UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Industrial

Analysis and Implementation of Control Strategies for Energy Management

Trabajo Fin de Máster

Máster Universitario en Ingeniería Industrial

AUTOR/A: Mateu Monterde, Roberto

Tutor/a: Montalva Subirats, José Miguel

Supervisor/a: HomChaudhuri, Baisravan

CURSO ACADÉMICO: 2022/2023

ACKNOWLEDGEMENT

I am deeply grateful to Dr. Baisravan HomChaudhuri, my project supervisor, for his guidance and support.

I would also like to extend my appreciation to Ignacio Iranzo Juan, my colleague. His unwavering assistance, particularly throughout my college years, has been instrumental in overcoming challenges and achieving milestones.

Furthermore, I am thankful to my partner, Beatriz Morales Azara, for her dedicated help and determined moral support.

ABSTRACT

This project deals with the study of a hierarchical robust energy management strategy. This energy optimization strategy is applied to electric vehicles in the presence of uncertainty, which in turn receive data from their environment.

Keywords: management, energy, control, robust, hierarchical, electric vehicle, uncertainty, optimization.

RESUMEN

Este trabajo trata del estudio de una estrategia de gestión energética robusta jerárquica. Esta estrategia de optimización energética se aplica a vehículos eléctricos en presencia de incertidumbre que a su vez reciben datos de su entorno.

Palabras clave: gestión, energía, control, robusto, jerárquico, vehículo eléctrico, incertidumbre, optimización.

RESUM

Aquest treball tracta de l'estudi d'una estratègia de gestió energètica robusta jeràrquica. Aquesta estratègia d'optimització energètica s'aplica a vehicles elèctrics en presència d'incertesa que al mateix temps reben dades del seu entorn.

Paraules clau: gestió, energia, control, robust, jeràrquic, vehicle elèctric, incertesa, optimització.

REPORT

CONTENTS PAGE

FIGURES

TABLES

1 Introduction

1.1 Introduction

During the Autonomous Systems and Robotics and the Industrial Engineering Masters, several subjects studied are related to control processes, such as Modern Control Systems, Industrial Instrumentation and Control, or Robotics. All these courses focus on the term "Control." The concept of control refers to mastery over the machine, whether it involves operating a machine, automating an industrial process with robots, or in this instance, enhancing the performance of an electric vehicle by managing and optimizing its battery and general functionality.

In the area of transportation, which is evolving by leaps and bounds, a remarkable change is taking place because society is transitioning and adopting the use of electric vehicles as a solution to environmental and energy sustainability problems. Internal combustion engines, despite the fact that they will continue to be present in the future, are gradually giving way to electric propulsion systems. “EU ban on the sale of new petrol and diesel cars from 2035” (Parliament, 2022).

This is the reason why the need to optimize the performance, efficiency and autonomy of these vehicles arises. In order to carry out this task, it is necessary to develop control strategies that allow improving both the lifespan of the vehicle components and the battery.

1.2 Motivation

This project is part of the research of Dr. Baisravan HomChaudhuri. Dr. HomChaudhuri focuses his studies on control systems and optimization strategies. Specifically, one of his main interests in recent years has been control strategies for various vehicles.

One such study, carried out by Ph.D. student Seyedeh Mahsa Sotoudeh, a student of Dr. HomChaudhuri, focused on Velocity optimization and energy management of hybrid electric vehicles. (Sotoudeh & HomChaudhuri, Velocity Optimization and Robust Energy Management of Connected Power-Split Hybrid Electric Vehicles, 2022).

Dr. HomChaudhuri pointed out at the beginning of this project that it would be interesting to apply a control similar to the one carried out by the student Seyedeh Mahsa Sotoudeh, but in this case applied to electric vehicles.

The study of these control strategies is very relevant due to their versatility, since they can be applied to a multitude of vehicles. The motivation for the project was born with the need to improve the efficiency of electric vehicle batteries. In addition to the abovementioned, the project also studies vehicle modeling and its subsequent simplification (linearization) to reduce the computational cost.

1.3 Objectives

The project is composed of a series of objectives that aim to enhance the performance and efficiency of Electric Vehicles (EVs). The main goal involves designing a detailed control strategy which is inspired by the effective approach demonstrated in Hybrid Electric Vehicles (HEVs). This strategy will be adapted and implemented to Electric Vehicles.

In order to implement the control strategy, a system model for electric vehicles will be developed. This model will combine general and additional dynamics for accurately representing Electric Vehicles. Building upon this model, the project will explore the simplification of these dynamics into a linear dynamic model.

Another objective is to explore a hierarchical control approach. The project aims to come up with a strategy that effectively controls the battery, to extend significantly its range and lifespan, as well as to enhance the overall efficiency of electric vehicles.

The last objective of the project is to develop and publish a beta version of this control code. This step ensures the continuity of the project, providing a foundation for future development succeeding in the efficiency of electric vehicles.

1.4 Structure

In Chapter 2 "Problem to be solved", the main problem of the project is explained: bridge the existing energy management control proved in HEVs to EVs.

In Chapter 3 "System Model", every equation used and developed is explained. This chapter shows the evolution of each model, from the first assumptions until its testing. These models take into account the longitudinal dynamics, battery dynamics, and also thermal dynamics. At the end of this chapter, a software tool called SINDyC is used to linearize the first system model.

In Chapter 4 "Energy Management Strategy, the optimal control problem is explained both in continuous and discrete time. Also, the suitability of the software library used is shown. And finally, there is an overview of the explored hierarchical control.

In Chapter 5 "Developed code", the code developed with the IPOPT software library is shown.

In Chapter 6 "Results and conclusion", three tests are shown and explained. These experiments aim to evaluate the explored hierarchical control for the high- and low-level controllers. Also, the conclusion of the project is stated.

In Chapter 7 "Future potential projects", there are some possible future projects that aim to extend this project.

In Chapter 8 "Budget", an estimation of the total cost of the project is shown.

Finally, i**n Chapter 9 "References",** the sources of information of this project are displayed.

2 Problem to be solved

2.1 Introduction to the problem

As previously explained, this work is the continuation of a line of research carried out by Dr. Baisravan HomChaudhuri. For example, (Sotoudeh & HomChaudhuri, Velocity Optimization and Robust Energy Management of Connected Power-Split Hybrid Electric Vehicles, 2022) was carried out by PhD student Mahsa. It demonstrates the improvement of the efficiency of hybrid vehicles. This project aims to analyze and implement a similar control to electric vehicles.

2.2 Details of the problem

To carry out this energy control strategy, it is first necessary to develop a good model of the system. For this, it is important to ensure that the equations that govern this model are correct and applicable.

In addition to good modeling, it is worth noting the importance of the computational cost of optimization programs. Therefore, another problem addressed in this project is the possibility of linearizing the models mentioned above.

Lastly and most importantly, the development of the optimization program. Based on previous works, the main problem is to bridge a similar solution presented for efficiently managing hybrid electric vehicles into electric vehicles.

These programs will be published to ensure their continuity in the future.

2.3 Tools used

To carry out the modeling of the system, Matlab has been used, specifically Simulink, a visual programming environment.

SINDy (Sparse Identification of Non-Linear dynamics) has been used for linearization, specifically SINDyC (with control variables). The possibility of using the Koopman operator was also explored, although it was finally ruled out since SINDy provided better results.

Regarding the optimization problem, IPOPT (Interior Point OPTimizer) has been used. It is a software library; it has been used in the Julia environment. IPOPT provides very good results for large-scale nonlinear optimization problems. Other types of optimizers were also explored, such as GPOPS or Yalmip, both Matlab libraries, but in the end, they were discarded due to their limitations in problem size and computational cost.

3 System Model

3.1 General dynamics model

3.1.1 Equations

This first model that includes the general dynamics of the vehicle and battery was obtained from the reference (Sotoudeh & HomChaudhuri, Velocity Optimization and Robust Energy Management of Connected Power-Split Hybrid Electric Vehicles, 2022).

This model simply considers the acceleration and braking longitudinal dynamics of the vehicle and, in turn, the state of charge of the battery.

So, the **state variables** are four in this case:

- Position "*s*"
- Velocity "*v*"
- Acceleration "*a*"

And the **control variables** are two:

- Acceleration force "*F_acc*"
- Braking force "*F_brake*"

The equation for the longitudinal dynamics of a vehicle:

$$\frac{dv}{dt} = \frac{1}{m_e} * (F_{acc} - F_{break} - 0.5 * \rho * A * \ C_d * v(t)^2 - C_r * m * g * \cos(\theta) - m * g * \sin(\theta) \quad (1)$$

Here, m is the vehicle's mass, m_e is effective mass considering inertial losses of powertrain (between 7% and 9%).

For other losses, C_r is the rolling resistance that depends on the road angle θ (positive uphill).

Regarding the aerodynamics, C_d is the drag coefficient, A is the area of the projection of the car, and ρ is the density of the air.

