assessment procedures are the FTP75 (Federal Test Procedure) shown in Figure 3.2, and the HWFET (Highway Fuel Economy Test) shown in Figure 3.2. Both are transient cycles intended to represent real-world driving scenarios. The former is used to simulate city driving conditions, including frequent start-stop operations, while the latter is used for highway driving.

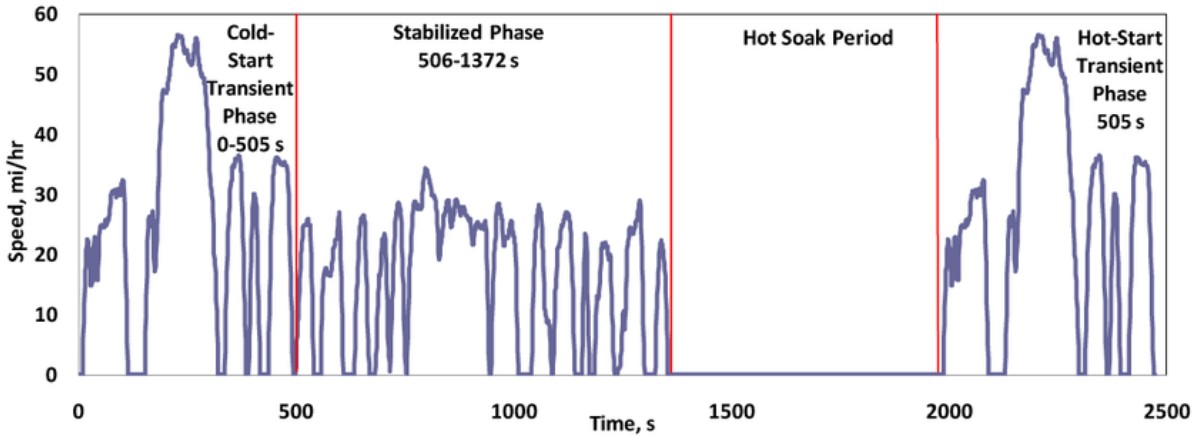

Figure 3.2 FTP75 Drive Cycle – Source [24]

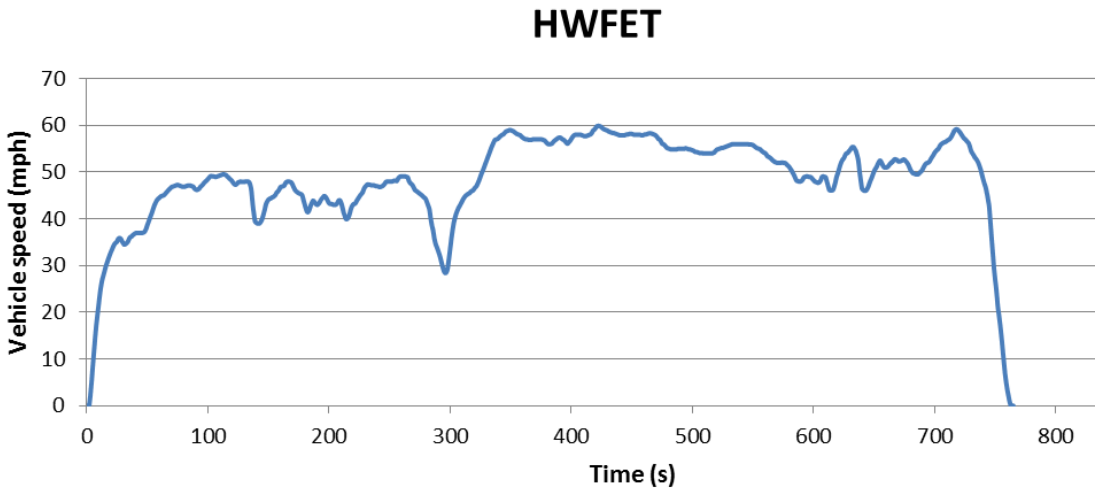

Figure 3.3 HWFET Drive Cycle

In the highway test, the average speed is higher and relatively constant with respect to the urban scenario, having a complete stop only at the end of the cycle. The FTP75 and the HWFET driving cycles are also used in this work

3.2 Toyota Mirai: Vehicle Architecture and Parameters

The state of the art in fuel cell vehicles is represented by the Toyota Mirai, which was launched in 2014 and its performance has been improving ever since. In 2021 it set the record for the longest autonomous range, a total of 1360 km on a tank of 5.65 kg of hydrogen. Therefore, its architecture and parameters are taken as reference to build the model, and test the controller performance in an application as close as possible to reality.

Figure 3.4 illustrates the architecture and components of the Toyota Mirai 2017:

1. Fuel cell stack
2. Fuel cell boost converter
3. Battery
4. High-pressure hydrogen tank
5. Motor
6. Power control unit
7. Auxiliary components

Figure 3.4 Toyota Mirai 2017 architecture and components

The components of the 2021 Mirai whose specifications are available from Toyota datasheets are used in the model created in this work. Typical values are used for the remaining components. Table 3.1 summarizes the data used and compares it with the values in the datasheets.

Component	Mirai 2021	Model
Vehicle		
Mass	1850 kg	1500 kg
Frontal Area	N/A	2.81 m^2
Drag Coefficient	N/A	0.29
Wheel	P235/55R19	0.35 m radius
Battery	Lithium-ion	Lithium-ion
Power Output	310.8 V	303.9 V
Capacity	4.0 Ah	5.3 Ah
Electric Motor	Permanent Magnet AC sync	Permanent Magnet AC sync
Max Power	134kW	
Max Torque	300 Nm	300 Nm
Fuel Cell Stack	PEMFC	PEMFC
Max Power	128kW	128 kW
Number of Cells	330	400
Output Density	5.4 kW/kg	

Table 3.1 Comparison Toyota Mirai and model data

3.3 Dynamic Model

The first model that needs to be modified to reproduce the behavior of the 2021 Toyota Mirai is the dynamic model. As introduced in Chapter 2, dynamic models account for transients and calculate the states of each variable using very small time steps.

The model used in this thesis was developed by the Matlab staff using Simulink and Simscape libraries [25]. Simscape is a physical modeling tool in the Simulink environment, that can be used to create physical models of components based on physical connections that can be directly integrated into block diagrams and other modeling paradigms. Signal flow in Simulink is unidirectional, whereas in Simscape signal flow between blocks is bidirectional. This is the difference between quasi-static and dynamic models presented in the previous chapters.

The fuel cell and battery are connected in a DC electrical network to the motor. The control system determines how much power should be drawn from the battery and from the fuel cell. When braking, energy is fed back into the battery to recharge it. A thermal system, modeled as a fluid network, controls the temperature of the battery, DC-DC converters and motor. Figure 3.5 represents the powertrain subsystem of the Simscape model.

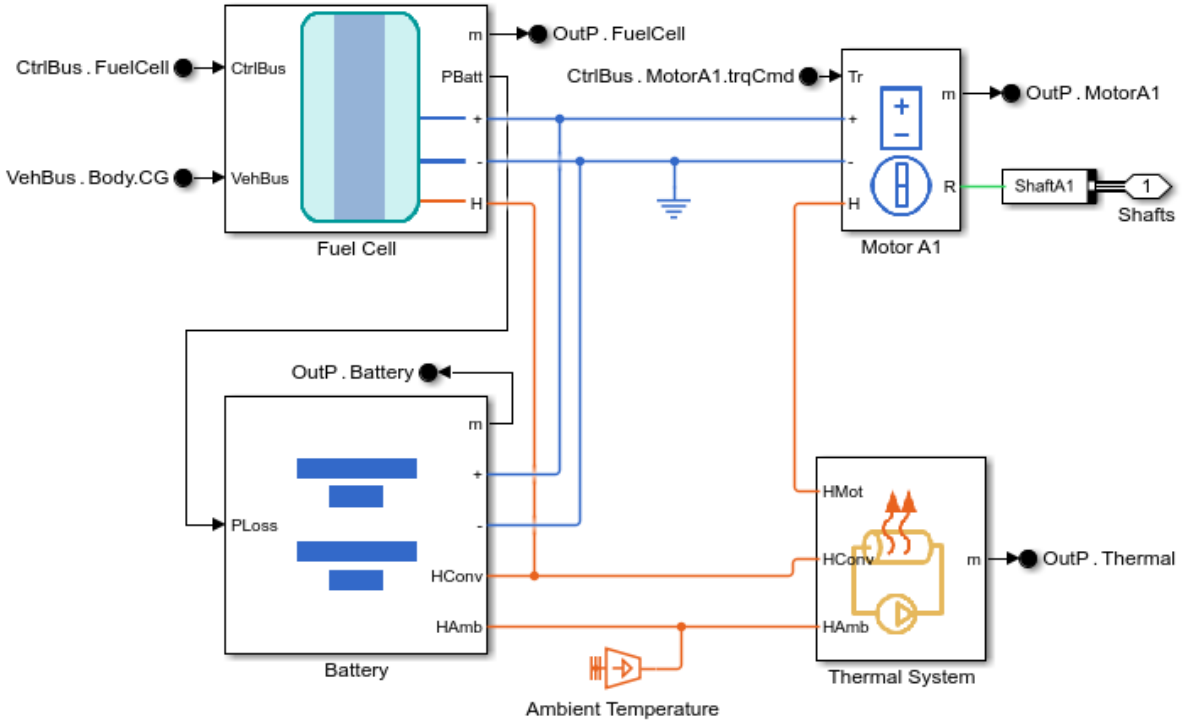

Figure 3.5 Powertrain subsystem of the Simscape Model - Source [18]

3.3.1 Vehicle Dynamics

Since this is a forward-facing dynamic model, there is a driver block implemented as a PI controller. This controller generates pedal commands from the error in the speed, i.e., the difference between the reference speed and the vehicle actual speed that is fed back. The acceleration pedal is translated into a torque command for the EM, which is connected to the wheel axle via the driveline block.

Tires are modeled using the Pacejka's formula [26], also known as "magic formula" since there is no physical basis for the structure of the equations chosen, but they fit a wide range of tire designs and operating conditions.

The longitudinal force is described as

$$F_y = D \cdot \sin\left[C \cdot \arctan\{B \cdot k - E \cdot (B \cdot k - \arctan(Bk)\}\right] \quad 3.1$$

where B, C, D and E are four coefficients that characterize the tire and k is the pure longitudinal slip defined as the ratio between the wheel slip velocity and the wheel hub longitudinal velocity, i.e.

$$k = \frac{V_{sx}}{|V_x|} = \frac{r_w\Omega - V_x}{|V_x|} \quad 3.2$$

This block also takes into account the inertia effect and the rolling resistance of the tires. It is connected to the vehicle body through the mechanical translational conserving port for the wheel hub, through which the thrust developed by the tire is transmitted to the vehicle. The body block takes into account body mass, air resistance, road slope and weight distribution between the axles due to the acceleration and the road profile. Finally, the brakes are represented as a cylinder that applies pressure to one or more pads that may contact the rotor of the shaft. The pressure of the cylinder causes the pads to exert a frictional torque on the shaft.

3.3.2 Electric Motor

The EM is modeled as a brushless motor, such as a Permanent Magnet Synchronous Motor (PSMS), which only operates within the range of torques and speed that are defined by the torque-speed envelope. Figure 3.6. represents a typical torque-speed envelope.

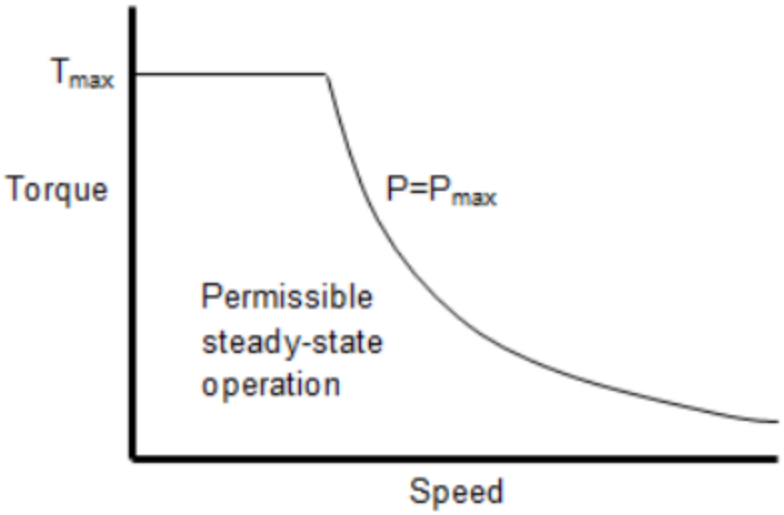

Figure 3.6 Torque-Speed curve for general EM

The EM operates in torque control mode, receiving the commanded torque from the control block and defining the possible range of operation through interpolation with the angular velocity of the motor. Electrical losses are parameterized by a tabulated efficiency data, as a function of speed and torque.

The torque generated is passed through the shaft and the drive block, which includes the transmission block (with a single gear ratio of 5) and the differential, to finally reach the wheel axle and transmit the desired torque.

3.3.3 Battery

The battery is modeled through a table-based block as a function of State of Charge (SOC) and temperature. The battery equivalent circuit, which is shown in Figure 3.7, consists of the fundamental battery model, the self-discharge resistor R_{SD}, the charge dynamics model (optional), and the series resistor R_0.

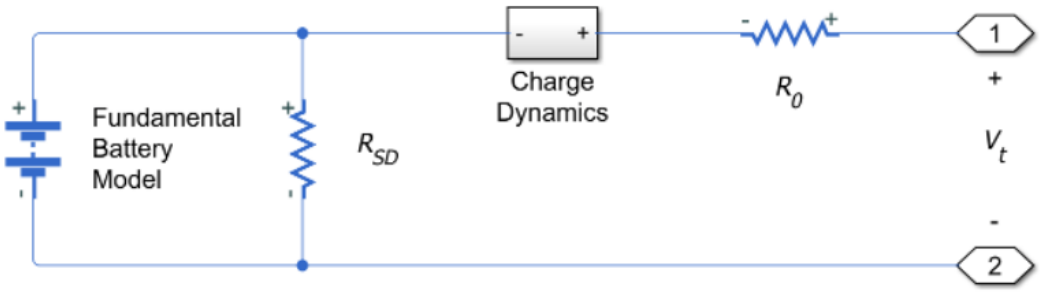

Figure 3.7 Battery Simscape model equivalent circuit

The block computes the no-load voltage, or the voltage across the fundamental battery model, using interpolation, i.e., as $v_0 = v_0(SOC, T)$, where the SOC is given as the ratio of current charge to nominal battery capacity q_{nom} specified in Ampere-hour rating

$$SOC(t) = \frac{1}{q_{nom}(T,n)} \int \left(i(t) - \frac{V_{open}(T,n,t)}{R_{SD}(T,n)} \right) dt \qquad 3.3$$

where n is the current number of battery cycles and T is the battery temperature. The block also models the series resistance R_0 as a function of SOC and T.

Terminals 1 and 2 are connected to a DC/DC converter that regulates the voltage on the load side and provides bidirectional current flow for regenerative braking. The equivalent circuit of the DC/DC converter is shown in Figure 3.8, where P_{fixed} represents a constant power loss that is independent of the load and R_{out} represents losses that increase with load current. The required current is drawn from the supply side to balance the input power, output power and losses, and

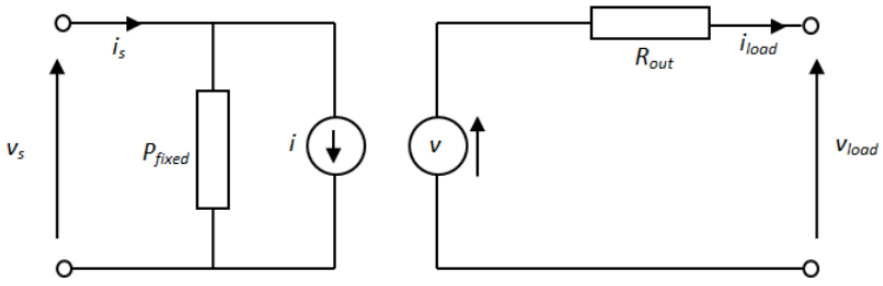

Figure 3.8 DC/DC converter equivalent circuit

$$v = v_{ref} - i_{load}D + i_{load}R_{out} \qquad 3.4$$

where the voltage reference of the motor is 650V and D is the output voltage drop for the output current. The value of the current source, i, is calculated so that the power flowing into the converter is equal to the sum of the power flowing out plus the converter losses.

3.3.4 Fuel Cell

The fuel cell subsystem is not modeled with the polarization curve, but capturing the flow of oxygen, hydrogen, nitrogen and water in a custom Simscape domain, as shown in Figure 3.9. The equations for the reactions and the heat generated are implemented in the Simscape language. A thermal management system keeps the fuel cell at the optimal operating temperature, set to 80 °C.

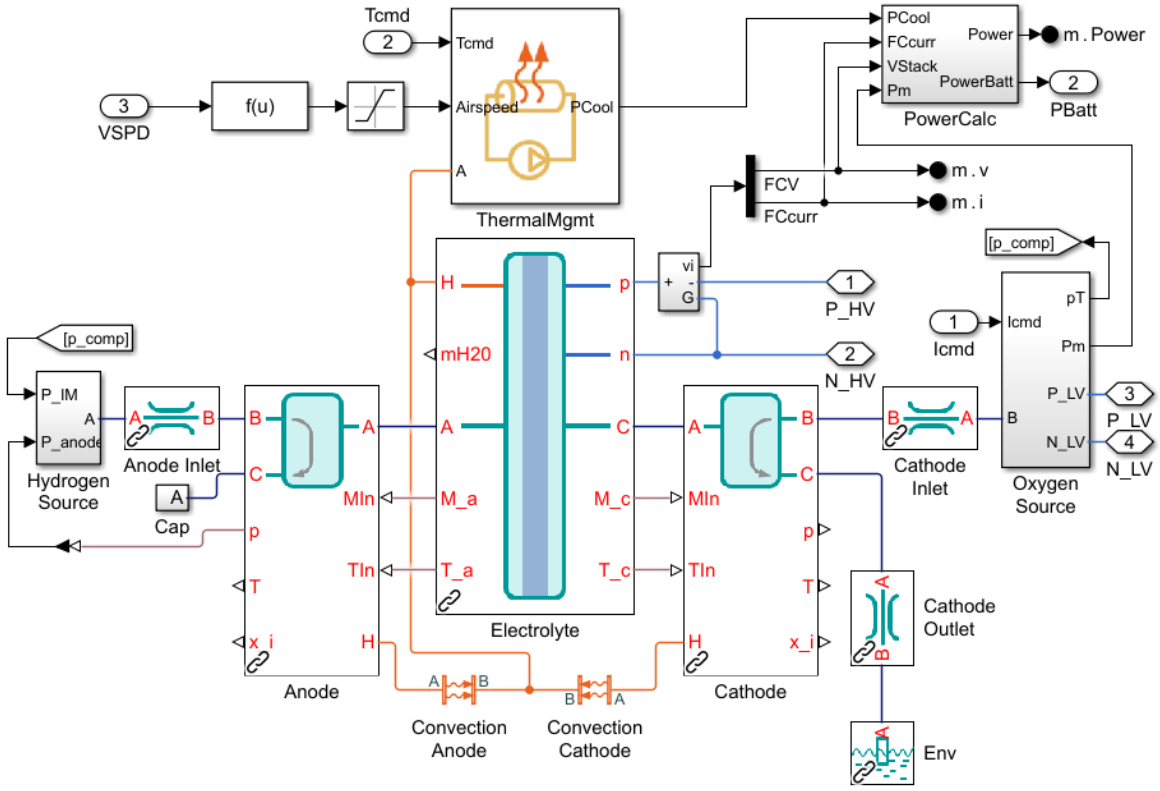

Figure 3.9 Fuel Cell Subsystem – Source [18]

Commanded current coming from the control block enters the oxygen source block, which comprises the humidifier, chiller and compressor block. Current is multiplied by 1.1 to account for the typical value of 10% losses in the compressor (as stated in chapter 2) and the pressure of the compressor is measured and sent to the hydrogen source block.

Here the mass flow rate block guarantees the mass flow by increasing the pressure difference $p_{comp} - p_{anode}$ to an appropriate value, taking hydrogen from the fuel tank.

The mass flow rate determined is used by the electrolyte block to compute analytically the resulting power output as the product of current and voltage. Reactant flow rates can be described as

$$m_{H2} = \frac{i}{2F} M_{H2}; \quad m_{O2} = \frac{i}{4F} M_{O2}; \quad m_{H2O} = \frac{i}{2F} M_{H2O} \tag{3.5}$$

where m are the flow rates that measure the reactants consumed in the cell given in g/s, i is the current, F is Faraday's constant and M are the molar masses. Afterwards the pressure of reactants involved is obtained, and Nernst Voltage is computed as

$$E_T = E_T^0 + \left(\frac{RT}{nF}\right) * \ln \frac{p_{H2} * \sqrt{p_{O2}}}{p_{H2O}} \tag{3.6}$$

The final cell voltage is

$$V_{cell} = N_{cell} * (E_T - V_{act} - V_{ohmic} - V_{conc}) \quad 3.7$$

where V_{act} , V_{ohmic} , V_{conc} represent the losses introduced in chapter 2.

The net power generated by the system should be the difference between the power of the fuel cell, calculated as the product of current and voltage, and the power dissipated by the auxiliary power units. Now, taking into account a general loss of 10% in the compressor, the system is able to produce exactly the desired output current. The power consumed by the auxiliary units, which consists mainly of the compressor, is set as the commanded P_{loss} to be provided by the batteries along with the difference between $P_{em} - P_{fc}$.

A DC boost converter that steps up the DC voltage driven by an attached controller and a gate-signal generator, is at a higher level of the model, between the fuel cell and the motor. Boost converters are also known as step-up voltage regulators because they increase voltage magnitude. Its behavior is represented by the circuit shown in Figure 3.10.

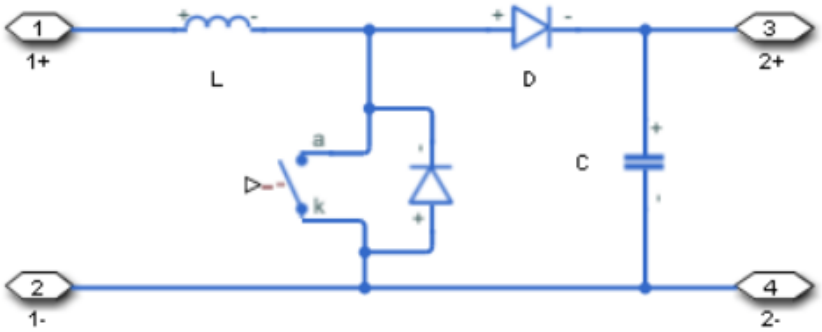

Figure 3.10 DC Boost converter equivalent circuit

3.4 Quasi-Static Model

A simpler quasi-static model is required to use the Dynamic Programming algorithm. This is due to the fact that the computation carried out by the DPM algorithm is slow, because it tests every possible combination and does not work for very small time steps, as opposed to the Simscape model. Moreover, since the FFNN will be trained offline using the data generated by the DPM, the behavior of the quasi-static model should be similar to the behavior of the Simscape model on which the controller is implemented at the end of this work. Consequently, the same parameters used in the Simscape model are implemented in the quasi-static model.

The root mean square error calculated as

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \quad 3.8$$

is used as the performance criterion to evaluate the fidelity of the quasi-static model with respect to the dynamic model. In 3.8, n is the number of observations, y_i is the value of the ith sample of the Simscape model and $\hat{y}_i$ is the estimation of y_i given by the quasi-static model. The RMSE is a good performance criterion because it penalizes high discrepancies and is also differentiable, which is very convenient for many minimization algorithms.

3.4.1 Vehicle Dynamics

Figure 3.11 illustrates the quantities involved in the vehicle longitudinal dynamics. As stated before, in quasi-static models there is no driver to translate the reference speed into pedal commands and there is no feedback of the vehicle speed. The speed trajectory is imposed on the vehicle as velocity and acceleration vectors. Torque required at wheels is calculated as

$$T_T = F_T r_w \tag{3.9}$$

where r_w is the wheel radius and F_T if the total tractive force at the contact patches of the tires given by

$$F_T = F_{aero} + F_{RR} + F_A + F_{inc} \tag{3.10}$$

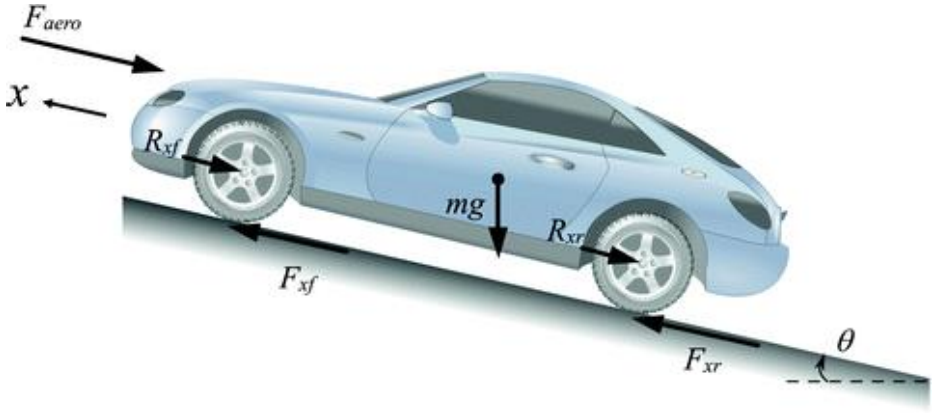

Figure 3.11 Vehicle Longitudinal Dynamics – Source [27]

The aerodynamic force is the force exerted by the air that opposes to vehicle motion and it is calculated as

$$F_{aero} = \frac{1}{2}\rho C_d A \dot{x}^2 \tag{3.11}$$

where ρ is the air density, C_d the drag coefficient, A the area of the front section of the vehicle and $\dot{x}$ the vehicle speed with respect to air. In this case $\rho = 1.18\ kg/m^3$, $C_d = 0.29$ and $A = 2.81\ m^2$.

The tire goes through repeated cycles of deformation and recovery, with the energy loss of hysteresis dissipated as heat. The rolling resistance force F_{RR} results from a resistive moment

on each tire caused by the deformation of the tire and the dynamics of the contact patch. F_{RR} is given by

$$F_{RR} = \mu_{RR} mg \tag{3.12}$$

where μ_{RR} is the rolling resistance coefficient, $g = 9.81 m/s^2$ the gravity acceleration and m the vehicle mass. Rolling resistance coefficients, which depend on many different factors but mainly on road conditions, are summarized in Table 3.2 [28].

Road Condition	**Rolling Resistance**
Bitumen, concrete	1.5
Dirt-smooth, hard, dry and well maintained	2.0
Gravel-well compacted, dry and free of loos material	2.0
Dirt-dry but not firmly packed	3.0
Gravel-dry not firmly compacted	3.0
Mud-with firm base	4.0
Gravel or sand-loose	10.0
Mud-with soft spongy base	16.0

Table 3.2 Typical rolling resistance values

The inertial force F_A is the force required to move the vehicle with a particular acceleration, and is given by

$$F_A = m\ddot{x} \tag{3.13}$$

where m is the mass m and $\ddot{x}$ the acceleration.

At last, F_{inc} is the gravitational force due to road inclination, i.e.,

$$F_{inc} = mgsin(\theta) \tag{3.14}$$

However, in this work it is assumed that the vehicle is travelling on a flat road, so the gravitational force is zero.

Another component that must be considered when attempting to increase the accuracy of the model is tire inertia, which is usually ignored in quasi-static models but is necessary in this case to match the behavior of Simscape. The tire moment of inertia J_w is multiplied by the number of wheels.

$$J_w = 4\, I_w \tag{3.15}$$

Finally, if $\dot{\omega}$ is defined as the angular acceleration of the wheel, the torque required at the wheels is

$$T_w = T_T + J_w\,\dot{\omega} \tag{3.16}$$

The slip of the wheels is not considered in this model, as it has no relevant influence on the final results and is quite complicated to calculate.

3.4.2 Electric Motor

Then, the value obtained of torque required at the wheels is converted by the gearbox to torque requested by the EM. This implies that

$$T_{EM} = T_w / fd_{ratio} \tag{3.17}$$

where T_{EM} is the torque requested by the EM and fd_{ratio} is the final drive ratio, which is 5 in the model under consideration.

Electric Motor maps are extracted from the Simscape model to define the operating region of maximum (minimum when generating) torque indexed by speed list. Using the Matlab linear interpolation function, T_{max} is obtained from the crankshaft angular speed that is imposed as the EM angular speed. Maps are then also used to compute the efficiency using interpolation of the torque and speed lists. Finally, power consumed by the EM is obtained as

$$P_{EM} = \begin{cases} T_{EM} * \omega_{EM} * \eta_{EM} & if\ T_{EM} < 0 \\ \dfrac{T_{EM} * \omega_{EM}}{\eta_{EM}} & if\ T_{EM} > 0 \end{cases} \tag{3.18}$$

When the motor act as a generator, i.e., when braking occurs, there is a maximum allowable regenerative energy which depends on the battery SOC; this defines the maximum allowable power. If negative power is greater than this maximum value, the excess power is provided by the disk brakes.

Figure 3.12 compares the values of motor torque obtained from the two models, while Figure 3.13 compares the corresponding currents, which in the quasi-static model is computed as

$$i_{EM} = \frac{P_{EM}}{v_{ref}} \tag{3.19}$$

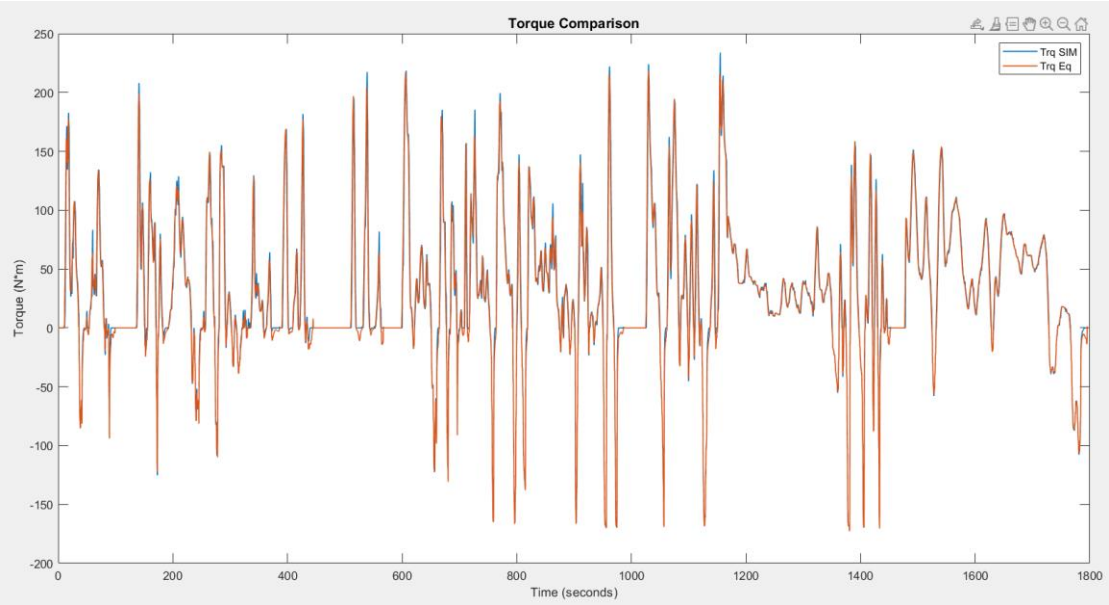

Figure 3.12 EM Torques obtained by simulation and by the quasi-static model – WLTP Class 3

$$RMSE_{TEM} = 8.9872$$

Except for some small discrepancies both curves almost perfectly overlap, showing that the quasi-static model is able to reproduce the Simscape behavior. Good results are obtained for the currents when fixing 650V as voltage reference in the static model, while in the Simscape model it exhibits very small oscillations.

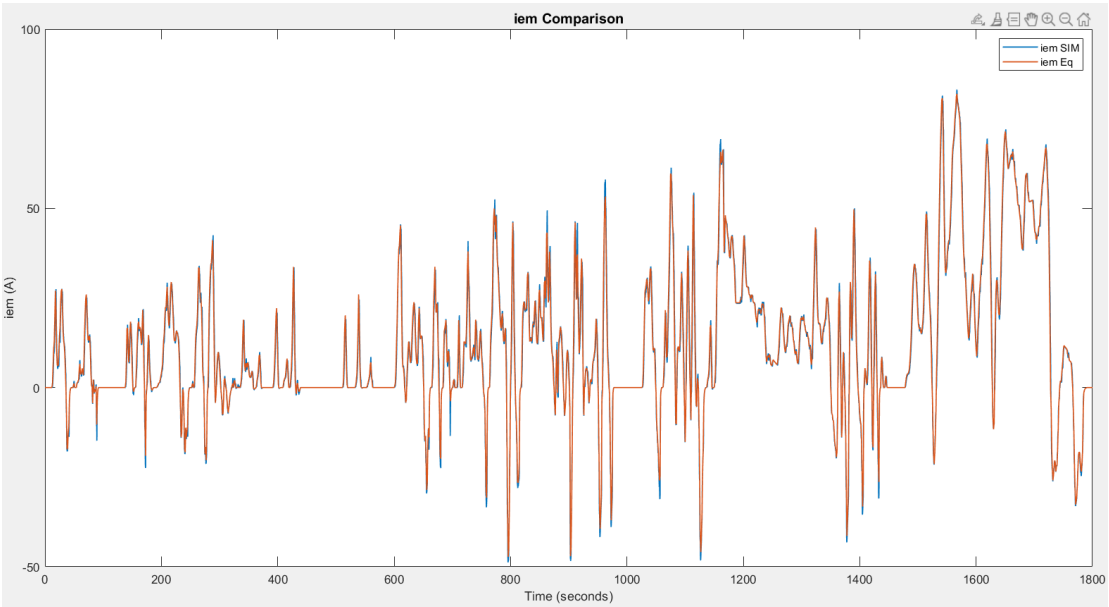

Figure 3.13 EM currents obtained by simulation and by the quasi-static model – WLTP Class 3

$$RMSE_{iEM} = 2.6468$$

3.4.3 Battery

Once the total power request is obtained, it is split between battery and fuel cell. Then, battery power is given by

$$P_{bat} = P_{EM} - P_{FC} + P_{loss} \tag{3.20}$$

which considers the power losses P_{loss} of the compressor, as seen in the Simscape model.