The control variables in this equation are:

$$\begin{gathered} F_{acc}\ is\ the\ acceleration\ force\ of\ the\ car \\ F_{brake}\ is\ the\ braking\ force\ of\ the\ car \\ F_{acc}, F_{brake} \geq 0 \\ If\ F_{acc} \neq 0\ then\ F_{brake} = 0 \\ If\ F_{brake}\ \neq 0\ then\ F_{acc} = 0 \end{gathered} \quad (2)$$

The equation for the State of Charge (SOC) of an electric vehicle is the following:

$$\frac{dSOC}{dt} = -\frac{V_{oc} - \sqrt{{V_{oc}}^2 - 4 * R_{batt} * (P_{acc} - P_{reg})}}{2 * C_{batt} * R_{batt}} \quad (3)$$

Where V_{oc} is the open-circuit voltage of the battery (considered constant), R_{batt} is the internal resistance of the battery (considered constant), and C_{batt} is the capacity of the battery (considered constant).

$$\begin{gathered} P_{acc}\ is\ the\ tractive\ power\ of\ the\ car \\ P_{acc} = \ F_{acc} * v(t) \\ P_{reg}\ is\ the\ regenerated\ power\ of\ the\ car \\ P_{reg} = \eta * F_{break} * v(t) \end{gathered} \quad (4)$$

The parameters used for these equations are the following:

For the longitudinal dynamics:

- $m = 2063\ kg$
- $m_e = 1.08 * m = 2228.04\ kg$, taking 8% of the vehicle's mass as inertial losses
- $C_r = 0.011$
- $\theta = 0\ rad$, this variable was taken constant throughout the whole project.
- $A = 2.22\ m^2$
- $\rho\ and\ g$ were taken in standard conditions.

For the state of charge of the battery:

- $V_{oc} = 400\ V$, constant throughout the whole project.
- $R_{batt} = 0.044\ \Omega$
- $C_{batt} = 738000\ A * s$

3.1.2 Final model obtained

The control-oriented system model obtained from the previous equations is the following:

$$\begin{pmatrix} \dot{s} \\ \dot{v} \\ \dot{SOC} \end{pmatrix} = \begin{pmatrix} v \\ \frac{1}{m_e} * (F_{acc} - F_{brake} - 0.5 * \rho * A * \ C_d * v(t)^2 - C_r * m * g * \cos(\theta) - m * g * \sin(\theta) \\ -\frac{V_{oc} - \sqrt{{V_{oc}}^2 - 4 * R_{batt} * (F_{acc} * v - \eta * F_{break} * v)}}{2 * C_{batt} * R_{batt}} \end{pmatrix} \quad (5)$$

3.1.3 Simulink model

To model the system in Simulink, the following blocks have been used (as shown in Figure 1):

To introduce the inputs, acceleration force and braking force, constant torque blocks have been introduced. These blocks have then been inserted into a switch block. This switch block also receives a clock block, in order to toggle the switch at a certain time. Once the signal leaves the switch block, it enters another block called "Matlab function" Figure 2, which calculates the force per unit mass.

This force per unit mass is then introduced in another "Matlab function" block in which the equations from the previous section are found. In the case of Figure 3, the acceleration leaves the block and then arrives at the integration block. The velocity comes out of that integration block, which is entered into another integration block and the position is obtained. Other blocks called "scopes" also appear and are able to visualize different system variables and thus be able to verify the correct operation. In Figure 4 , the "Matlab function" outputs the $\frac{SOC}{dt}$ which goes to another integrator block to obtain the SOC.

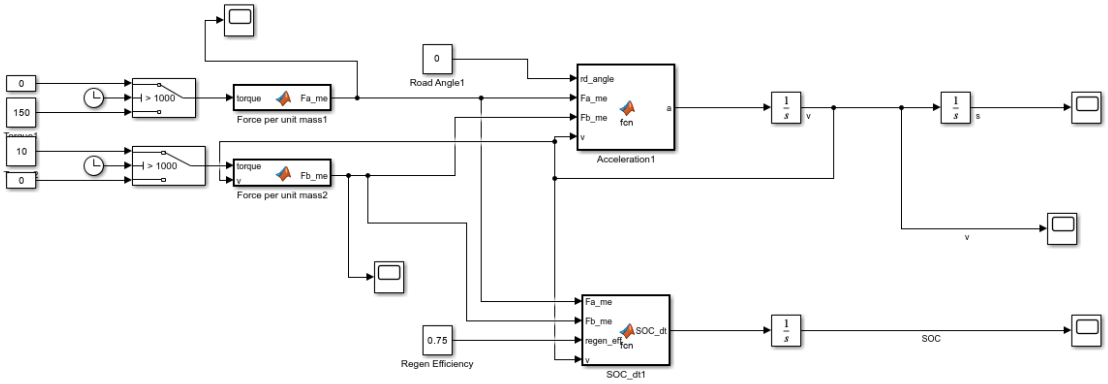

Figure 1 Simulink, General dynamics model

```
function Fa_me  = fcn(torque)
%Params
m_e=2063*1.08; %kg inertial losses
wh_rad=0.13; %m

%EQN
Fa_me=torque/(m_e*wh_rad);

```

Figure 2 Simulink, General dynamics function Fa/me

```
function a    = fcn(rd_angle,Fa_me,Fb_me,v)
%Params
m=2063;          %car mass [kg]
me=2063*1.08;    %inertial losses [kg]
ro=1.293;        %air density [kg/m^3]
A=2.22;          %area [m^2]
Cd=0.23;         %drag coeff
ang=rd_angle;    %road angle [rad]
Cr=0.011;        %rolling resistance coeff
g=9.81;          %gravity [m/s^2]


F_brake=Fb_me;
%EQN
if v<=1
    F_brake=0;
    a=Fa_me-F_brake;
else
    a=Fa_me-F_brake-(1/(2*me))*ro*A*Cd*v^2-Cr*(m/me)*g*cos(ang)-m*g*sin(ang);
end
```

Figure 3 Simulink, General dynamics function acc

```
function SOC_dt  = fcn(Fa_me,Fb_me,regen_eff,v)
%Params
Voc=400;             %Open-circuit voltage [V]
Rbatt=0.044;         %Internal battery resistance [Ohm]
C_bat=205*3600;      %Battery capacity [A*s]
me=2063;             %EV mass (inertial losses) [kg]

%EQN
SOC_dt=-(Voc-sqrt(Voc^2-4*Rbatt*me*(Fa_me*v-Fb_me*v*regen_eff)))/(2*Rbatt*C_bat);
```

Figure 4 Simulink, General dynamics function SOC/dt

3.1.4 Model testing

To test the model, an input torque of $150\ N*m$ was applied to the system model for a duration of 1000 seconds. Subsequently, a braking torque of $100\ N*m$ was engaged after 1000 seconds, persisting until a full stop.

The initial conditions are: $s(0) = 0\ m; v(0) = 0\ \frac{m}{s}; SOC(0) = 90\ \%$

The objective is to verify the equations outlined in the preceding sections. The anticipated outcome involves the vehicle experiencing acceleration until reaching a consistent velocity, followed by deceleration until a complete stop. Concerning the battery, the expectation is that it will deplete during acceleration and subsequently undergo a minor recharge while undergoing braking.

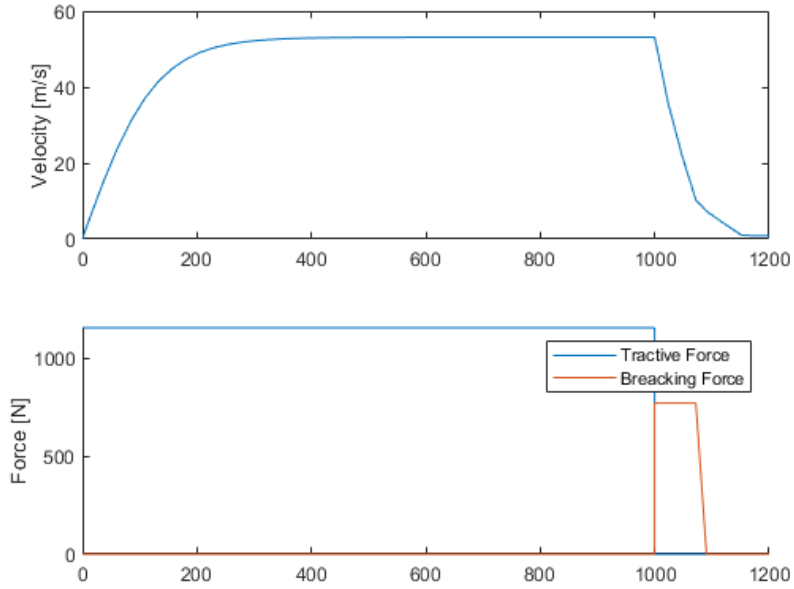

Figure 5 Testing, General dynamics Force[N] and Velocity [m/s]