The controller will be in charge of determining the power split, as presented in chapter 4. Quasi-static maps are extracted from the Simscape model, and the DC/DC converter is not considered as an external component but lumped inside the battery. The battery internal resistance R_0 is obtained through interpolation depending on the SOC. Efficiency is assumed one when discharging and 0.98 when charging. Then, battery voltage v is computed using interpolation of the V_{noload} and SOC maps. Resulting current is calculated as

$$i_b = \eta_{batt} * \frac{v - \sqrt{v^2 - 4R_0P_{bat}}}{2R_0} \tag{3.21}$$

At each time step, the SOC is updated using its previous value and the current passing through the battery, according to equation

$$SOC_{t+1} = \frac{-i_b}{Q_{bat} * 3600} + SOC_t \tag{3.22}$$

The effect of temperature is not considered in the quasi-static model and a room temperature of 25 °C is assumed. To test the effectiveness of this model, same P_{bat} is applied to both Simscape and quasi-static models, and the resulting current is shown in Figure 3.14.

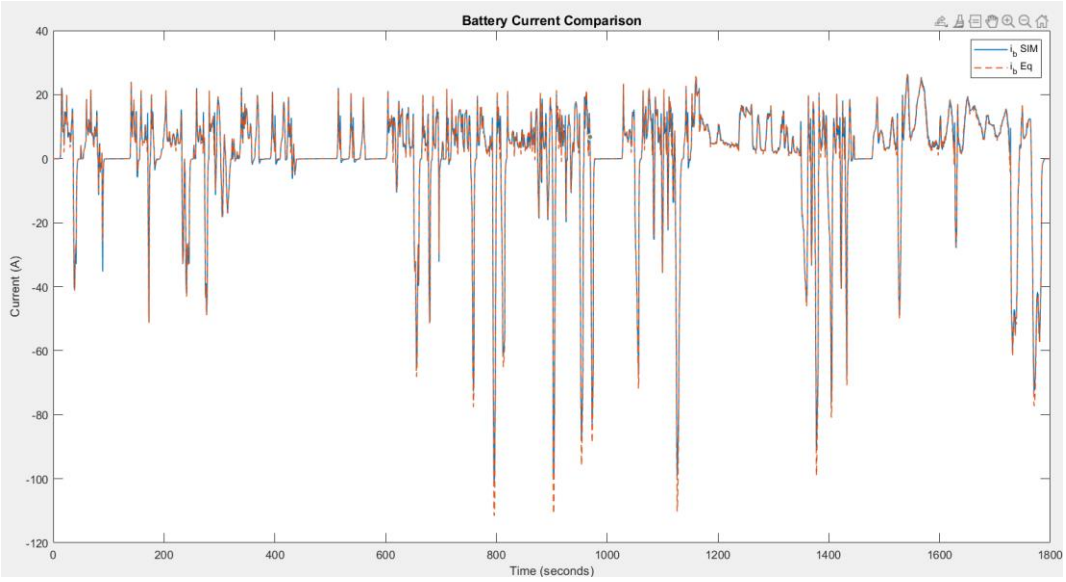

Figure 3.14 Battery currents obtained by simulation and by the quasi-static model – WLTP Class 3

$$RMSE_{ib} = 1.4592$$

In contrast, the voltage exhibits a filtered behavior, and thus a higher discrepancy

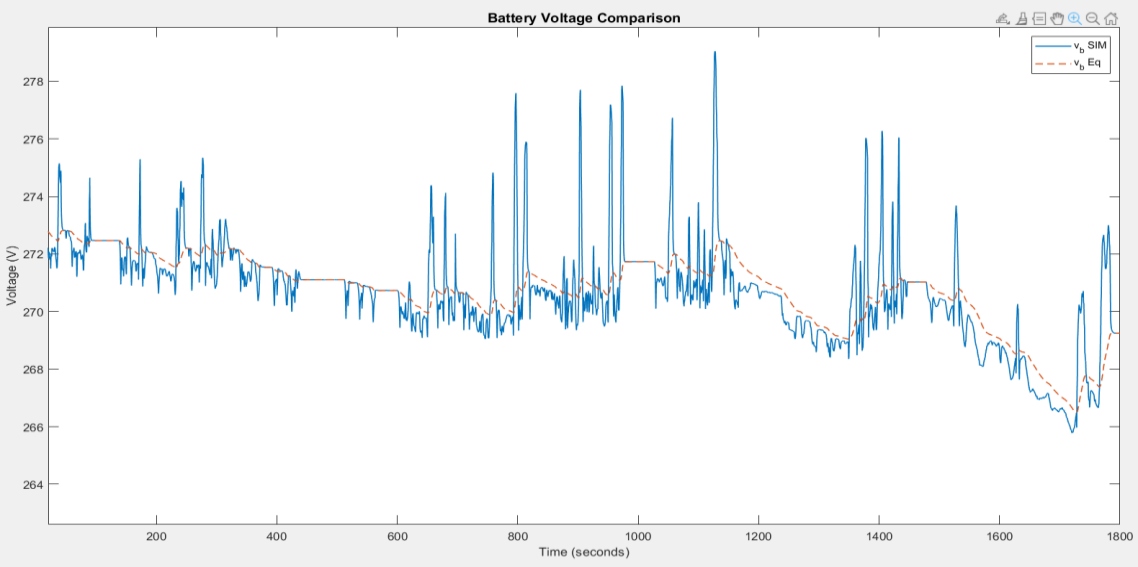

Figure 3.15 Battery voltages obtained by simulation and by the quasi-static model – WLTP Class 3

$$RMSE_{vb} = 4.6954$$

3.4.4 Fuel Cell

The last component of the powertrain that needs to be modeled is the fuel cell. Since there are no maps for this element in the Simscape model, they were created using a data-driven approach. First, simulations are performed with the WLTP3 and FTP75 driving cycles, to obtain data on fuel cell performance, power losses due to auxiliaries and hydrogen consumption.

Then, the Matlab curve fitting toolbox is used to find a correlation that describes this data. A quadratic polynomial is used for fuel consumption, which is measured as kg of hydrogen used per second.

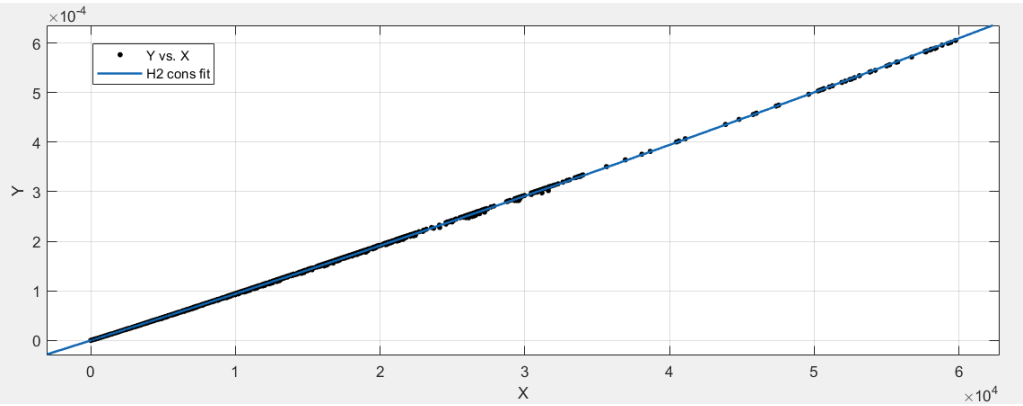

Figure 3.16 Curve fitting H2 consumption

On the other hand, a 3^{rd} order polynomial is fitted to the power losses due to auxiliaries, showing the expected exponential behavior. Indeed, Fuel Cells are significantly more efficient at low loads, as seen in the second chapter.

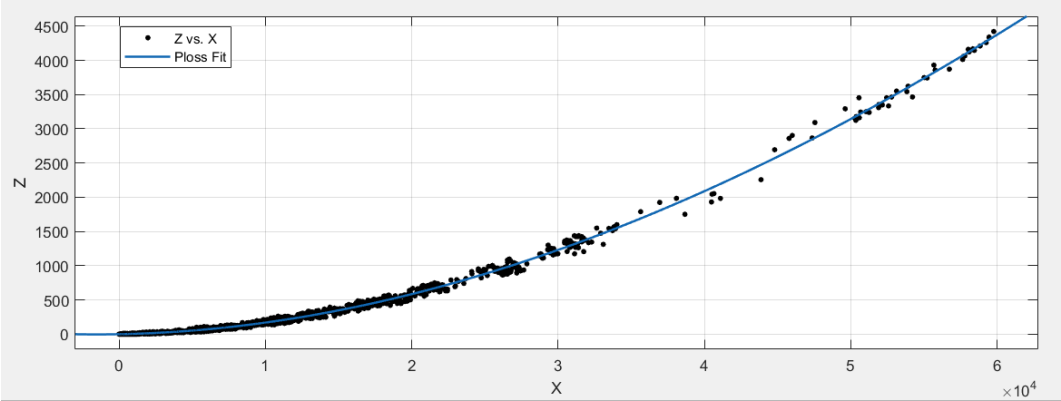

Figure 3.17 Curve fitting Ploss

In Figure 3.16 X is the power of the fuel cell which ranges from 0 kW to 60 kW, and is calculated as the product of voltage and current. On the other hand, Z is the power dissipation due to auxiliaries, with the compressor accounting for the largest share.

Once both relations are determined a vector of P_{FC} values is created, and the corresponding values of P_{loss} and H_2 consumption are used to build the maps. Likewise for the EM and for the Battery, quasi-static maps are tested considering the same sequence of P_{FC} both in the Simscape and quasi-static models for the WLTP Class 3 cycle. Figures 3.17 and 3.18 respectively show the values of P_{loss} and H_2 consumption, obtained with both the Simscape and the quasi-static models.

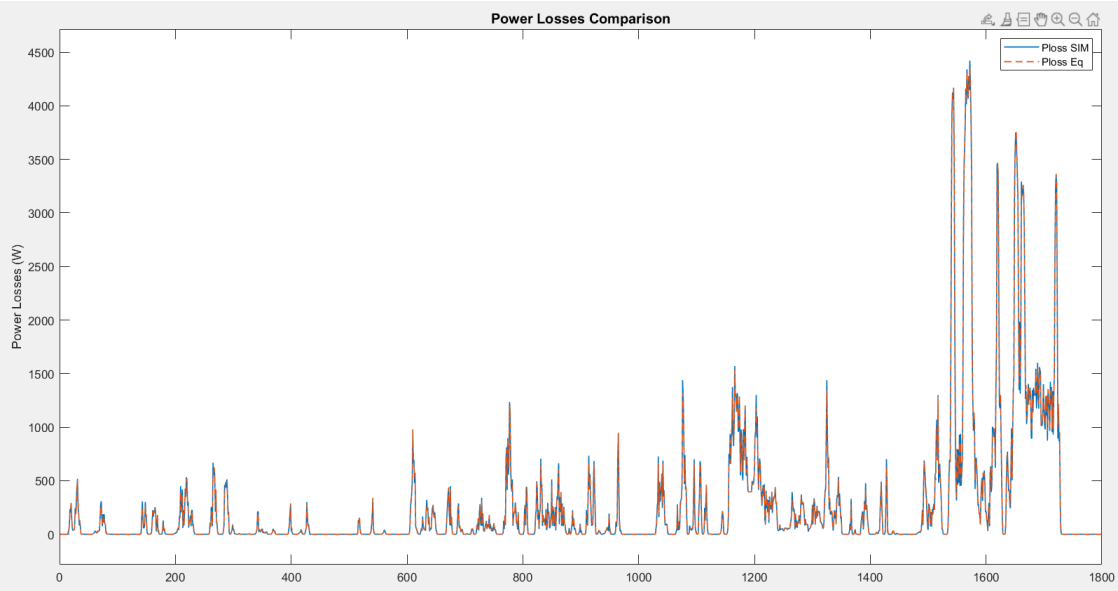

Figure 3.18 Ploss obtained by simulation and by the quasi-static model – WLTP Class 3

$$RMSE_{Ploss} = 28.8938$$

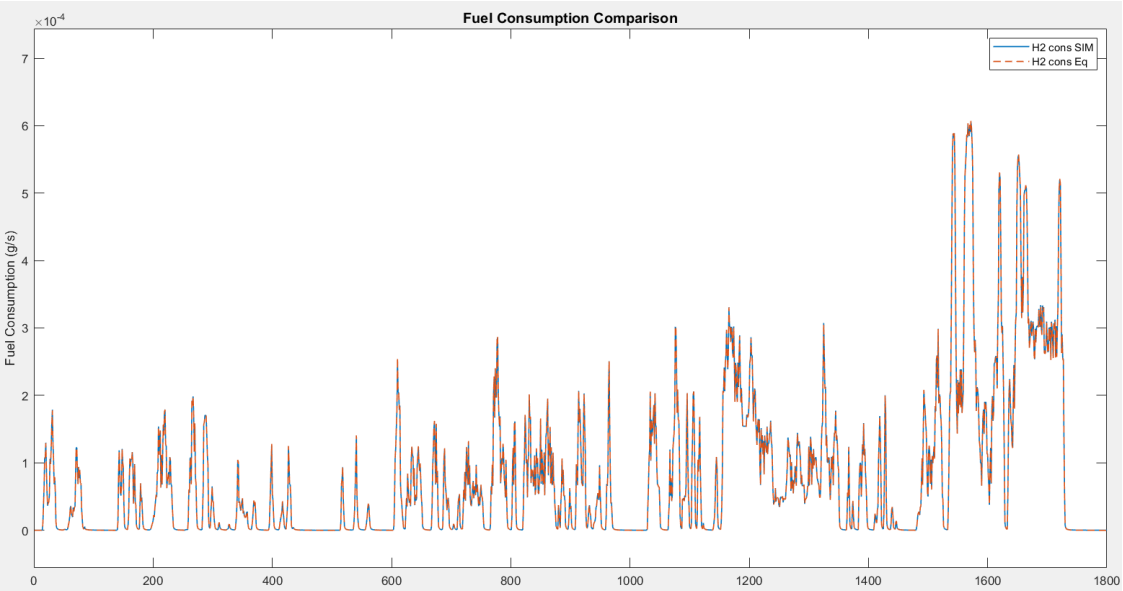

Figure 3.19 Fuel consumption obtained by simulation and by the quasi-static model – WLTP Class 3

$$RMSE_{mfuel} = 5.7320 * 10^{-7}$$

Models obtained in this chapter are implemented in Matlab as the `fun`, the handle of the DPM to the model function.

4 Energy Management Strategies

Hybrid Electric Vehicles (HEVs) combine two power sources to meet the load demand. The Energy Management Strategy (EMS) is responsible for determining how the power should be split between both sources at every time instant during the operation of the HEV. Consequently, the EMS should be designed to guarantee normal operation of the vehicle, and also to provide better performance of the various power sources [29].

At present, many different types of EMS are subject of study, and the general direction is moving from single-objective towards multi-objective optimization problems. Indeed, the high cost of fuel cells at their initial stages is still one of the major obstacles in the competition of FCHEVs with conventional ICEs and with HEVs. In this context, at first instance the EMS plays a key role in reducing fuel consumption, but also in prolonging FC lifetime, thus reducing the total cost.

In some papers, EMS are also referred to as Power Management Strategies PMS; these terms will be used as synonyms in this work.

4.1 Review of EMS

FCHEVs are relatively new and there is no extensive literature about EMS dedicated to them. However, since they are very similar to HEVs in terms of their operation, the review is conducted considering EMS applied to HEVs.

A first classification is to categorize EMS as online and offline strategies. The former refers to the possibility of real-time implementation, while the latter require a-priori information about the driving cycle and other parameters, and thus cannot be implemented in real-time. Moreover, offline optimization methods are usually slow and have a high computational cost, which is another obstacle to their use in real vehicles. Their use is limited to analysis purposes or as benchmarks to check the performance of online strategies.

Thus, offline strategies are mainly global optimization algorithms or stochastic search methods such as genetic algorithms, dynamic programming, particle swarm optimization, among others.

On the other hand, online EMS comprise various other methods besides those that are based on optimization strategies. Indeed, when dealing with real-time implementation, the main purpose is to get a vehicle up and running as quickly as possible, and thus simpler implementations generally based on coded heuristic rules that can have various level of complexity are preferred. Figure 4.1 shows a that online EMS can be classified in rule-based and optimization-based strategies.

Rule-based approaches are based on human expertise without prior knowledge of the driving condition. They have some advantages such as low computational load, easy real-time implementation and robustness. However, they fail in finding the optimal solution and have to

be re-tuned if changes are made. Rule-based EMS are in turn divided in deterministic and fuzzy logic.

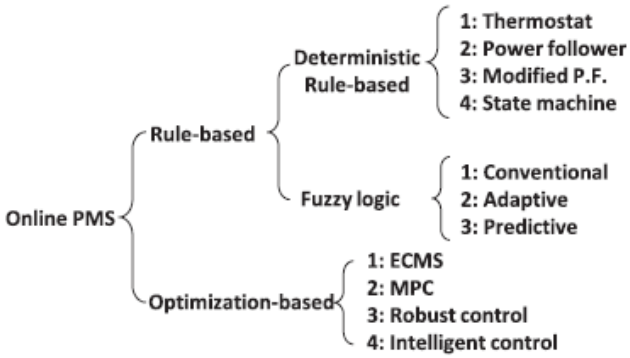

Figure 4.1 Online EMS Classification – Source [2]

As opposed to offline optimization strategies, online optimization-based strategies are able to find only a suboptimal solution due to their limited knowledge of the driving conditions. The main philosophy of this type of implementation is to divide the whole cycle into sub-cycles, and find the optimal solution along this limited horizon. They have the advantage of both the rule-based approach, i.e., real-time implementation due to their fast computation and robustness features, and offline optimization. The challenge is to keep the suboptimal solutions as close as possible to the global optimal solutions. Table 4.1 [29] presents the advantages and disadvantages of the main EMSs:

EMS	Optimization	Advantages	Disadvantages
Dynamic Programming DP	Energy consumption & FC lifetime	High accuracy of fuel consumption calculation, Used as benchmark by other strategies	Difficult for real –time application because of the large computational load
Pontryagin's minimal principle	Energy consumption & FC lifetime	Computational load is relatively low and result is close to global optimization	Calculation accuracy is greatly affected by the initial value of the costate variable.
Extremum Seeking	Energy consumption	Calculation results are close to the global optimization and suitable for real vehicle operation	Difficult to simultaneously optimize fuel consumption and fuel cell lifetime
ECMS	Energy consumption & FC lifetime	Relatively high accuracy of fuel consumption calculation and high performance in real-time	It involves functional analysis, which is difficult to understand.
Neural Network	Energy consumption & FC lifetime	High quality in real-time performance, near optimal performance	Large amounts of data are required to train the NN.