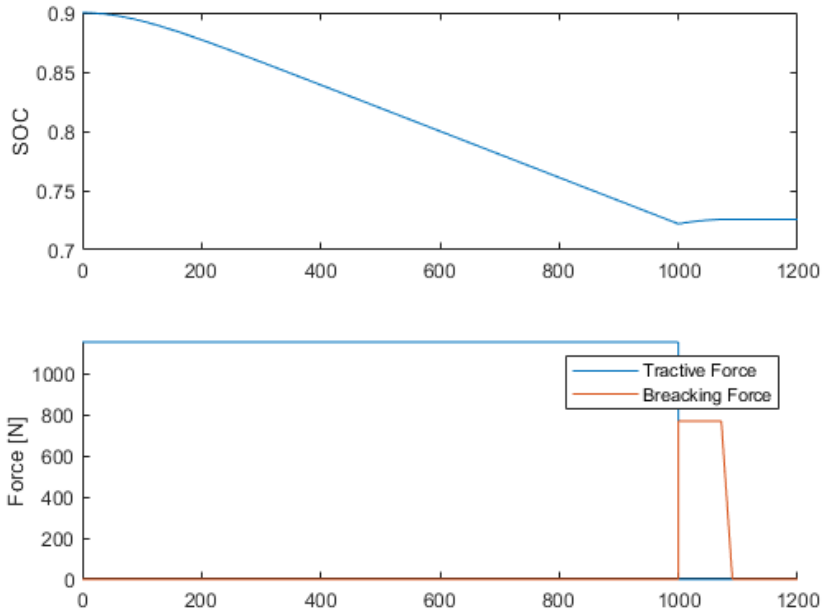

Figure 6 Testing, General dynamics Force[N] and SOC

The results in Figure 5 and Figure 6 align with the anticipated expectations.

The state of charge at the end of the test is: $SOC(t = 1200) = \ 72.5755\ \%$

3.2 Thermal model

3.2.1 Thermal equations

This section introduces a new control variable: forced heat dissipation, which appears in the battery state-of-charge equation as the electrical power required to carry out this heat dissipation, and a new state variable: temperature.

$$\frac{dSOC}{dt} = -\frac{V_{oc} - \sqrt{{V_{oc}}^2 - 4 * R_{batt} * \left(P_{acc} - P_{reg} - P_{temp}\right)}}{2 * C_{batt} * R_{batt}}$$

$$where\ P_{temp}\ is\ the\ electric\ electric\ power\ necessary\ to\ produce\ the\ heat\ flow\ rate\ \dot{Q}\ at\ the\ battery. \quad (6)$$

$$\dot{Q} = P_{temp} * E$$

In the above equation, $\dot{Q}$ is the forced heat dissipated and E is the efficiency of the heat dissipation system, that is, watts of heat dissipated per watts of electrical power.

The battery is described as a lumped mass with heat capacity $C_{th,batt}$. This leads to the following equation for the battery temperature T:

$$\frac{dT}{dt} = \frac{1}{C_{th,batt}} * (I^2 * R_{batt} + \dot{Q}_{amb} + \dot{Q}) \quad (7)$$

Two equations can be derived from this one, the first:

$$\frac{dT}{dt} = \frac{\frac{V_{oc} - \sqrt{V_{oc}^2 - 4 * \left(P_{acc} + P_{regen} + P_{temp}\right) * R_{batt}}}{4 * R_{batt}} + \dot{Q}_{amb} + \dot{Q}}{C_{th,batt}} \quad (8)$$

However, considering that electrical intensity can be written as electrical power divided by voltage (in direct current), from equation (7)

$$\frac{dT}{dt} = \frac{1}{C_{th,batt}} * ((\left(\frac{P_{acc}}{V_{oc}}\right)^2 + \left(\frac{P_{regen}}{V_{oc}}\right)^2) * R_{batt} + \dot{Q}_{amb} + \dot{Q}) \quad (9)$$

Substituting acceleration power and braking power:

$$\frac{dT}{dt} = \frac{1}{C_{th,batt}} * (((\frac{F_{acc} * v}{V_{oc}})^2 + (\frac{F_{brake} * \eta * v}{V_{oc}})^2) * R_{batt} + \dot{Q}_{amb} + \dot{Q}) \quad (10)$$

To calculate the heat transferred by convection from the battery to the ambient:

$$h = 2.38 * (u_\infty)^{0.89}$$
$$where\ h\ is\ the\ heat\ transfer\ per\ m^2 and\ °C\ \left[\frac{W}{m^2 * °C}\right] \quad (11)$$
$$and\ u_\infty\ is\ the\ air\ velocity\ in\ [\frac{m}{s}]$$

That leads to the following equation:

$$\dot{Q}_{amb} = 2.38 * (v)^{0.89} * 2.38 * A_{batt} * (T - T_{amb}) \quad (12)$$

The parameters chosen for this additional dynamic are:

- $C_{th,batt} = 1000 \frac{J}{kg*K}$
- $T_{amb} = 25\ °C$
- $E = 0.75$

3.2.2 Simulink model

The Simulink model is the same as in the previous section, but a new "Matlab function" block has been added, described in Figure 7. In addition, a forced heat transfer (control variable) constant block has also been included. This signal goes to a switch block to choose when to activate it. Finally, the “Matlab function" block that calculates the $\frac{SOC}{dt}$ has been updated to include the new control variable, Figure 8.

The "Matlab function" in Figure 9 block contains the equations described in the previous section and three signals come out of it: heat exchanged with the environment, forced heat and temperature derivative. This last signal goes to an integrator, from which the battery temperature is output.

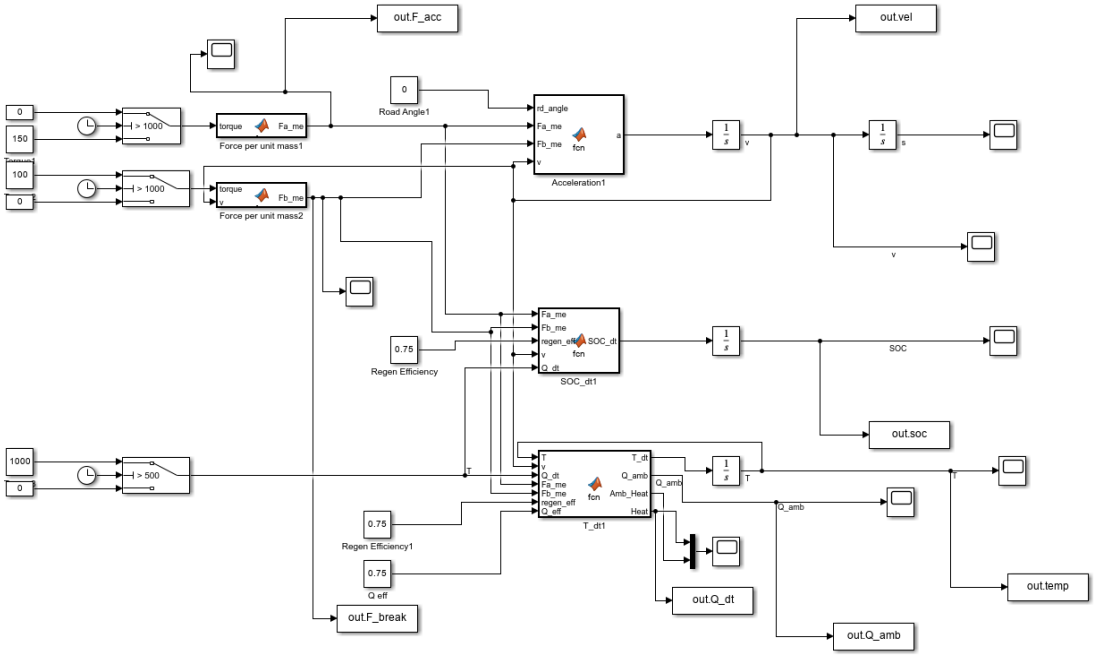

Figure 7 Simulink, Thermal dynamics model

```
function SOC_dt  = fcn(Fa_me,Fb_me,regen_eff,v,Q_dt)
%Params
Voc=400;            %Open-circuit voltage [V]
Rbatt=0.044;        %Internal battery resistance [Ohm]
C_bat=205*3600;     %Battery capacity [A*s]
me=2063;            %EV mass (inertial losses) [kg]

%EQN
SOC_dt=-(Voc-sqrt(Voc^2-4*Rbatt*(Fa_me*v*me-Fb_me*v*me*regen_eff+Q_dt)))/(2*Rbatt*C_bat);
```

Figure 8 Simulink, General dynamics function SOC

```
function [T_dt,Q_amb,Amb_Heat, Heat]  = fcn(T,v,Q_dt,Fa_me,Fb_me,regen_eff,Q_eff)
%Params
Voc=400;                %Open-circuit voltage [V]
Rbatt=0.044;            %Internal battery resistance [Ohm]
me=2063*1.08;           %EV mass (inertial losses) [kg]
Cnom=1000;              %J/kg*K
m_batt=497;             %EV battery mass [kg]
T_amb=25;               %ambient temp [ºC]


%EQN
Q_amb=2.38*(v^0.89)*(T-T_amb)*2;
T_dt=(1/(m_batt*Cnom))*((((Fa_me*me*v)/Voc)^2+((Fb_me*me*regen_eff*v)/Voc)^2)*Rbatt-Q_dt*Q_eff-Q_amb);
Amb_Heat=Q_amb;
Heat=Q_dt*Q_eff;
```

Figure 9 Simulink, General dynamics function T

3.2.3 *Model testing*

To test the model with the thermal dynamics of the battery added, an experiment similar to the one in the previous section has been carried out. During the first 1000 seconds a constant torque of 150 N*m has been applied and then braking has been applied until the vehicle comes to a complete stop.