Table 4.1 Review of the main EMSs

Neural networks EMSs are the most promising because they are able to achieve near-optimal performance, including multi-objective optimization, with a much lower computational cost than traditional algorithms. This work proposes a neural network EMS that attempts to approximate the performance of the Dynamic Programming algorithm using limited information about previous time instants. DP was chosen as a reference because it is capable of finding a global optimal solution.

4.2 Performance evaluation criteria

Performance criteria provide a systematic and quantitative approach to compare controllers. These criteria form the basis for the cost function and constraints used in the power split optimization routine that is carried out using the DPM algorithm.

When analyzing alternative powertrain solutions, it is necessary to define standardized units to compare results in the case of alternative fuels, which have different energy densities and different units that make direct comparison not viable. For this purpose, the Gasoline Gallon Equivalent (GGE) was defined and accepted worldwide; it represents the amount of alternative fuel that is necessary to equalize the energy of one liquid gallon of gasoline. US liquid gallon is a unit of volume in imperial units

$$1\ US\ gal = 3.785411784\ L$$

and although it is not included in the International System of Units (SI), it is widely used in the automotive sector. Table 4.2, available from the U.S. Department of Energy website, contains conversion factors that state fleets and alternative fuel providers can use to report their compliance with Energy Policy Act EPAct requirements.

Fuel Type	Fuel Measurement Unit	Conversion Factor	GGE Calculation
B100	Gallons	1.066	$GGE = B100\ gal * 1.066$
CNG	Hundred cubic feet	0.877	$GGE = CNG\ ccf * 0.877$
Diesel	Gallons	1.155	$GGE = Diesel\ gal * 1.155$
E85	Gallons	0.734	$GGE = E85\ gal * 0.734$
Electricity	kWh	0.031	$GGE = Electricity\ kWh * 0.0031$
Gasoline	Gallons	/	/
Hydrogen	Kg	1.019	$GGE = H_2\ kg * 1.019$
LNG	Gallons	0.666	$GGE = LNG\ gal * 0.666$
LPG	Gallons	0.758	$GGE = LPG\ gal * 0.758$

Table 4.2 Gasoline Gallon Equivalent Factors - Source epact site

Since the case under study corresponds to a FCHEV, there are two different sources, and thus fuel consumption is considered as the sum of the two sources expressed in GGE. To be competitive with respect to a conventional vehicle, the objective is to maximize the Miles per GGE that measure the average distance traveled per unit of energy consumed, or equivalently to minimize liters/100km.

4.3 DPM ETH in Matlab

As already introduced in chapter 2, the DPM function uses a model of the vehicle and computes every possible trajectory to find the optimal one.

Inputs used in the `prb` structure are the vectors representing the drive cycle, speed and acceleration vectors under consideration. A major disadvantage of the DPM algorithm is its heavy computational load, which makes unfeasible to use the continuous time vectors extracted from the Simscape model. Thus, vectors are discretized in time steps of 1 second.

The multi-objective aims of the controller, i.e.

- Fuel consumption minimization
- Charge-Sustaining
- Improve fuel cell and battery lifetime

should be considered to build the remaining structures.

Fuel consumption minimization is achieved by setting the cost function as the sum of the GGEs of hydrogen and battery power, i.e.,

```
GE_tot = m_fuel_ge + Batt_ge;

C{1} = (GE_tot);
```

At each time step k, the following state within the feasible region is computed following the sets of controls and costs at each stage. The Costs-to-go are the costs associated to the movement from one step to another. Once all costs are computed, DPM finds the trajectory that minimizes the cost function over the entire path. This is illustrated in Figure 4.2.

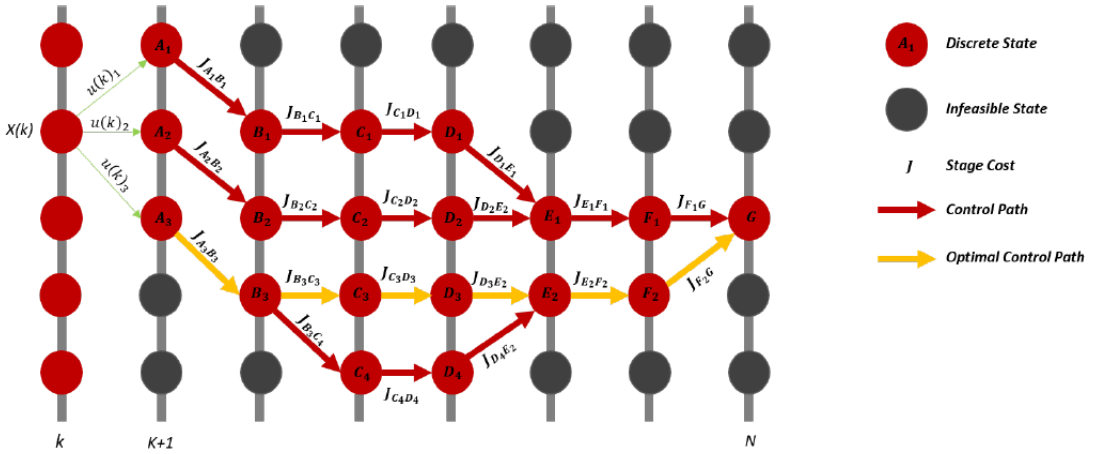

Figure 4.2, Illustration of the DPM algorithm for the problem of interest

Charge sustaining mode implies that the net energy provided by the battery during the entire cycle should be zero, i.e., the initial SOC should be equal to the final SOC. In this way, EMS ensures that the battery does not have to be charged from outside, thus exploiting the first advantage of fuel cell vehicles over BEVs: faster refueling time of hydrogen with respect to battery charging.

The last objective, to improve both fuel cell and battery lifecycle, is necessary to reduce the WTW emissions, as seen in chapter 2. Common practices to reduce FC aging are the reduction of transient loading, and prevention of reactant starvation [30]. Battery life is dependent on a wide variety of parameters including current, operating temperature and the depth of discharge (DOD) [31].

These objectives, namely life cycle of components and charge sustaining mode, are achieved by defining two different states to take into account FC and battery degradation.

Admissible values of State of charge are set in the range from 0.65 to 0.55, with a discretization step of 0.001. The selected range ensures small depth of discharge and also constitutes a high efficiency region; indeed, moving away from these values the battery performance decreases. These values set the limits of the feasible region for the first state

```
grd.Nx{1}    = 101;
grd.Xn{1}.hi = 0.65;
grd.Xn{1}.lo = 0.55;
```

To ensure charge sustaining mode, a penalty function is used to enforce the constraint that the final state is equal to the initial one

```
%set initial state
grd.X0{1} = 0.60;
%final state constraints
grd.XN{1}.hi = 0.61;
grd.XN{1}.lo = 0.60;
```

The slightly higher value of 0.61 is set to take advantage of the complete regenerative braking, because for the SOC lower limit of 0.55 high aggressive cycles, such as WLTP, were not giving feasible solutions.

Regarding the fuel cell, smooth transitions in power output are ensured by checking the previous state and setting a limit on the variations.

$$\Delta P_{FC} < 5\ kW/s$$

When the variation between two consecutive time steps is higher than this limit, the cost-to-go is set to a very high value in the `options` structure by choosing `MyInf` value.

```
grd.Nx{2}    = 61;
grd.Xn{2}.hi = 60000;
grd.Xn{2}.lo = 0;
```

Another limitation is set on the maximum value for the output power. The fuel cell has a maximum power of 128 kW but, as already seen in chapter 2, its efficiency decreases significantly at high loads. Then a power of 60kW is chosen to ensure that the fuel cell operates in a high efficiency range. The discretization step for P_{FC} is set at 1kW.

Finally, the control input is defined as the ratio

$$u = \frac{P_{FC}}{P_{EM}}$$

This ratio defines how much power is provided by the fuel cell; the remaining part is drawn from the battery to meet the load demand. Alternatively, the fuel cell may provide more power than the requested by the motor, to charge the battery if needed.

```
grd.Nu{1}    = 25;
grd.Un{1}.hi = 1.2;
grd.Un{1}.lo = 0;
```

The range for the control input is chosen from 0 to 1.2, with a discretization step of 0.05. DPM tries every possible control input and compute the cost for each one. Physical limits of the components are taken into account by setting a `MyInf` value as cost.

```
% Summarize infeasible EM
inm = (isnan(e)) + (Tem<0)   .* (Tem < MotTrqMin) +...
                    (Tem>=0) .* (Tem > MotTrqMax);


% Summarize infeasible FC
Pfc_max = 128*1000;
infc = (Pfc > Pfc_max);


% Summarize infeasible Battery
inb = (v.^2 < 4.*r.*Pbat) + (abs(Pbat)>Plim);
```

At the end, the dpm function return the `res` structure containing all the outputs selected and the optimal control policy, which is then applied through a forward simulation starting from the given initial state.

4.4 Neural Network modeling of the behavior of the DPM algorithm

Once the optimal policy is determined and the forward simulation results are satisfactory, the data obtained are used to train a Feedforward Neural Network to reproduce, as accurate as possible, the behavior of the DPM algorithm. The power split decision of the DPM algorithm is based on three elements:

- Total power required by the motor
- State of Charge of the battery
- Previous values of fuel cell power

The total power required by the motor may be computed from the reference speed and reference acceleration vectors; hence, it was decided to use this information as inputs to the Feedforward Neural Network. The SOC is also considered, because the SOC bounds set in the previous paragraph should be met. At last, previous values of the fuel cell power are also included as inputs to the FFNN, with the purpose of preventing sudden changes. On the other hand, the power split produced by the DPM, expressed as the fuel cell power P_{FC}, is the output of the

neural network; hence, this value from the data is set as the target during the training phase. In summary, the FFNN has 11 inputs, corresponding to the actual and two previous values of the reference speed, reference acceleration and SOC and two previous values of P_{FC}, and one output corresponding to the current value of P_{FC}; in addition, the FFNN has two hidden layers of 10 units each. Figure 4.3 illustrates the architecture of the FFNN trained to reproduce the behavior of the DPM algorithm.

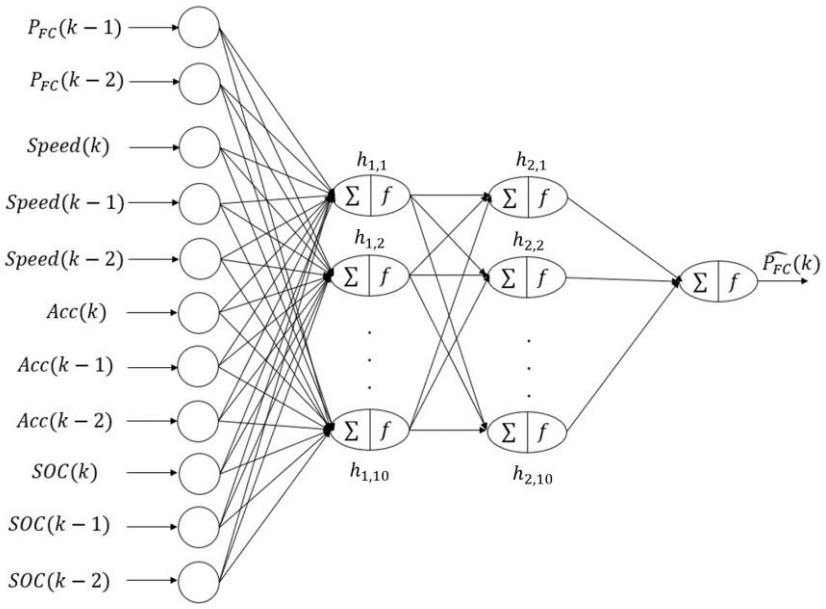

Figure 4.3 architecture of the FFNN trained to reproduce the behavior of the DPM algorithm

Another important step is the selection of the sample time. A time step of 1 s is used in the DPM function to reduce the computational load; however, since the Simscape model is a dynamic model, it employs a significantly finer discretization to simulate the quasi-continuous behavior of Simscape. Consequently, all vectors obtained from the DPM function were resampled with a time step of 0.2 seconds. Table 4.3 summarizes the features of the dataset used to train this FFNN.

Inputs	Output/Target	Sample Time
Speed Acceleration SOC $P_{FC}(t-1)$	$P_{FC}(t)$	$T_S = 0.2\ s$

Table 4.3 Features of the dataset used to train the FFNN

The results given by the DPM algorithm for both the WLTP3 and FTP75 driving cycles, are used to build the dataset for training the FFNN according to its architecture. Before using the data to train the FFNN, the values of each variable were normalized to span the interval $[-1,1]$. Then, to ensure that the training, validation and test sets have data from both driving cycles and to prevent the FFNN from overfitting the training data, the data vectors obtained from both driving cycles were merged and further shuffled. All these tasks were implemented using the instructions shown below.

```
X1 = [SpeedX1; AccX1; SOCX1; PfcX1];

X2 = [SpeedX2; AccX2; SOCX2; PfcX2];

X=[X1 X2];

Y1 = [-1+2*(Pfc1-Pfc_min)/(Pfc_max-Pfc_min)];

Y2 = [-1+2*(Pfc2-Pfc_min)/(Pfc_max-Pfc_min)];

Y=[Y1 Y2];


%% Shuffle

N=length(X);

Ish=randperm(N);

Xsh=[];

Ysh=[];


for i=1:N

    Xsh=[Xsh X(:,Ish(i))];

    Ysh=[Ysh Y(:,Ish(i))];

end
```

Then, the data was split into train, validation and test sets, with a 70%-15%.15% proportion, respectively. Finally, the FFNN is trained using the Levenberg-Marquardt algorithm. The dialog box of the NN Matlab Toolbox at the end of the training process is shown in Figure 4.4.

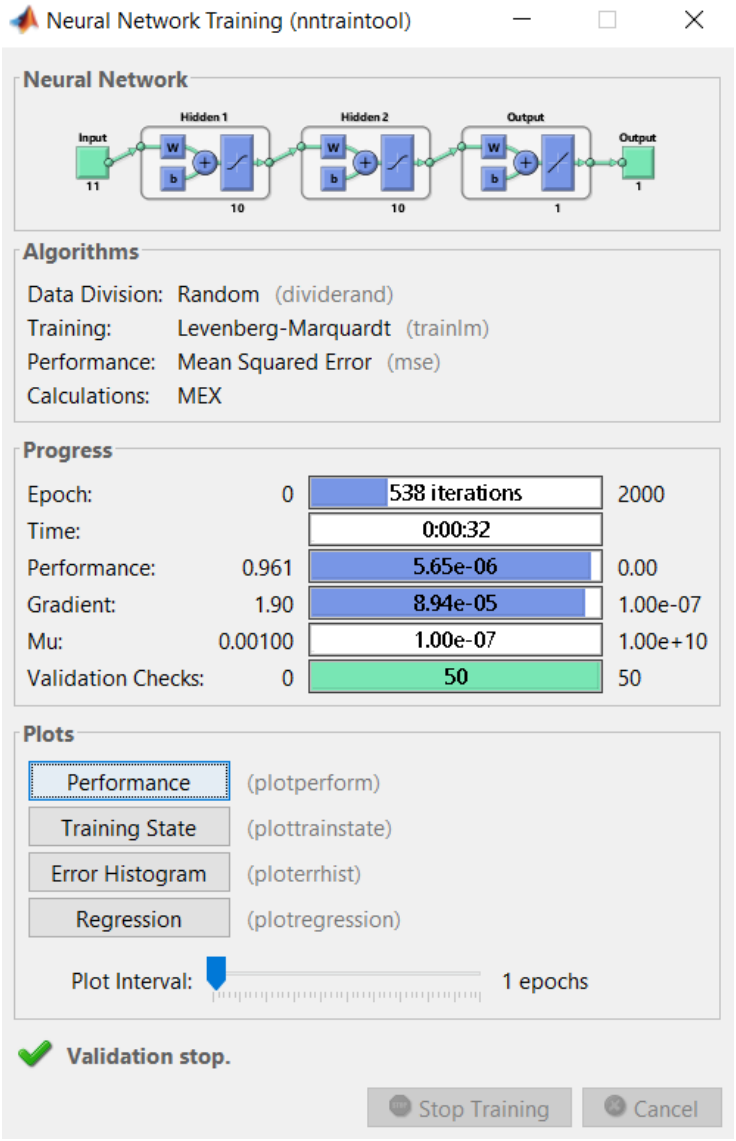

Figure 4.4 Neural Network toolbox dialog box at the end of the training process

The training stops after 538 iterations due to the validation criterion, and shows good performance results. Then, the output vector produced by the FFNN is denormalized. and compared with target values in the WLTP3 cycle. Figure 4.5 plots the values estimated by the FFNN vs the real values given by the DPM algorithm. The RMSE is computed considering the error between estimated value and actual DPM output, resulting in

$$RMSE_{NN} = 68.4115$$

Results shows that the FFNN achieves a good approximation accuracy of the DPM behavior, and thus it is proceeded to implement the controller.