The objective of this experiment is to see the behavior of the temperature under acceleration and regenerative braking. As well as that, the objective was also to observe the heat transfer at high velocities to check if the chosen parameters are reasonable.

Figure 10 shows the evolution of the ambient heat dissipated throughout the velocity profile.

Figure 11 shows the evolution of the temperature throughout acceleration and braking.

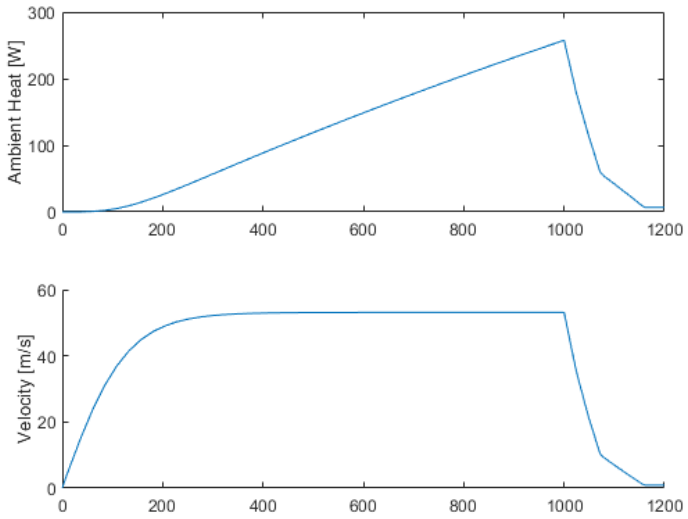

Figure 10 Testing, Thermal dynamics Velocity [m/s] and Ambient heat [W]

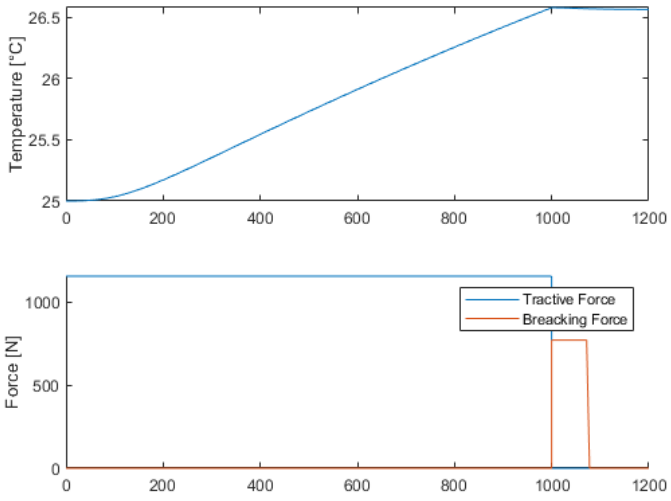

Figure 11 Testing, Thermal dynamics Temperature [ºC] and Force [N]

In view of the previous figures, it seems that the results obtained are reasonable. The temperature rises considerably with constant high torque, and while under braking, the temperature still rises but to a lesser extent, because there is still electrical current flow due to regenerative braking.

In turn, the heat exchanged with the environment reaches reasonable values at high temperatures and decreases as the velocity decreases, which is expected.

At the end of this test the state of charge of the battery dropped to $SOC(t = 1200s) = 72.57\%$

Next, a second test has been carried out to see the correct operation of the equations by varying the added control variable (forced heat dissipation).

In order to do so, an experiment similar to the previous ones has been carried out, accelerating with a constant torque until the velocity stabilizes and then applying a braking torque until the vehicle comes to a complete stop. However, the control variable P_{temp} is activated at 500s with a value of 1000W (electrical).

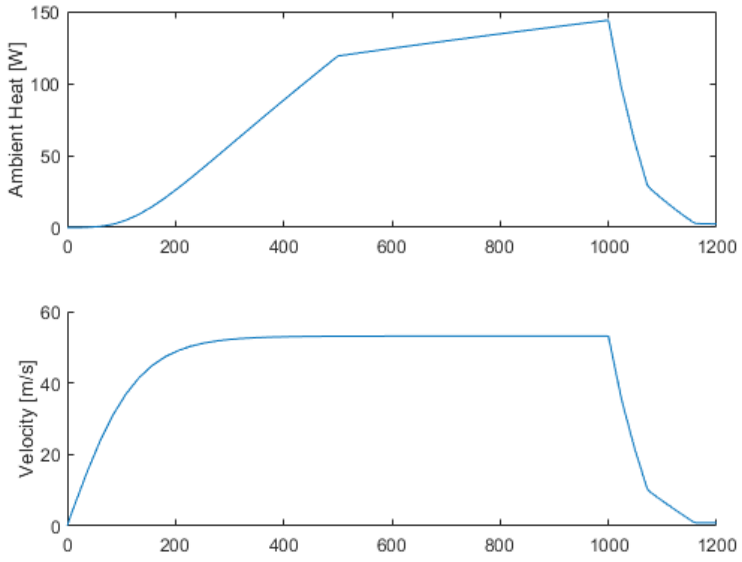

Figure 12 Testing, Thermal dynamics Ambient heat [W]t and Velocity [m/s]

Figure 12 shows the heat dissipated with the environment. It can be seen that from *t=500s* the heat transfer increases at a lower rate. This is because some of the heat is now dissipated through forced transfer.

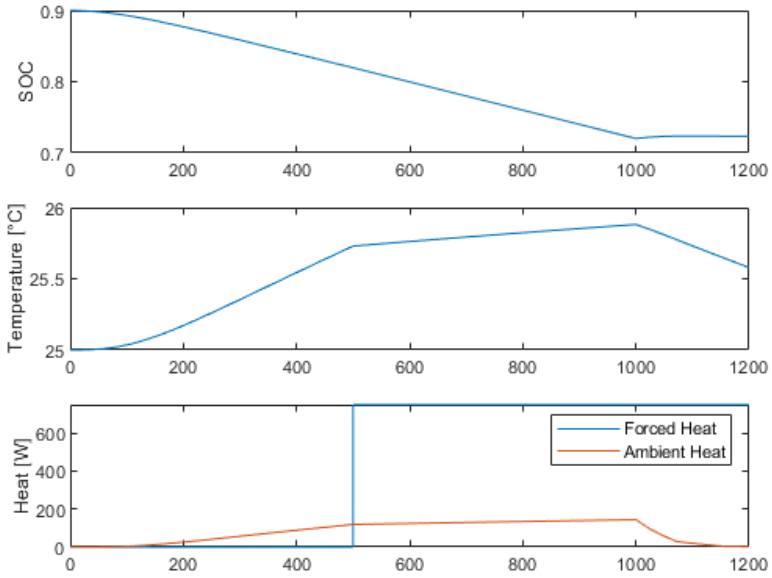

Figure 13 Testing, Thermal dynamics SoC, Temperature [ºC] and Heat [W]

In Figure 13 it can be seen how at *t=500s* the rate of increase of the temperature also decreases because there is more heat transfer and from *t=1000s* (when it stops accelerating) the temperature decreases because it is forcing heat dissipation.

The state of charge of the battery at the end of the test is: $SOC(t = 1200s) = 72.33\%$, which is lower than in the previous test that was not forcing any heat dissipation, which is what it was expected.

In conclusion, the created model provides reasonable values, and its behavior corresponds to the expected one.

3.3 Internal battery resistance model

3.3.1 Internal battery resistance equations (regression)

The objective of this new model is to add more coupling between the two state variables temperature and state of charge.

To do this, based on a Siemens article, in which a Tesla model 3 is modeled to climb Pikes Peak Hill, the internal resistance of the battery is modeled based on the state of charge and temperature.

Figure 14 shows the relationship described above. Taking values from the graph and entering them in an excel sheet, the relationship between the state variables and the internal resistance can be obtained.

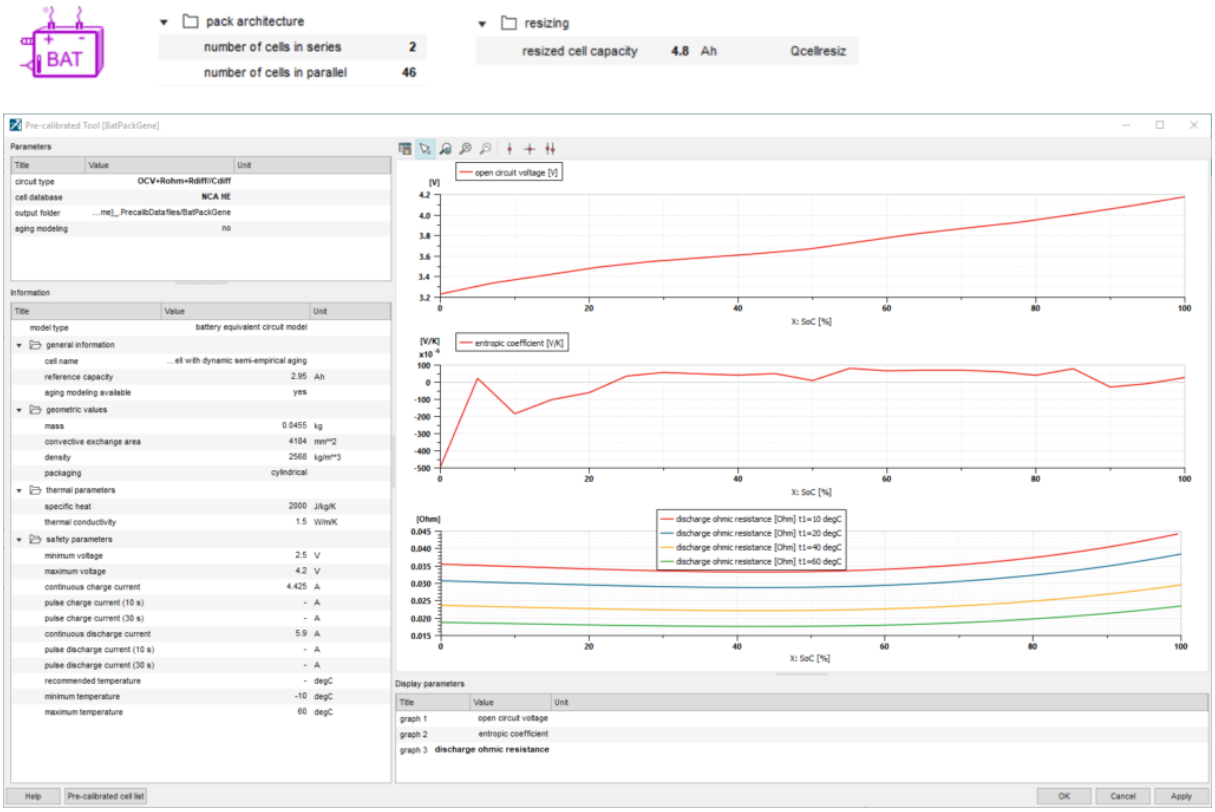

Figure 14 Siemens article: Internal Resistance of Battery

To carry out the regression, in addition to taking values, it has been decided to see if there is a quadratic relationship.

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	4	1269.582	317.3955	316.163	4.19778E-17
Residual	19	19.07407	1.003898		
Total	23	1288.656			

Table 1 ANOVA regression

This is the result obtained:

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	41.97782	0.88638	47.35872	3.46E-21	40.12261	43.83304	40.12261	43.83304
SOC	-0.16152	0.021354	-7.56367	3.82E-07	-0.20621	-0.11682	-0.20621	-0.11682
SOC2	0.00221	0.000205	10.78079	1.55E-09	0.001781	0.002639	0.001781	0.002639
T	-0.57257	0.056403	-10.1514	4.13E-09	-0.69062	-0.45452	-0.69062	-0.45452
T2	0.003275	0.000787	4.158615	0.000533	0.001627	0.004923	0.001627	0.004923

Table 2 Regression results

RESIDUAL OUTPUT

Observation	*Predicted R*	*Residuals*
1	36.57958	-0.57958
2	31.83628	-0.83628
3	24.3145	-0.3145
4	19.41249	-0.41249
5	34.23315	0.766849
6	29.48985	0.510149
7	21.96808	1.031925
8	17.06607	0.933935
9	33.65458	-0.65458
10	28.91128	0.08872
11	21.3895	0.610496
12	16.48749	1.012506
13	34.84387	-0.84387
14	30.10057	-0.60057
15	22.57879	-0.07879
16	17.67678	0.32322
17	37.80101	-0.80101

18	33.05771	-1.05771
19	25.53593	-0.53593
20	20.63392	-0.63392
21	42.52601	2.473992
22	37.78271	1.217292
23	30.26093	-0.26093
24	25.35892	-1.35892

Table 3 Residual output

"SOC 2" and "T2" are the squared variables to see if the squared relationship is relevant.

As can be seen in Table 2, both linear and quadratic variables are relevant ($p - value < \epsilon = 0.001$).

The crossover term SOC*T was calculated not to be relevant ($p - value > \epsilon$), so it was not included in the regression.

The equation that defines the internal resistance of the battery from the state of charge and the temperature of the battery is the following:

$$R_{batt} = \frac{41.978 - 0.162 * SOC + 0.002 * SOC^2 - 0.573 * T + 0.0033 * T^2}{1000} \quad (13)$$

To validate the resulting equation, the following figure shows the values of the internal resistance of the battery as a function of temperature and state of charge, in a similar way to that shown in the reference Figure 14.

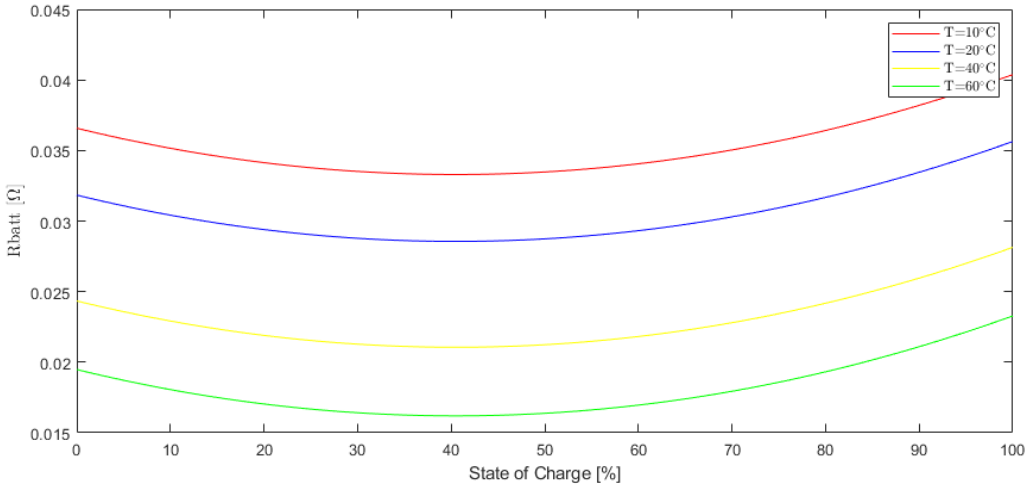

Figure 15 Validation, Regression for Internal Resistance of battery

Also, to add coupling between the SOC and temperature state variables, the battery capacity now depends on the temperature using the following relationship:

$$C_{batt} = C_{batt,nom} * (1 + 0.01 * (T - T_{ref}))$$
$$where\ T_{ref}\ is\ the\ reference\ temperature\ for$$
$$C_{batt,nom}, which\ is\ the\ nominal\ battery\ capacity \quad (14)$$

In this case, the values for $T_{ref}\ and\ C_{batt,nom}$ are: (reference)

- $C_{batt,nom} = 738000\ A * s$
- $T_{ref} = 20\ °C$

Finally, a new equation for the derivative of the state of charge is obtained:

$$\frac{dSOC}{dt} = -\frac{V_{oc} - \sqrt{{V_{oc}}^2 - 4 * R_{batt}(SOC, T) * (P_{acc} - P_{reg} - P_{temp})}}{2 * R_{batt}(SOC, T) * C_{batt} * (1 + 0.01 * (T - T_{ref}))} \quad (15)$$

3.3.2 Simulink Model

As in section 3.2.2, the model shares several blocks with the initial model. However, in this case a new "Matlab function" block (Figure 18) has been added in which the internal resistance of the battery is defined as a function of the temperature and state of charge (explained in the previous sections).

Also, as shown in Figure 16 a new gain block has been added, since the state of charge varies between 0 and 1 and the regression was performed for values between 0 and 100.

Finally, the "Matlab function" (Figure 17) block of $\frac{SOC}{dt}$ has been modified so that it follows the equations developed in the previous section.