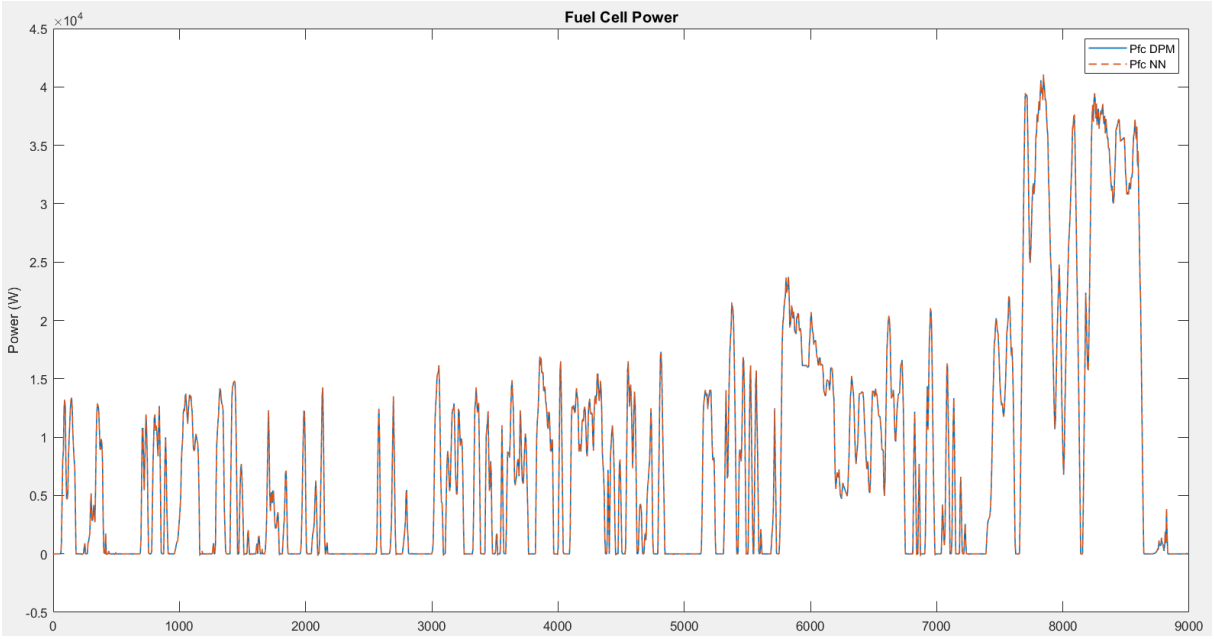

Figure 4.5 Neural Network estimation vs real DPM output for the WLTP3 cycle

4.5 Controller implementation

The implementation of the NN controller in the Simulink environment is carried out using the `gensim` Matlab function. This command takes as input the `net` object obtained at the end of the training process and the sampling time, 0.2 seconds in this case. In other words, the command is executed as

```
gensim (net,st)
```

and creates the Simulink system shown in Figure 4.6.

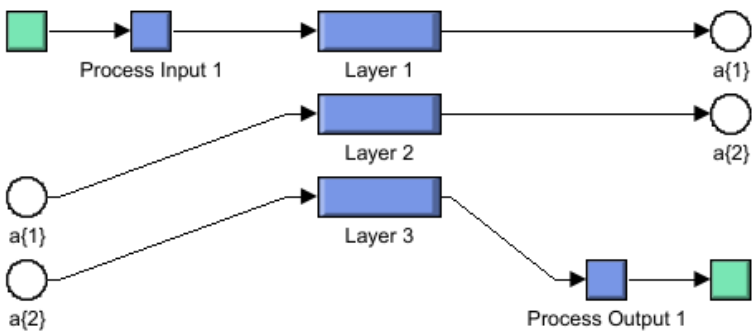

Figure 4.6 System created by the `gensim` command

Then, the block generated is implemented as the controller in the Simscape model in place of the original PID controller. The diagram of the controller implemented is shown in Figure 4.7.

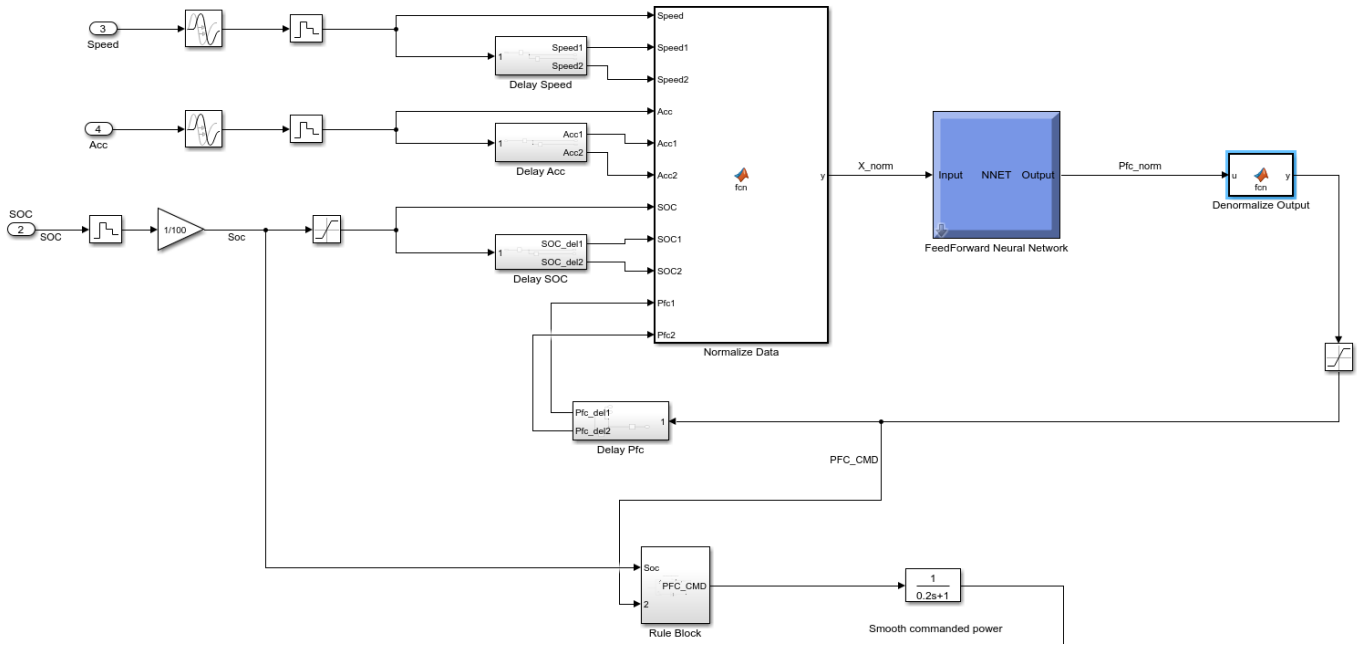

Figure 4.7 Diagrama of the controller implementation

Inputs are discretized using a zero-order-hold block with a sampling time of 0.2 seconds. Then, delay blocks are used to configure the vector of measured values according to the 11-input structure of the FFNN.

Normalization and denormalization are carried out using two Matlab functions block, to obtain the commanded fuel cell power as output. To handle exceptions, a rule block is built that checks if the SOC is within the desired range 0.55-0.65, and modifies the commanded power if needed. This block is necessary because the FFNN is trained only with SOC values inside the range, because DPM results only obtain values in this range, but due to some differences between the quasi-static and the Simscape model, SOC can divert from the range causing the network to perform unproperly. Specifically, the following rules are used

- $If\ 0.55 < SOC < 0.65 \rightarrow P_{FC} = P_{FCNN}$
- $If\ SOC > 0.65 \rightarrow P_{FC} = 0$
- $If\ SOC < 0.55 \rightarrow P_{FC} = P_{FCNN} + P_{FCmin}$

The last rule adds a minimum power of 1 kW to the commanded output of the neural network.

Finally, the rule block output is smoothed using a filter with a time constant equal to the sampling time of the network.

5 Results

5.1 DPM results

5.1.1 WLTP Class 3 Results

The first cycle tested is the WLTP Class 3, for which Figure 5.1 shows speed and acceleration and Figure 5.2 shows the output power of the DPM algorithm and the resulting SOC trajectory.

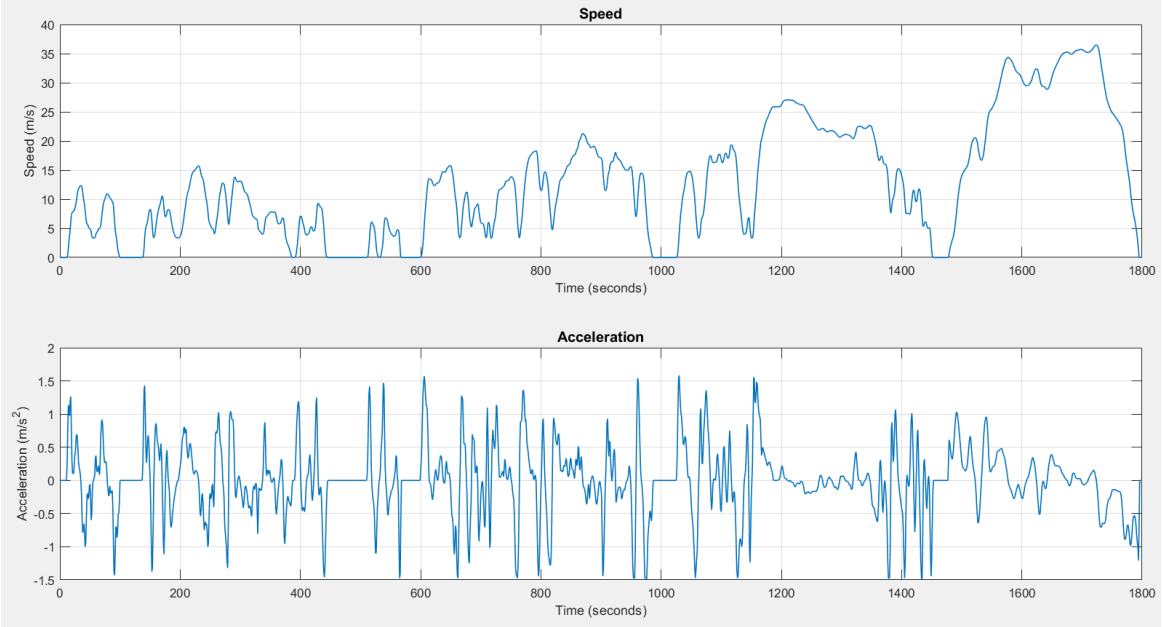

Figure 5.1 Speed and Acceleration for the WLTP3 cycle

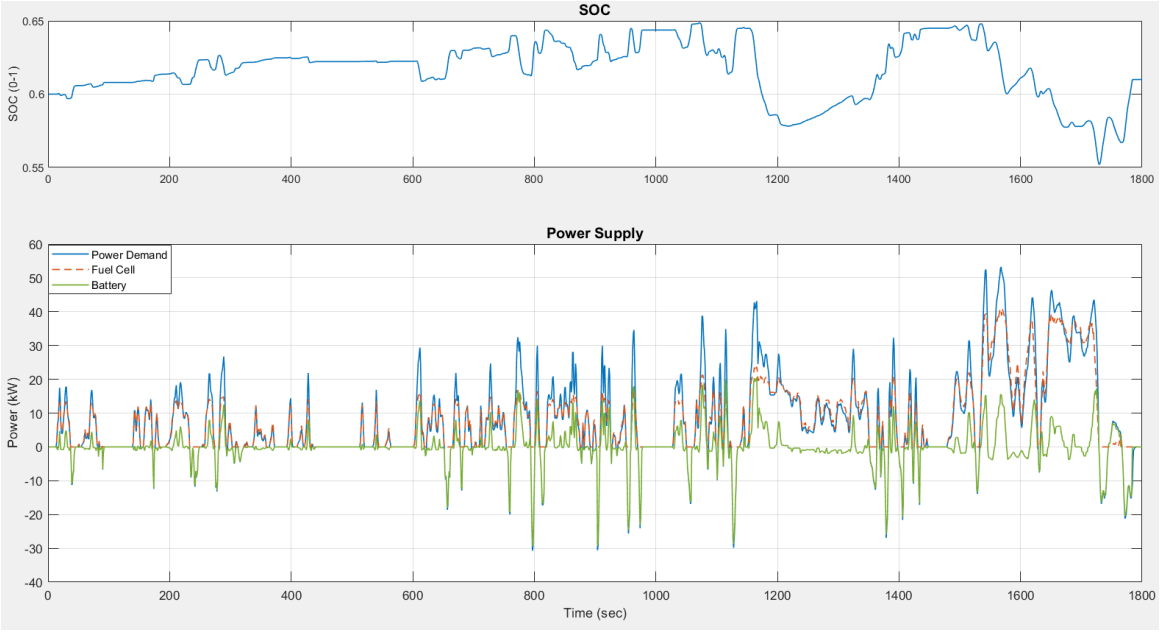

Figure 5.2 DPM Output for the WLTP3: SOC and power supply

From the behavior of the SOC, it is seen that the DPM algorithm meets the power demand for low load requests with the fuel cell and maintains the SOC close to its initial value, charging the battery when brake events occur. Since the fuel cell experiences a performance loss for higher power outputs, particularly when the aggressive region of the cycle is reached at time instant 1150, the battery helps boosting the fuel cell to avoid sudden increases and does not let the fuel cell to work in the low efficiency region. The DPM algorithm takes advantage of the regenerative energy coming from the motor, which could not be possible without setting to 0.61 the constraint for the final state of SOC.

Figure 5.3 shows the total fuel consumption resulting from applying the DPM control policy, for the WLTP3 driving cycle. Note that the instantaneous fuel consumption is measured in g/s, while the cumulative fuel consumption is given in kg. At the end of the cycle, the vehicle has consumed 0.137304 kg of hydrogen. The result obtained in liters/100 km are in correspondence with the requirements. Indeed, according to the technical roadmap of energy-saving and new energy vehicles, the average fuel consumption of new ICE passenger cars in 2020 and 2025 will reach 5 liters/100 km and 4 liters/100 km, respectively [32]. It is seen that the FCHEV outperforms conventional vehicles with respect to Gasoline Gallon Equivalent.

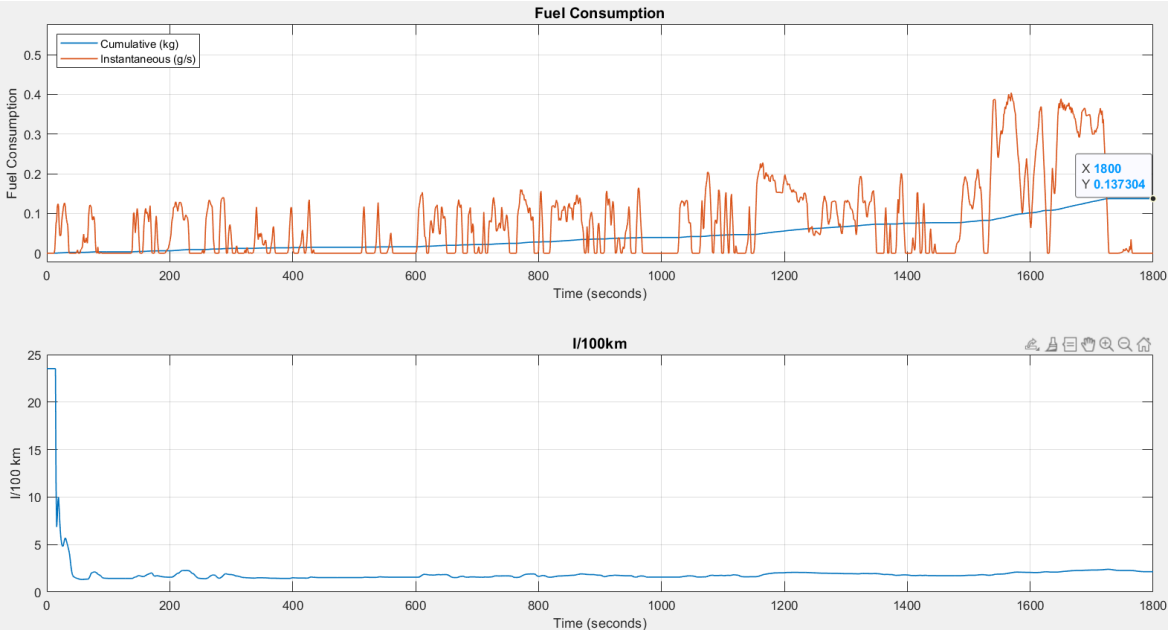

Figure 5.3 Fuel Consumption obtained in the DPM run for the WLTP3 cycle

5.1.2 FTP75 Results

The same analysis was carried out for the FTP75 driving cycle. Figure 5.4 shows the cycle speed and acceleration profile, while Figure 5.5 shows the resulting output of the DPM algorithm.