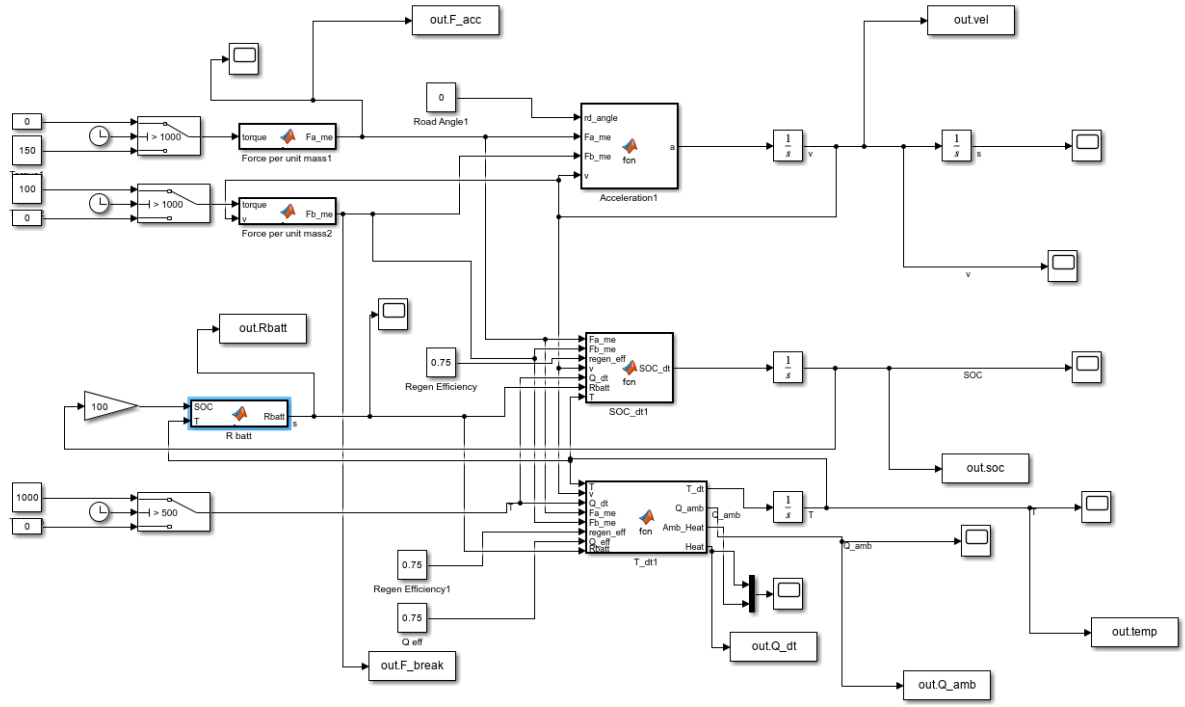

Figure 16 Simulink, Thermal + Rbatt dynamics model

```
function SOC_dt  = fcn(Fa_me,Fb_me,regen_eff,v,Q_dt,Rbatt,T)
%Params
Voc=400;            %Open-circuit voltage [V]
%Rbatt=0.044;       %Internal battery resistance [Ohm]
C_bat=205*3600;     %Battery capacity [A*s]
me=2063;            %EV mass (inertial losses) [kg]
T_ref=20;           %Ref temp for battery capacity [ºC]

%EQN
SOC_dt=-(Voc-sqrt(Voc^2-4*Rbatt*(Fa_me*v*me-Fb_me*v*me*regen_eff+Q_dt)))/(2*Rbatt*C_bat*(1+0.01*(T-T_ref)));

```

Figure 17 Simulink, Thermal + Rbatt dynamics function SOC

```
function Rbatt  = fcn(SOC,T)

Rbatt=(41.978-0.162*SOC+0.002*SOC^2-0.573*T+0.0033*T^2)/1000;
```

Figure 18 Simulink, Thermal + Rbatt dynamics function Rbatt

3.3.3 Model Testing

To test this model, an experiment similar to those described in sections 3.1.3 and 3.2.3 has been carried out.

In this case it is more difficult to know the expected results after the cycle of acceleration to constant velocity and braking to a complete stop.

However, seeing the resistance of the battery throughout the experiment in Figure 19 (lower than in the previous cases) it can be expected that the temperature increase will be less than in the previous cases and consequently that the heat exchanged will also be less.

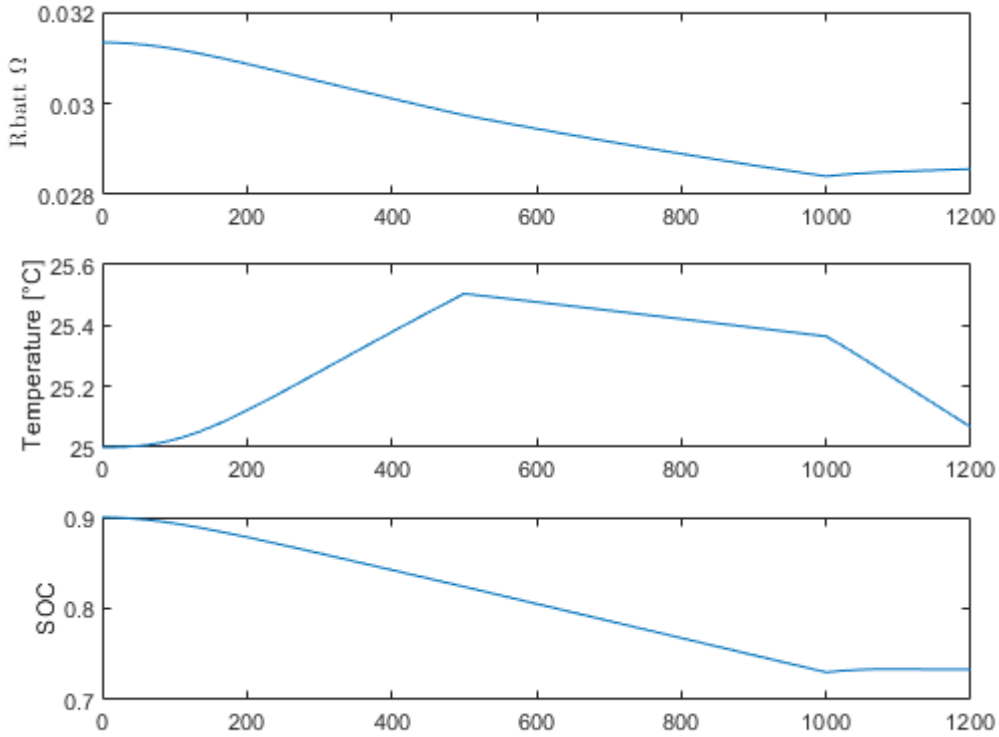

Figure 19 Testing, Thermal + Rbatt dynamics Rbatt(SOC,T)

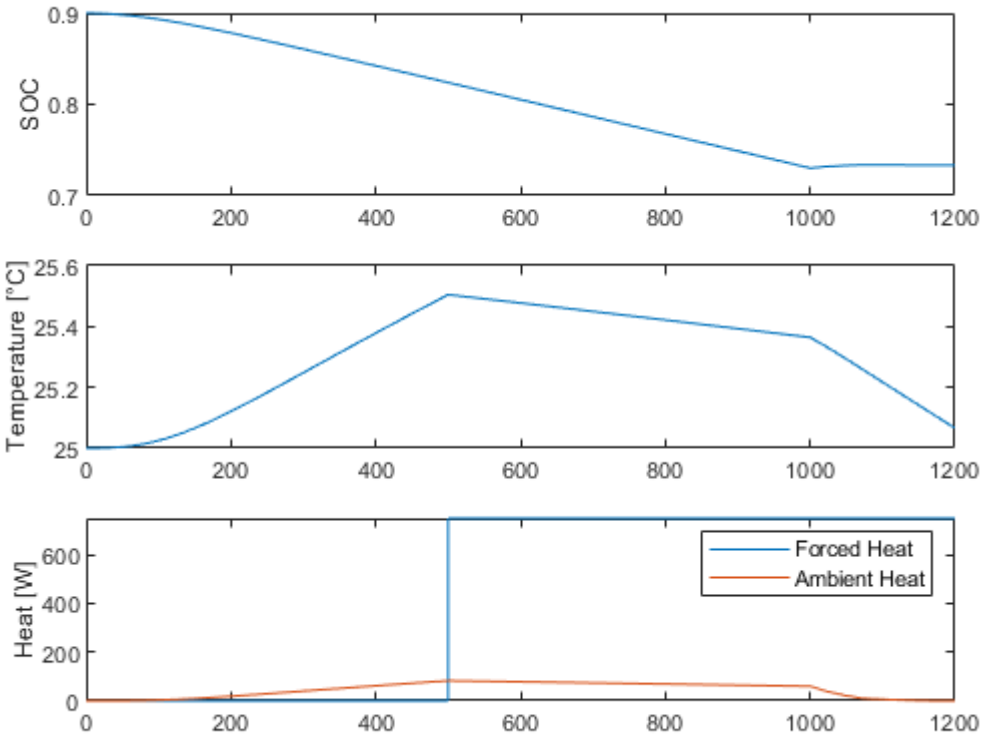

Figure 20 Testing, Thermal + Rbatt dynamics SOC, Temp and Heat

As can be seen in figure 19, the temperature variation is less than in the previous cases and consequently the heat dissipated to the environment is also reduced. Regarding the state of charge, it is similar to the previous cases.

It should be noted that this experiment is the same as the one carried out in section 3.2.4, second case.

3.4 Data-driven linear model using SINDyC

3.4.1 Introduction (SINDy)

Sparse Identification Non-Linear Dynamics is a technique for identifying dynamic systems from data.