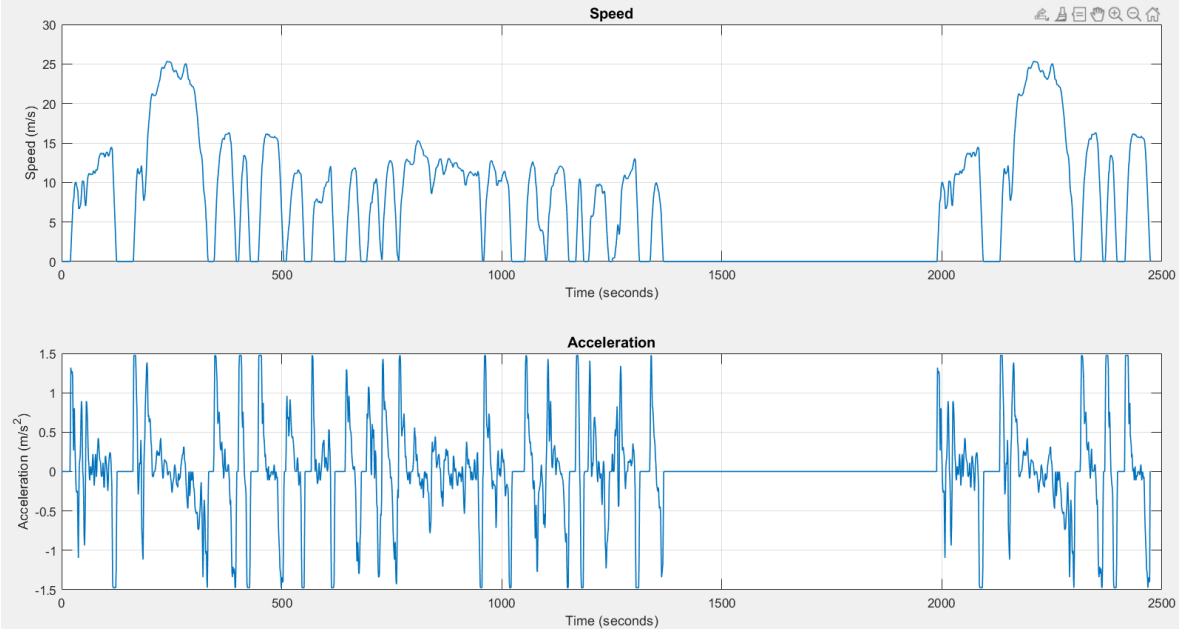

Figure 5.4 Speed and Acceleration of the FTP75 cycle

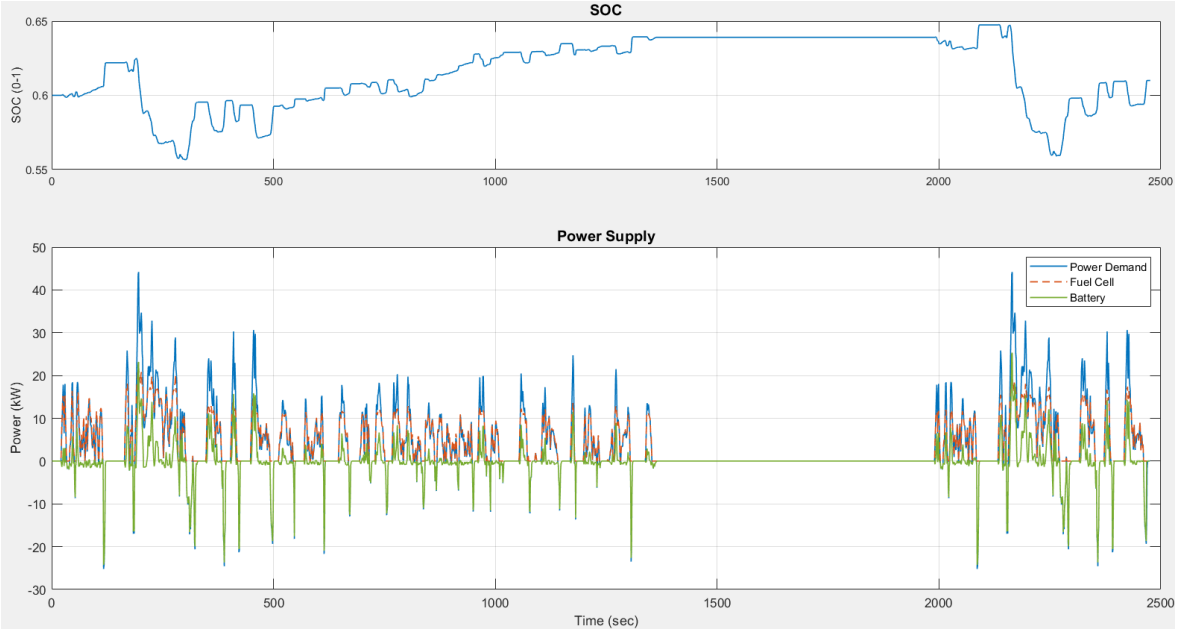

Figure 5.5 Policy obtained in the DPM run for the FTP75 cycle

Also, for the simulation of the FTP75 cycle, DPM is able to fulfill the requirements, with a final value of SOC of 0.609. Battery intervenes for high load requests, to prevent the fuel cell from operating in the low efficiency region.

The resulting fuel consumption is shown in Figure 5.6.

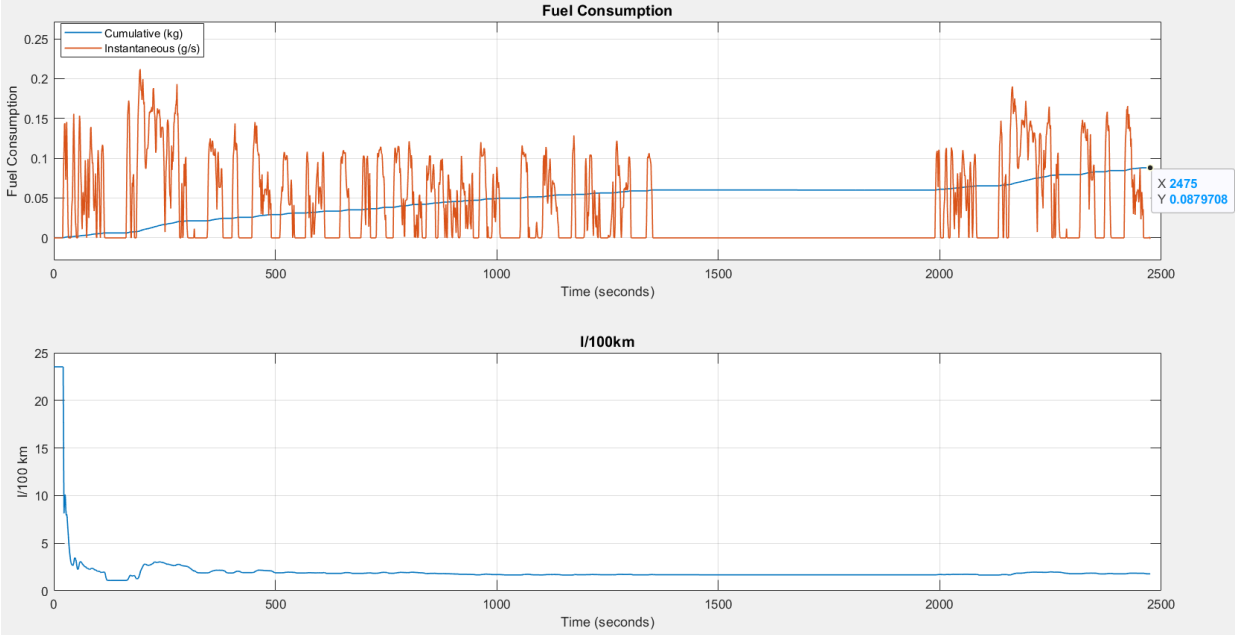

Figure 5.6 Fuel Consumption obtained in the DPM run for the FTP75 cycle

The resulting fuel consumption for this cycle is lower than for the WLTP3 cycle. This result was expected, because the latter is a more aggressive cycle, which is clear by looking at the average power request of both cycles: WLTP3 has an average $mean(P_{tot}) = 7.65\ kW$, while the FTP75 has a $mean(P_{tot}) = 3.66\ kW$.

Even in this cycle, the achieved value of liters/100km is good, which is significantly below the standards imposed to conventional vehicles.

5.2 Comparison between models

Chapter 3 shows that the performance of the quasi-static model is satisfactory compared to the Simscape model. However, in that section the sample time was 0.2 seconds and the components are considered separately. In contrast, the sampling time used to run the DPM is 1 second, to reduce the computational load typical of that algorithm. For this reason, a slightly different behavior is expected when using the Simscape model, and the following test is carried out to quantify these differences.

First, the DPM policy computed offline is applied directly on the Simscape model for the WLTP3 cycle. Then, the difference between the two SOC profiles, the obtained in the DPM offline run and the one resulting from the direct application of the DPM policy on the Simscape model. The purpose of performing this test only for the WLTP3 cycle is to demonstrate that even if the Neural Network controller is able to approximate the DPM with reasonable accuracy, it is not able to satisfactorily reproduce the offline results when implemented in real time; however, the discrepancies obtained are acceptable.

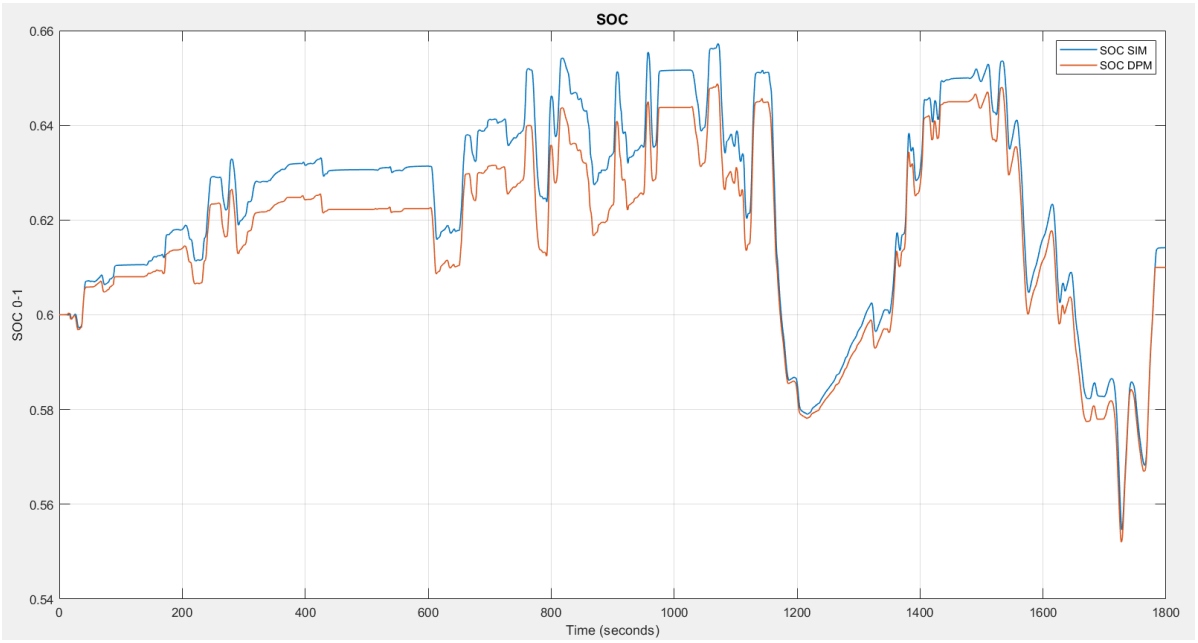

Figure 5.7 Comparison of SOC in Simscape and in the DPM applying same policy

It can be seen in Figure 5.7 that both SOCs follow similar trajectories, but they are slightly different. Indeed, the resulting $RMSE = 0.0068$ represents approximately 1% of SOC maximum value. On the other hand, Figure 5.8 shows the curves corresponding to fuel consumption for both cases; note that the two curves almost perfectly overlap, which is also verified by the negligible difference between the values of hydrogen consumed at the end of the cycle, which were $H2_{SIM} = 0.1375\ kg$ for the Simscape model and $H2_{DPM} = 0.1373\ kg$ obtained in the DPM offline run.

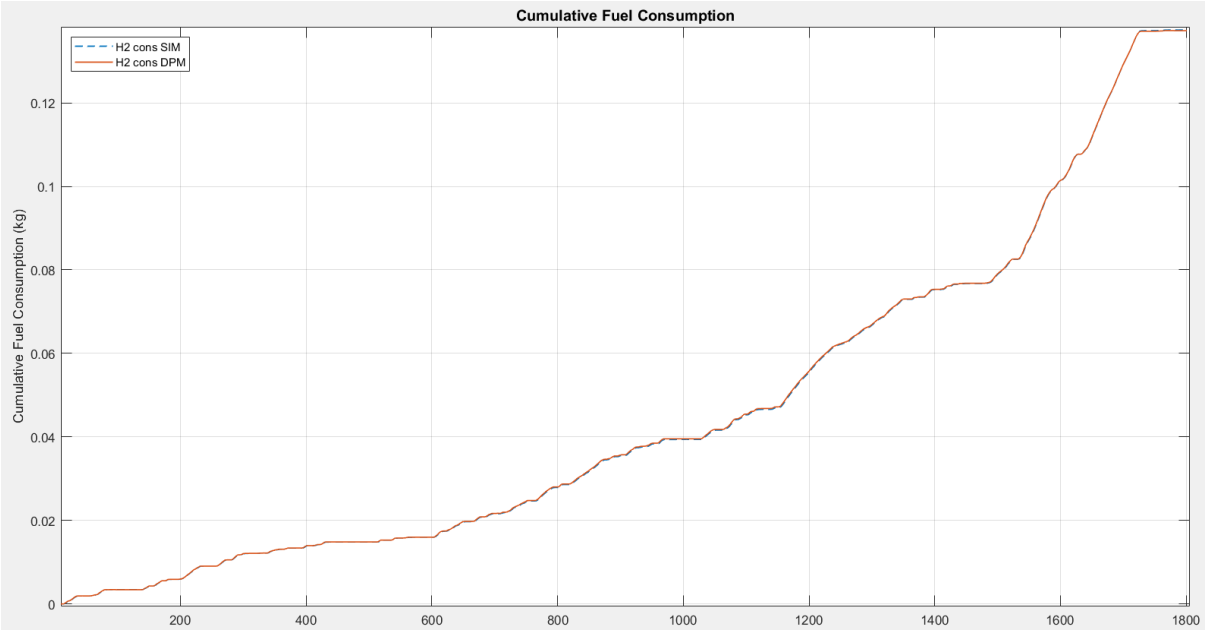

Figure 5.8 Comparison of fuel consumption in Simscape and in the DPM applying the same policy

5.3 Real-Time Neural Network Controller

The final test to be conducted is the real-time implementation of the NN controller obtained in chapter 4. To assess its performance, it is compared with the original PID controller implemented in the Simscape model, which is presented in Figure 5.9.

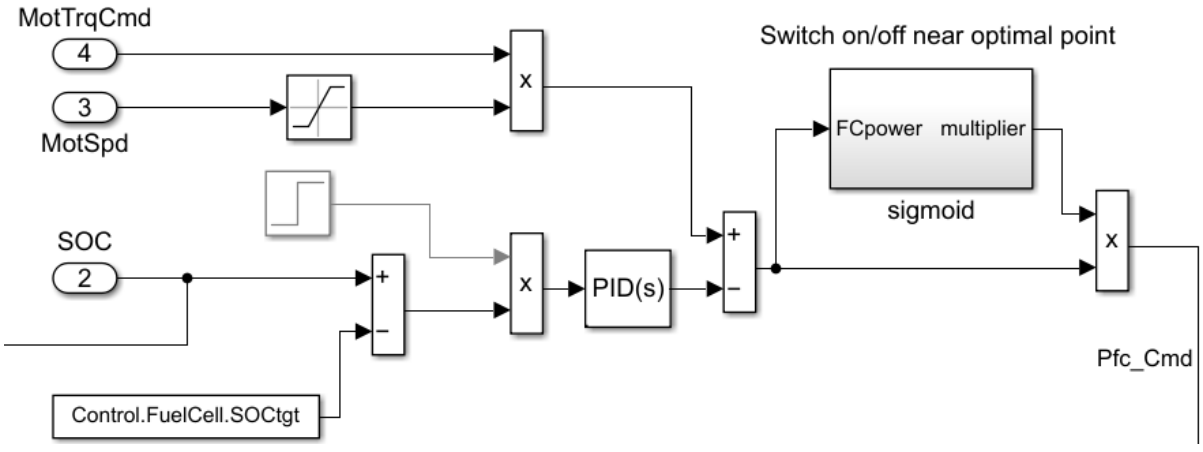

Figure 5.9 Original PID controller implemented in Simscape

In its original version, the PID controller is not able to guarantee the charge sustaining mode, and thus it is modified for such purpose. After a trial-and-error tuning approach, the gains of the PID controller were set as

$$K_P = 100;\ \ K_I = 10;\ \ K_D = 15$$

It is important to remark that the tuning process for processes with complex dynamics is one of the major issues with PID controllers. Gains that show a good performance for one cycle may not behave well for other cycles, and the controller needs to be re-tuned. Moreover, it is not possible to set constraints for the variables when using the PID controller.

The sigmoid block is used to turn on/off the fuel cell when the power request is close to its optimal value of 6.5 kW.

5.3.1 WLTP Class 3 Results

The NN controller was able to control the vehicle to successfully track the reference speed profile, as can be seen in Figure 5.10 which displays plots of the actual and reference vehicle speeds.