The system has the following form:

$$\frac{d}{dt}x = f(x) \tag{16}$$

The possible non-linear candidates for the system are:

$$\Theta(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \dots \\ \sin(x) \\ \sin(2x) \\ \dots \end{bmatrix} \tag{17}$$

$$\dot{X} = \Xi * \Theta^T(x) \tag{18}$$

Ξ a matrix of coefficients (sparse) → employ sparse regression to calculate it.

Can also use the LASSO algorithm to find Ξ:

$$\xi_k = argmin_{\xi_k} \|\dot{x_k} - \xi_k * \Theta^T(X)\|_2 + \alpha * \|\xi_k\|_1 \tag{19}$$

The parameter α is selected to identify the Pareto optimal model that best balances low model complexity with accuracy.

3.4.2 Theory behind SINDyC

In this context, the SINDY approach is extended to incorporate inputs and control. Specifically, it examines a nonlinear dynamical system with inputs *u*:

$$\frac{d}{dt}x = f(x, u) \tag{20}$$

The SINDY algorithm can be easily adapted to incorporate actuation by expanding the library $\Theta(x, u)$ of potential functions to include *u*. This expansion can consist of non-linear cross-terms in both x and u. However, implementing this requires measuring both the state x and input signal *u*.

The sparse coefficients Ξ can be determined by solving the following equation if the signal *u* is associated with an external forcing:

$$\dot{x} = \Xi * \Theta^T(x, \Upsilon) \tag{21}$$

In the case where the signal *u* is a feedback control signal, represented by u = k(x), it becomes impossible to distinguish the impact of the internal feedback terms k(x) from the feedback control u within the dynamical system. Consequently, the SINDY regression becomes ill-conditioned. However, in such cases, the actuation *u* can be determined as a function of the state:

$$\Upsilon = \Xi_u * \Theta^T(x) \tag{22}$$

To determine the coefficients in Equation (21), there is a need to differentiate the signal *u* from *k(x)* terms. This can be achieved by introducing a white noise signal of adequate magnitude or by intermittently applying a significant impulse or step in *u*. An exciting prospect for future research would be to create input signals that assist in identifying the dynamical system in Equation (20) by perturbing the system in ways that produce information of higher value.

HOW THE PROGRAM WORKS

- Data Generation (if necessary):

First, from the non-linear dynamic model, the code generates data for the states from the input "*u*".

It is necessary to define: initial conditions, number of variables, and timestep.

For the model, it also requires setting up the *ode45* function.

- Splitting the data into training and validation:

After the data has been generated, it must be split into two sets.

The first one is used to train the SYNDYc algorithm, and the second one is to validate the model obtained.

Usually, the dataset is split in half.

- User parameters:

The user defines the parameters to specify how SYNDYc is going to be computed:

Polyorder: specifies the order of the system that is going to be computed. This includes cross-terms, e.g. if polyorder=3 there will be 1,x,x^2,x^3… but also x*y,x*y^2,…,u^2*x.

Usesine: 0 or 1. Determines if there will be sin(x), sin(2*x)… .

Lambda_vec & eps: optimized for balancing low model complexity with accuracy.

- Training the model:

First, the derivative of the states generated is computed (using fourth-order central difference).

Then, the program uses Sparse Regression to find (matrix of coefficients).

- Validating the model:

First, the algorithm makes predictions from the trained model. It uses the Runge-Kutta scheme of order 4 for the control system “rk4u(v,X,U,h,n)” and performs n steps of the scheme for the vector field v using step size h on each row of the matrix X.

Finally, it plots the following:

- Input Signal *‘u’* over time.
- State(s) ‘*x*’ over time.
- Validation plots the state(s) over time and also the state(s) generated by the trained model.

3.4.3 Example: Linear dynamic model

To see the potential of this tool, it has been decided to carry out an experiment to test this tool.

For this, the model created in section 3.1 will be used, which takes into account the longitudinal dynamics of the car and the dynamics of the state of charge of the battery.

The objective is to obtain a precise linear system so that in future control sections, the computational cost is much lower. By linearizing the dynamics, the constraints are no longer non-linear, so the computational cost of the optimizer is drastically reduced.

In this case there is only one control variable, which is acceleration (the braking control variable has been eliminated) to simplify the program.

The control action used to train the model is as follows:

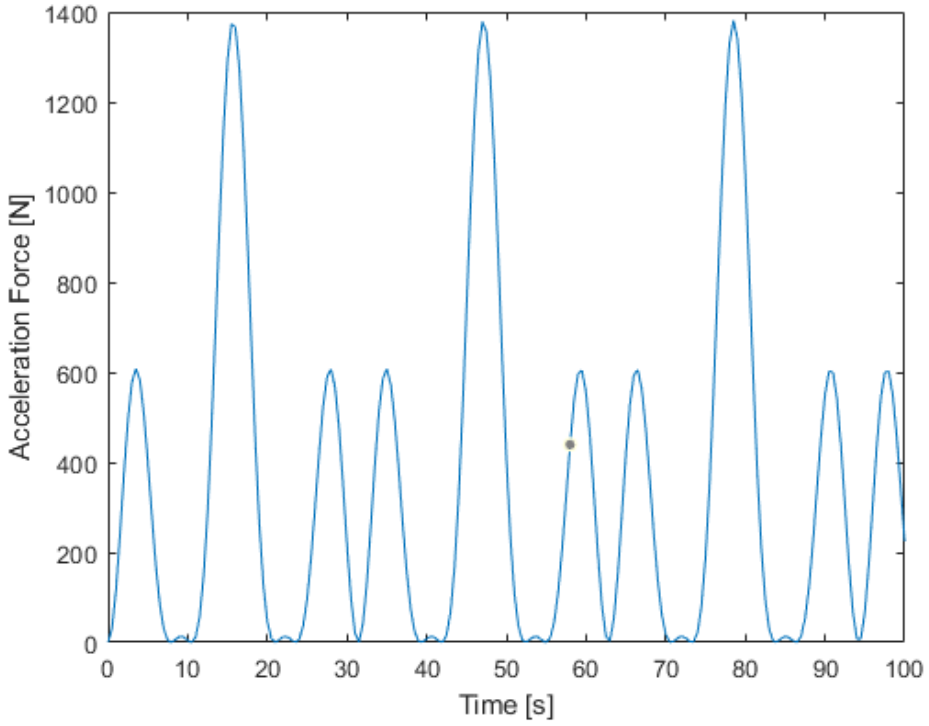

Figure 21 SINDyC, Input control Variable

Once the model has been trained and performs the sparse regression, the following linear model is obtained:

```
{0×0 char}    {'sdot'       }    {'vdot'       }    {'SOCdot'     }    {'udot'       }
{'1'     }    {[    11.7534]}    {[   -6.6601]}    {[    0.0036]}    {[    11.3429]}
{'s'     }    {[-3.1561e-05]}    {[1.9884e-05]}    {[         0]}    {[-2.8920e-05]}
{'v'     }    {[    0.9954]}    {[   -0.0084]}    {[-2.3789e-04]}    {[    -0.0047]}
{'SOC'   }    {[    -0.1539]}    {[    0.0885]}    {[ 6.1996e-05]}    {[    -0.1486]}
{'u'     }    {[    -0.0010]}    {[    0.9976]}    {[    -0.0258]}    {[ 9.2000e-05]}
```

Figure 22 SINDyC, linearized model

In order to validate the linear model, the result obtained with the approximate linear model is superimposed with the real data:

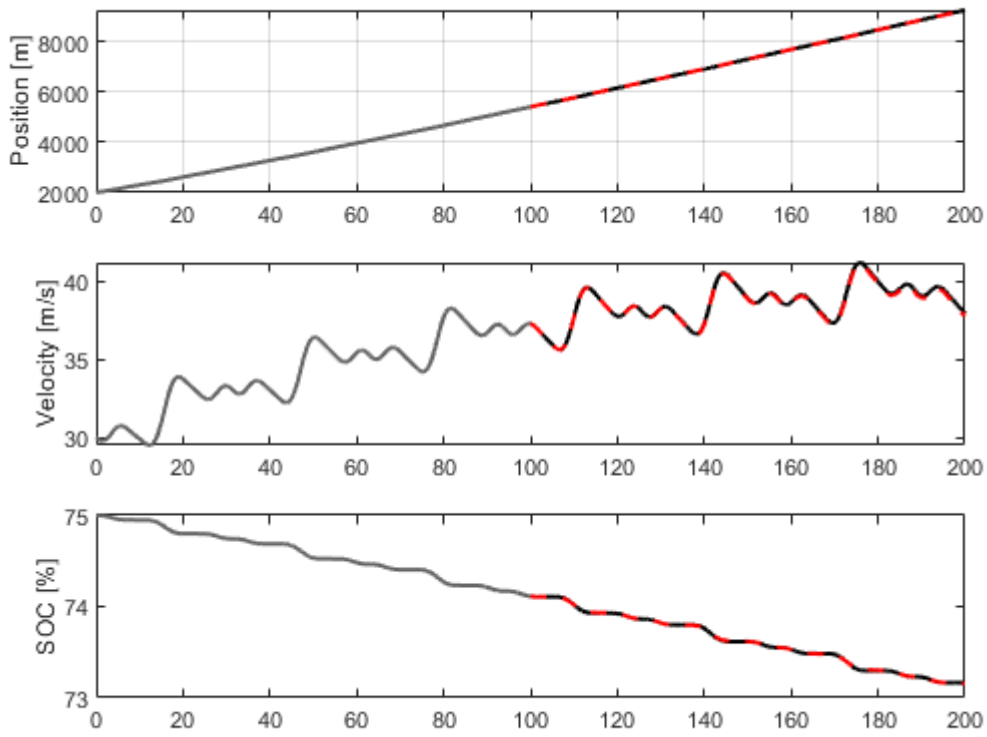

Figure 23 SINDyC, Training and Validation Results

As can be seen in Figure 23, the linear model obtained fits the real results almost perfectly.