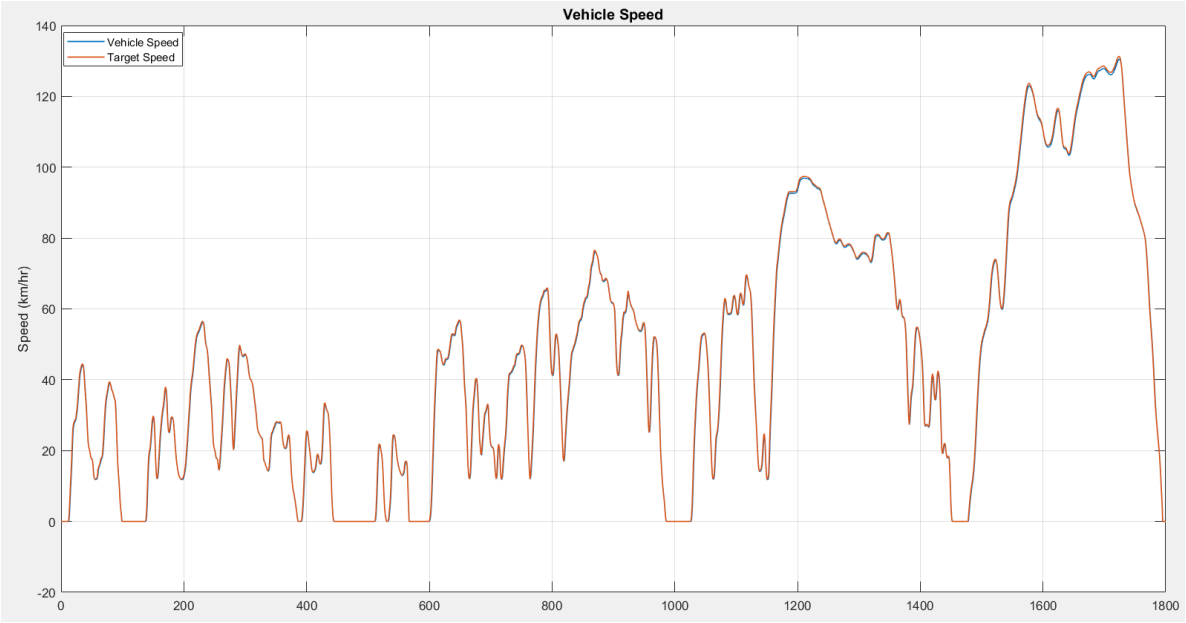

Figure 5.10 Vehicle speed using NN controller for the WLTP3 cycle

The vehicle tracks the reference speed with a delay of 0.2 seconds, due to the driver response and the vehicle dynamics. A maximum difference of 0.8 km/h between both curves occurs in the high velocity region. Similar performance was also obtained for the remaining control configurations and cycles; therefore, the profiles of actual and reference vehicle speed will not be shown in further simulations.

To better highlight the advantages of the proposed controller over the original one, the driving cycle is repeated periodically. The results obtained using the NN controller are shown in Figure 5.11

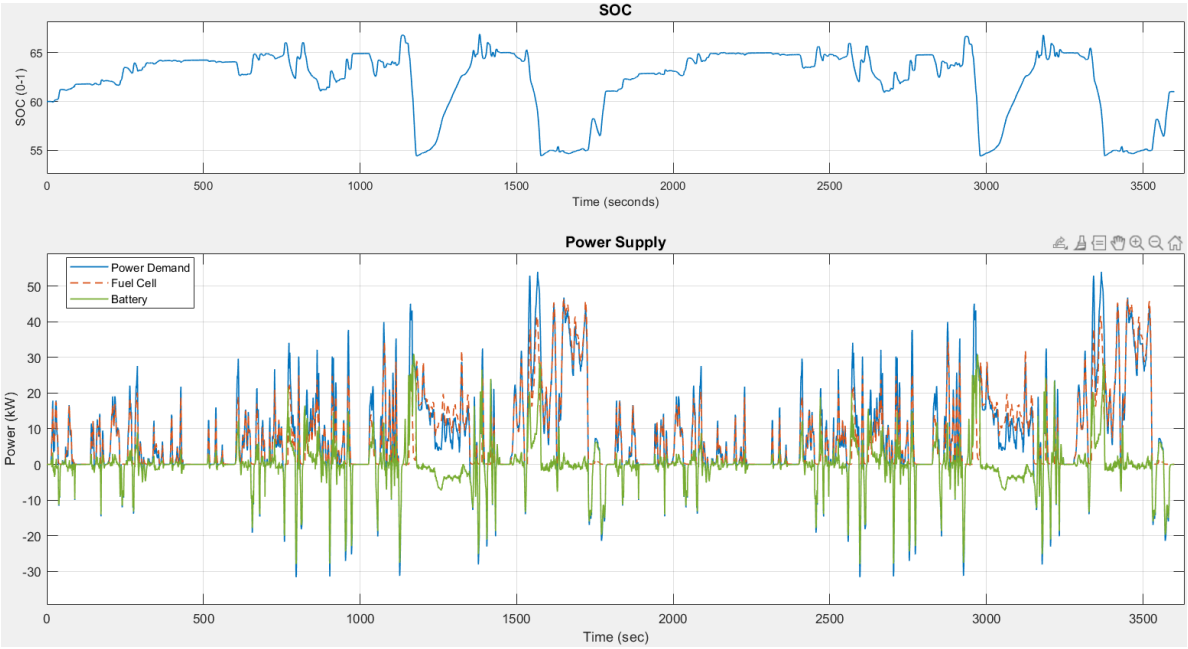

Figure 5.11 Results using NN controller for the WLTP3 cycle

The NN controller reproduces the DPM policy with small discrepancies in the SOC trajectory. Indeed, it goes out of the imposed limits of 0.55-0.65 in the most demanding regions, but ensures a final SOC value of 0.61 as it is seen in the DPM. The SOCs obtained with the PID and NN controllers are compared in Figure 5.12; note that the PID controller is not able to guarantee that the SOC remains within the established range nor the final state constraint.

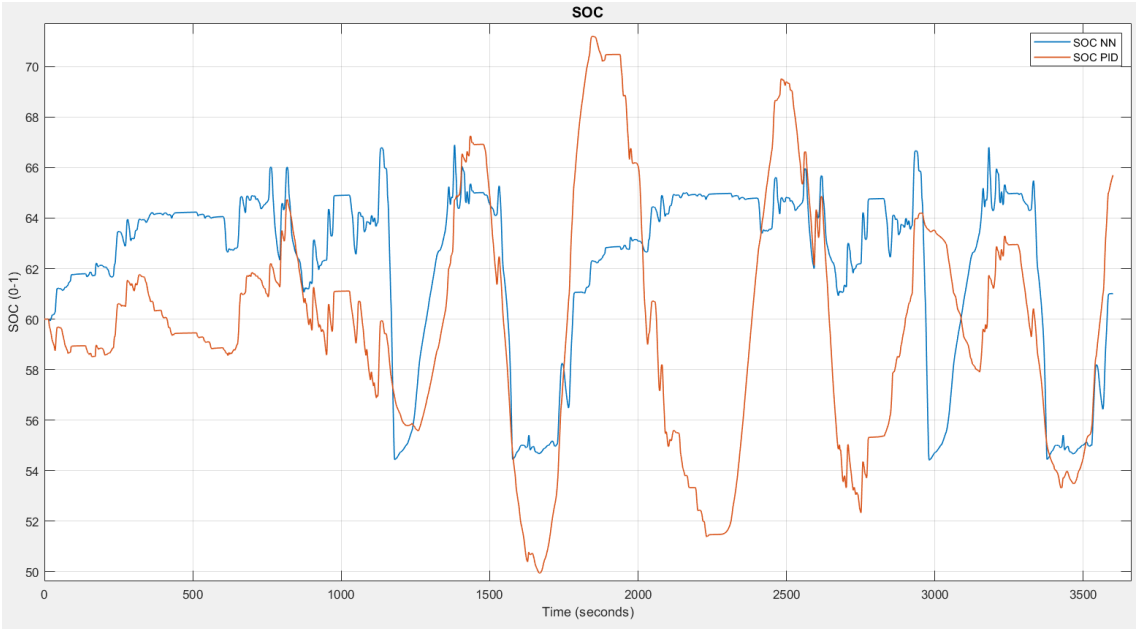

Figure 5.12 Comparison of SOC for the NN and PID controllers for the WLTP3 cycle

Another notable difference is the higher depth of discharge DoD that the battery experiences when the PID controller is used, which can be also seen from the Simscape results for the number of discharge cycles shown in Figure 5.13. This results in premature battery aging.

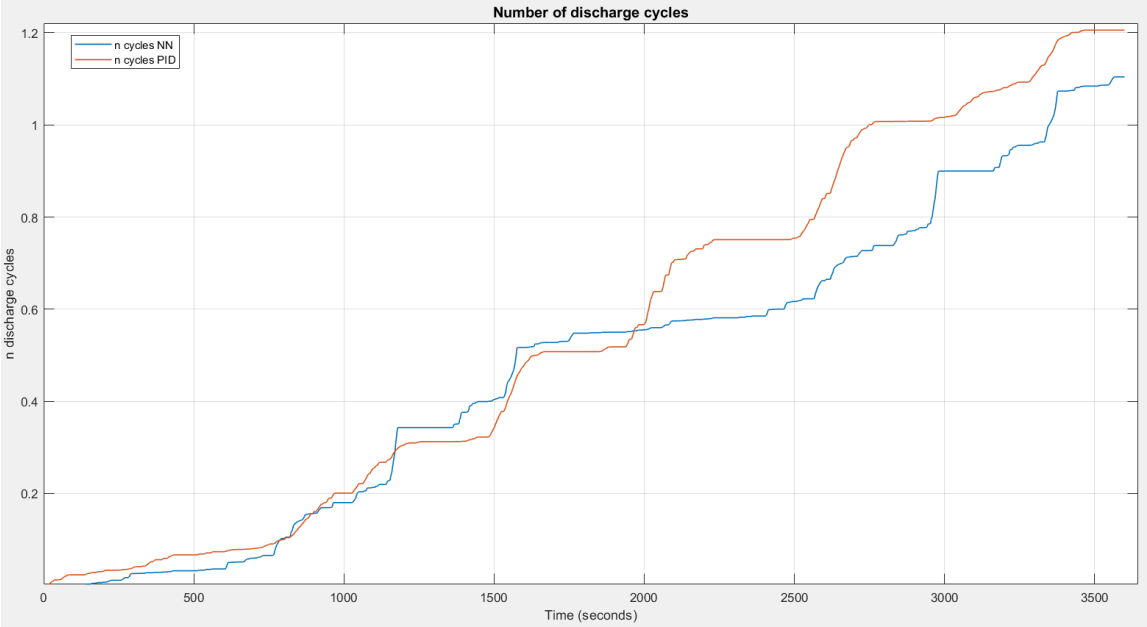

Figure 5.13 Number of battery discharge cycles for the NN and PID controllers for the WLTP3 cycle

The NN controller also shows a better performance regarding fuel consumption minimization, which is the most important objective, while guaranteeing to fulfill the established limits. Figure 5.14 compares fuel consumption in terms of final kg of hydrogen and in liters/100 km of GGE.

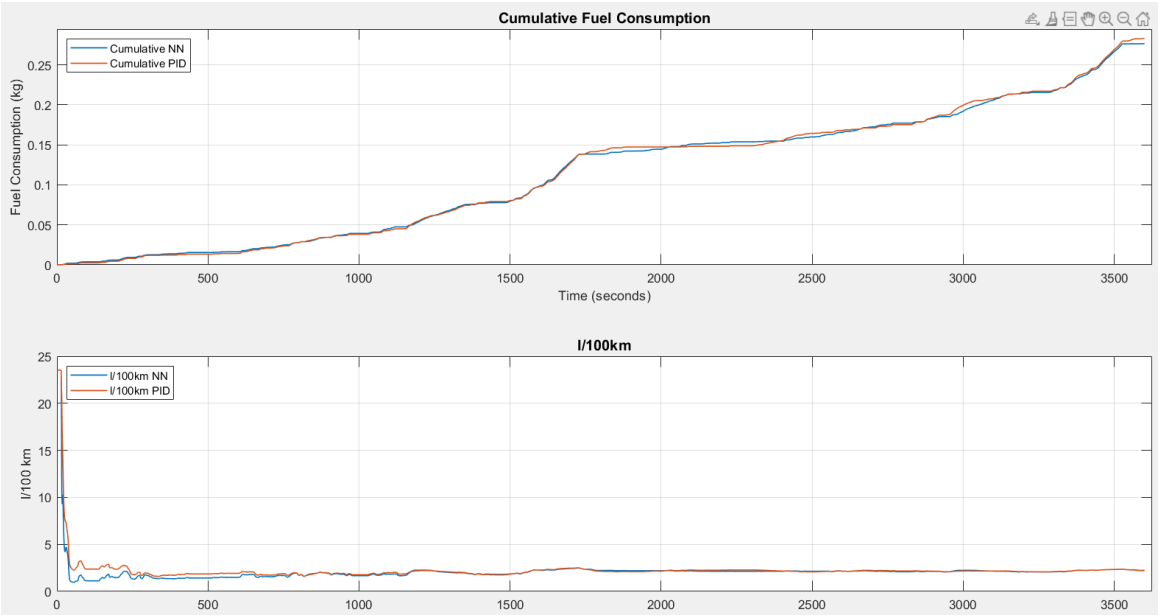

Figure 5.14 WLTP3 Fuel Consumption comparison NN and PID

After repeating the cycle twice, the NN controller achieves a lower fuel consumption, $H2_{NN} = 0.2764\,kg$ compared to $H2_{PID} = 0.2829\,kg$, and is also in correspondence with the DPM results computed offline.

5.3.2 FTP75 Results

The same procedure is used for the FTP75 driving cycle, which it is repeated twice to emphasize results that are shown in Figure 5.15.

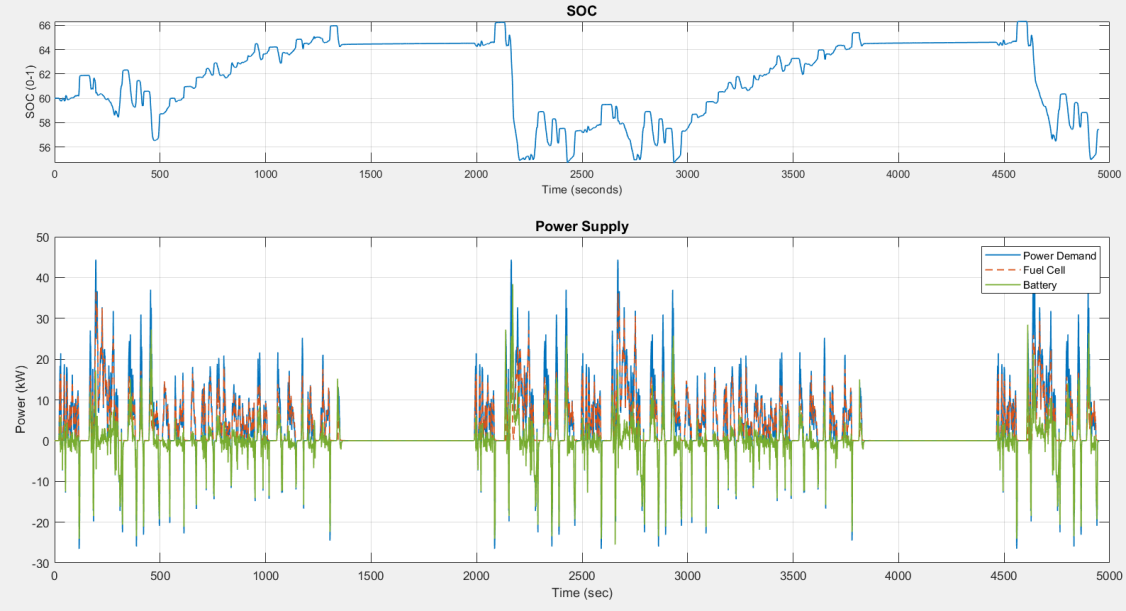

Figure 5.15 Results using the NN controller for the FTP75 cycle

For this case, the NN controller is not able to guarantee a final SOC of 0.60. This issue is probably due to the less aggressive behavior of the FTP75 with respect to the WLTP3. Indeed, the NN controller keeps the SOC at a lower value waiting for braking events to occur, but in this cycle the braking phases are much less. However, it should be noted that the SOC at time instant 2475, i.e., at the end of the first cycle is $SOC_{2475} = 0.5733$, which is very similar to the SOC at time instant 4950 obtained as $SOC_{4950} = 0.5745$.

This led to the conclusion that even if it is not achieved that the final SOC is equal to the initial one, the controller is robust enough to guarantee charge sustaining mode, recovering from the end of the first cycle without a reduction of the SOC value. The only difference is that it keeps the SOC at a slightly lower value, to take advantage of all the regenerative braking energy while not exceeding the higher limit of 0.65.

Compared with the PID controller, the NN controller still shows better performance for the battery life cycle. In figure 5.16 the two different SOCs are compared, where it can be seen that the DoD is higher for the PID, while the NN controller is more stable. This is also observed in the number of discharge cycles that are shown in Figure 5.17, with final values of 0.93 for the NN controller and 1.03 for the PID.

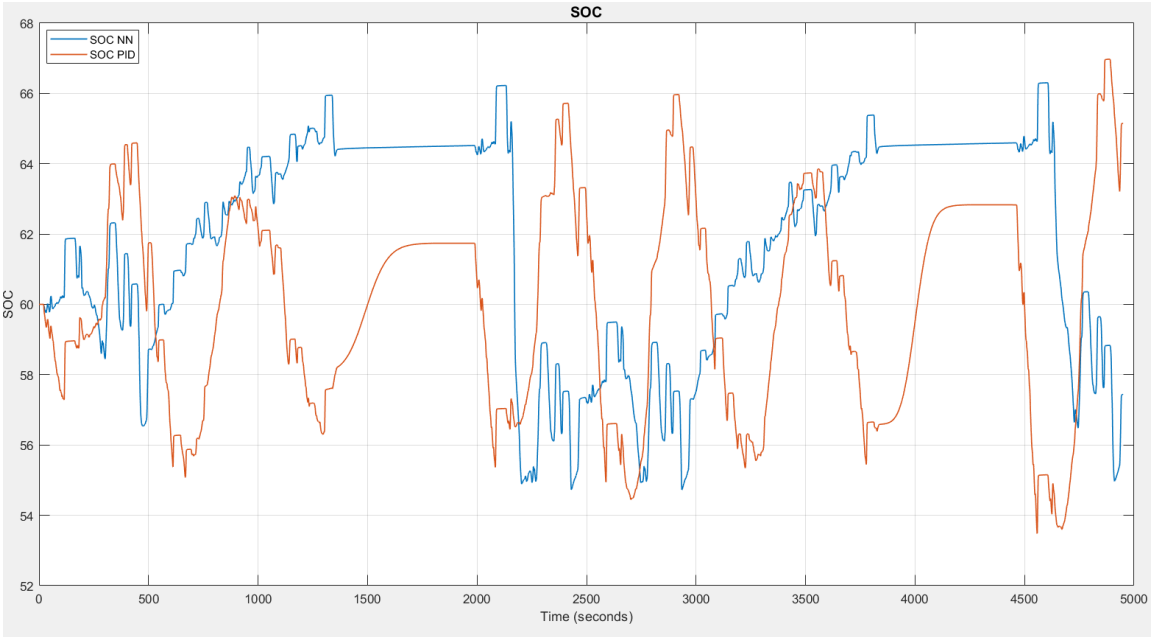

Figure 5.16 SOCs for the NN and PID controllers for the FTP75 cycle

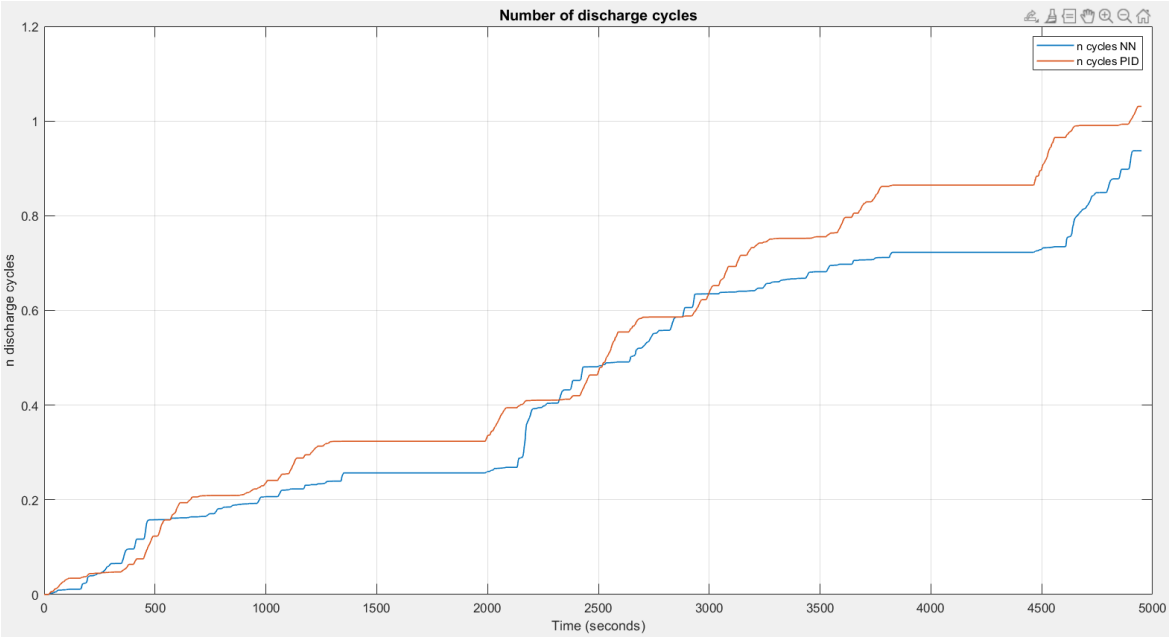

Figure 5.17 Numbers of battery discharge cycles for the NN and PID controllers for the FTP75 cycle

The fuel consumption results are shown in Figure 5.18. Comparing the results of the NN controller and of the DPM, the total hydrogen consumed in kg is very close to the one obtained offline. On the other hand, with respect to the PID controller it shows a better fuel economy, since $H2_{NN} = 0.1721\ kg$ and $H2_{PID} = 0.1807\ kg$

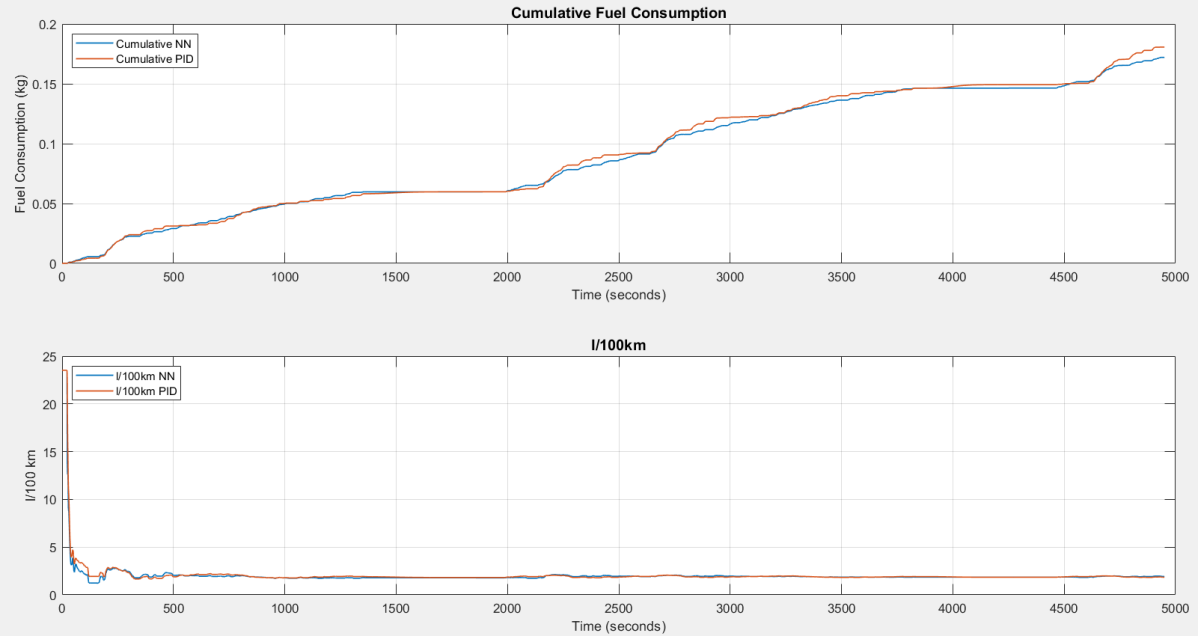

Figure 5.18 FTP75 Fuel Consumption for the NN and PID controllers for the FTP75 cycle

6 Conclusions and future work

This thesis proposed a controller based on a feedforward neural network (FFNN) for real-time implementation of an Energy Management Strategy, that determines the power split between the fuel cell and the battery to meet the load demand in a Fuel Cell Hybrid Electric Vehicle (FCHEV). Specifically, the FFNN was trained to reproduce the behavior of a DPM algorithm. In order to achieve this purpose, a quasi-static model is first created using the Toyota Mirai components and the Simscape dynamic model as reference. Results show that the quasi-static model was able to approximate the dynamic model with a satisfactory accuracy. Then, an optimization routine is implemented with the DPM algorithm to find the optimal policy that establishes such power split for two driving cycles, namely WLTP Class 3 and FTP75, while meeting all constraints imposed. As expected, the DPM uses as much regenerative braking as possible and both the fuel cell and the battery are operated in their most efficient regions.

Then, the data obtained for the two driving cycles was used to train the FFNN. After training the FFNN using the Matlab NN Toolbox, the neural network showed a very good performance in terms of RMSE between the estimated and real values of the DPM policy. To assess the discrepancies between the two models, it was also tested the direct implementation of the P_{FC} sequence determined by the DPM algorithm. Simscape model behaved slightly different from the quasi-static model used in DPM, but constraints were still meet and the fuel consumption obtained was equal for both models.

Finally, the neural network controller was implemented in place of the original PID controller, and their performances were further compared. The NN was able to adequately capture the behavior of the DPM algorithm, and outperformed the original PID controller in both cycles tested. To better remark the advantages of the proposed EMS, both WLTP3 and FTP75 cycles were run periodically. The hydrogen consumption was 6.5 g for the WLTP3 and 8.6 g for the FTP75, which represents more than 2% saving in the former and almost 5% in the latter, when considering Tank-to-Wheel performance. Expanding the analysis to the Well-To-Wheel emissions, the proposed controller has additional advantages such as the improvement of life cycles for both components; this is the major issue for the FCHEVs to be competitive with conventional vehicles, since the FCHEV have a higher footprint at first stages of life.

As future work, it will be attempted to further improve the performance of the NN controller, using a model which better reproduces the dynamic behavior. On the other hand, alternatives will be assessed to remove or improve the rule blocks incorporated in the controller implementation to handle exceptions; in particular, it will be evaluated to train a neural network to manages SOC values outside the DPM limits. Another issue observed in the results is the different behavior when lower aggressive cycles are used; this problem could be handled by using two different NN controllers depending on the speed profile recognized, namely Urban and Highway. Finally, it could be tested in real vehicles or prototype to assess its performance.

7 Bibliography

[1] A. Ajanovic and R. Haas, "Economic and Environmental Prospects for Battery Electric- and Fuel Cell Vehicles: A Review," *Fuel Cells*, vol. 19, no. 5, pp. 515–529, Oct. 2019, doi: 10.1002/fuce.201800171.

[2] Y. Huang *et al.*, "A review of power management strategies and component sizing methods for hybrid vehicles," *Renewable and Sustainable Energy Reviews*, vol. 96, pp. 132–144, Nov. 2018, doi: 10.1016/j.rser.2018.07.020.

[3] Q. Wang, M. Xue, B.-L. Lin, Z. Lei, and Z. Zhang, "Well-to-wheel analysis of energy consumption, greenhouse gas and air pollutants emissions of hydrogen fuel cell vehicle in China," *Journal of Cleaner Production*, vol. 275, p. 123061, Dec. 2020, doi: 10.1016/j.jclepro.2020.123061.

[4] M. Kannangara, F. Bensebaa, and M. Vasudev, "An adaptable life cycle greenhouse gas emissions assessment framework for electric, hybrid, fuel cell and conventional vehicles: Effect of electricity mix, mileage, battery capacity and battery chemistry in the context of Canada," *Journal of Cleaner Production*, vol. 317, p. 128394, Oct. 2021, doi: 10.1016/j.jclepro.2021.128394.

[5] R. J. Rieveley and B. P. Minaker, "Variable Torque Distribution Yaw Moment Control for Hybrid Powertrains," Apr. 2007. doi: 10.4271/2007-01-0278.

[6] G. Mohan, F. Assadian, and S. Longo, "An Optimization Framework for Comparative Analysis of Multiple Vehicle Powertrains," *Energies*, vol. 6, no. 10, pp. 5507–5537, Oct. 2013, doi: 10.3390/en6105507.

[7] C. C. Chan, A. Bouscayrol, and K. Chen, "Electric, Hybrid, and Fuel-Cell Vehicles: Architectures and Modeling," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 2, pp. 589–598, Feb. 2010, doi: 10.1109/TVT.2009.2033605.

[8] G. Rizzoni, L. Guzzella, and B. M. Baumann, "Unified modeling of hybrid electric vehicle drivetrains," *IEEE/ASME Transactions on Mechatronics*, vol. 4, no. 3, pp. 246–257, 1999, doi: 10.1109/3516.789683.

[9] D A J Rand and R M Dell, *Hydrogen Energy: Challenges and Prospects*. Cambridge: Royal Society of Chemistry, 2007. doi: 10.1039/9781847558022.

[10] M. Jouin, R. Gouriveau, D. Hissel, M. C. Péra, and N. Zerhouni, "PEMFC aging modeling for prognostics and health assessment," *IFAC-PapersOnLine*, vol. 48, no. 21, pp. 790–795, 2015, doi: 10.1016/j.ifacol.2015.09.623.

[11] P. J. Berlowitz and C. P. Darnell, "Fuel Choices For Fuel Cell Powered Vehicles," Mar. 2000. doi: 10.4271/2000-01-0003.

[12] M. Ehsani, Y. Gao, S. Longo, and K. M. Ebrahimi, *Modern Electric, Hybrid Electric, and Fuel Cell Vehicles*. 2018.

[13] G. J. Offer, D. Howey, M. Contestabile, R. Clague, and N. P. Brandon, "Comparative analysis of battery electric, hydrogen fuel cell and hybrid vehicles in a future sustainable road transport system," *Energy Policy*, vol. 38, no. 1, pp. 24–29, Jan. 2010, doi: 10.1016/j.enpol.2009.08.040.

[14] R. Bellman, "Dynamic Programming," *Science*, vol. 153, no. 3731, pp. 34–37, Jul. 1966, doi: 10.1126/science.153.3731.34.

[15] F. L. Lewis, D. L. Vrabie, and V. L. Syrmos, *Optimal Control*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2012. doi: 10.1002/9781118122631.

[16] O. Sundstrom and L. Guzzella, "A generic dynamic programming Matlab function," in *2009 IEEE International Conference on Control Applications*, Jul. 2009, pp. 1625–1630. doi: 10.1109/CCA.2009.5281131.

[17] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989, doi: 10.1016/0893-6080(89)90020-8.

[18] L. Fausett, *Fundamentals of neural networks: Architecture, algorithms and applications*. Prentice-Hall, 1994.

[19] R. Hecht-Nielsen, "Neurocomputing: picking the human brain," *IEEE Spectrum*, vol. 25, no. 3, pp. 36–41, Mar. 1988, doi: 10.1109/6.4520.

[20] S. Haykin, *Neural Networks: A comprehensive foundation*. John Wiley & Sons, 2001.

[21] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, Dec. 1989, doi: 10.1007/BF02551274.

[22] R. Battiti, "First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method," *Neural Computation*, vol. 4, no. 2, pp. 141–166, Mar. 1992, doi: 10.1162/neco.1992.4.2.141.

[23] M. M. Davari, J. Jerrelind, and A. Stensson Trigell, "Energy efficiency analyses of a vehicle in modal and transient driving cycles including longitudinal and vertical dynamics," *Transportation Research Part D: Transport and Environment*, vol. 53, pp. 263–275, Jun. 2017, doi: 10.1016/j.trd.2017.04.019.

[24] G. Karavalakis *et al.*, "Criteria Emissions, Particle Number Emissions, Size Distributions, and Black Carbon Measurements from PFI Gasoline Vehicles Fuelled with Different Ethanol and Butanol Blends," Apr. 2013. doi: 10.4271/2013-01-1147.

[25] S. Miller, "Fuel Cell Vehicle Model in Simscape." 2021. Accessed: Aug. 01, 2021. [Online]. Available: https://github.com/mathworks/Fuel-Cell-Vehicle-Model-Simscape/releases/tag/21.2.1.3

[26] H. Pacejka, *Tire and Vehicle Dynamics*. Elsevier, 2012. doi: 10.1016/C2010-0-68548-8.

[27] R. Rajamani, "Longitudinal Vehicle Dynamics," 2012, pp. 87–111. doi: 10.1007/978-1-4614-1433-9_4.

[28] A. Soofastaei, S. M. Aminossadati, M. M. Arefi, and M. S. Kizil, "Development of a multi-layer perceptron artificial neural network model to determine haul trucks energy consumption," *International Journal of Mining Science and Technology*, vol. 26, no. 2, pp. 285–293, Mar. 2016, doi: 10.1016/j.ijmst.2015.12.015.

[29] T. Teng, X. Zhang, H. Dong, and Q. Xue, "A comprehensive review of energy management optimization strategies for fuel cell passenger vehicle," *International Journal of Hydrogen Energy*, vol. 45, no. 39, pp. 20293–20303, Aug. 2020, doi: 10.1016/j.ijhydene.2019.12.202.

[30] T. Fletcher, R. Thring, and M. Watkinson, "An Energy Management Strategy to concurrently optimise fuel consumption & PEM fuel cell lifetime in a hybrid vehicle," *International Journal of Hydrogen Energy*, vol. 41, no. 46, pp. 21503–21515, Dec. 2016, doi: 10.1016/j.ijhydene.2016.08.157.

[31] J. Wang *et al.*, "Cycle-life model for graphite-LiFePO4 cells," *Journal of Power Sources*, vol. 196, no. 8, pp. 3942–3948, Apr. 2011, doi: 10.1016/j.jpowsour.2010.11.134.

[32] X. Liu *et al.*, "From NEDC to WLTP: Effect on the Energy Consumption, NEV Credits, and Subsidies Policies of PHEV in the Chinese Market," *Sustainability*, vol. 12, no. 14, p. 5747, Jul. 2020, doi: 10.3390/su12145747.