In future projects, this type of tool could be used to reduce the computational cost when analyzing more complex models.

4 Energy Management Strategy

4.1 Optimal Control Problem

The IPOPT software library aims to solve the following non-linear power management problem:

The continuous problem equation is:

$$\min \int_{t_0}^{t_f} [-W_1 * SOC\big(u(t)\big) + W_2 * F_{acc}(t) * v(t) + W_3 * F_{acc}(t)^2 + W_3 * F_{acc}(t)^2]dt \tag{23}$$

Where $W_1,\ W_2, W_3\ and\ W_4$ are weights used to give more importance to some terms or others.

$$subject\ to\ \dot{x} = f(x(t), u(t)) \tag{24}$$

Here $f(x(t), u(t))$ are the equations explained in the previous sections.

$$\begin{gathered} And\ also\ to\ the\ following\ constraints: \\ s_{profile}\big(t_f\big) - \epsilon_s \leq s\big(t_f\big) \leq s_{profile}\big(t_f\big) + \epsilon_s \\ LB_{v,profile} \leq v(t) \leq UB_{v,profile} \\ SOC(t) \geq 30\ \% \\ 10 \leq T(t) \leq 40\ [°C] \\ 0 \leq v(t) \leq 50\ \left[\frac{m}{s}\right] \\ 0 \leq F_{acc}(t), F_{break}(t) \leq 15000\ [N] \\ 0 \leq Q_{heat}(t) \leq 1000\ [W] \\ F_{acc}(t) * F_{break}(t) = 0 \end{gathered} \tag{25}$$

The $v_{profile}$ is given by a driving cycle called FTP-75, this cycle provides a reference velocity. “The EPA Federal Test Procedure, commonly known as FTP-75 for the city driving cycle, are a series of tests defined by the US Environmental Protection Agency (EPA) to measure tailpipe emissions and fuel economy of passenger cars (excluding light trucks and heavy-duty vehicles).” (Wikipedia, 2023)

From the results following the FTP75 as a reference, it is possible to compare the efficiency of different controls.

The lower and upper bounds of the velocity ($LB_{v,profile}\ and\ UB_{v,profile}$) are calculated from the FTP75. An acceptable differential is added or subtracted from the driving cycle $\epsilon_v = 1.5\frac{m}{s}$.

In the case of the position, an acceptable range is $\epsilon_s = 2m$.

The last constraint has the objective of preventing the vehicle from accelerating and braking at the same time.

However, the software library solves the following discrete problem:

$$\min \sum_{t_0}^{t_f} [-W_1 * SOC(k) + W_2 * F_{acc}(k) * v(k) + W_3 * F_{acc}(k)^2 + W_3 * F_{acc}(k)^2] * \Delta t \tag{26}$$

$$subject\ to\ x(k+1) = f(x(k), u(k)) \tag{27}$$

$$\begin{gathered} And\ also\ to\ the\ following\ constraints: \\ s_{profile}(k_f) - \epsilon_s \leq s(k_f) \leq s_{profile}(k_f) + \epsilon_s \\ LB_{v,profile}(k) \leq v(k) \leq UB_{v,profile}(k) \\ SOC(k) \geq 30\ \% \\ 10 \leq T(k) \leq 40\ [°C] \\ 0 \leq v(k) \leq 50\ \left[\frac{m}{s}\right] \\ 0 \leq F_{acc}(k), F_{break}(k) \leq 15000\ [N] \\ 0 \leq Q_{heat}(k) \leq 1000\ [W] \\ F_{acc}(k) * F_{break}(k) = 0 \end{gathered} \tag{28}$$

4.2 Tool used: IPOPT

The tool used to solve the problem is IPOPT (Interior Point OPTimizer). It is a software library that solves large-scale nonlinear optimization problems in continuous problems. This software library is managed by Julia.

The chosen optimizer is suitable for the following reasons:

- Constraints and nonlinear objective: The problem to be solved includes nonlinear constraints. For example, equality restrictions to follow the dynamics exposed in the previous sections. Furthermore, the objective function to be minimized is also nonlinear.
- Variable limits: Depending on the system model that is going to be optimized in the problem, there will be a significant number of variables that must be limited, and IPOPT is capable of considering various limits even for the same variable.
- Handling large-scale nonlinear problems: IPOPT is capable of efficiently solving large-scale problems. For example, in the problem to be solved, depending on the chosen horizon, the

number of variables can even exceed a thousand. IPOPT takes advantage of problem sparsity to optimize the problem efficiently.

- Interior-Point Method: IPOPT uses the interior-point optimization method. This method is effective for problems with nonlinear constraints, such as the one to be solved.

4.3 Hierarchical control

Here's an overview of the concepts behind this control strategy:

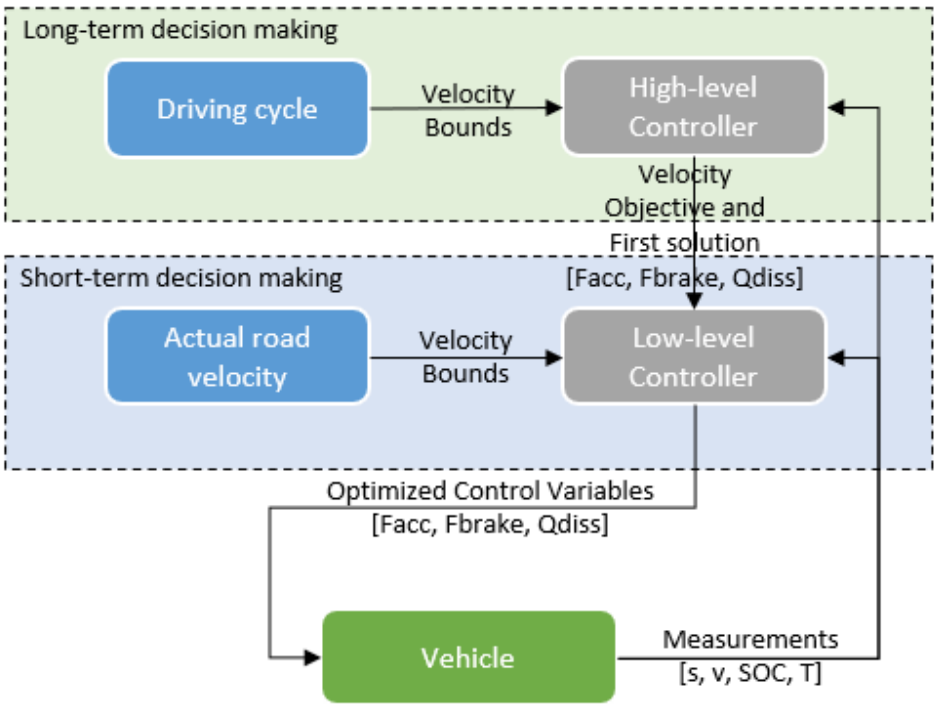

Figure 24 Overview Hierarchical Control Strat (own elaboration)

The driving cycle (average road velocity) provides the High-level Controller with velocity bounds (constraints). The High-level Controller calculates the optimal velocity profile for the entire trip from the driving cycle and the initial vehicle measurements.

This first solution is sent to the Low-level Controller. The Low-level Controller from this first solution and the optimal velocity calculated by the High-level Controller, the current road velocity, and the current vehicle measurements, calculates the optimal control variables for a shorter time horizon than the travel time. This optimal solution is close to the first solution calculated by the High-level Controller but also respects the actual velocity constraints of the road.

4.3.1 High-Level Controller

This section aims to explain the operation of the code developed in chapter 5 from Figure 24.

The first thing the high-level controller does is get the driving cycle. This is the average velocity of the roads. In the case of this project, the FTP75 will be taken as the driving cycle.

Once the driving cycle information (reference velocity) has been extracted, the functions are defined. These functions are the same as those defined in the models in the previous sections.

Next, the initial conditions are defined, and the constraints are defined.

Then, the code proceeds to solve the system defined by the variables to be optimized and the constraints.

Finally, the code stores both the state variables and the optimized control variables in a .csv file. This file will later be read by the low-level controller.

4.3.2 Low-Level Controller

The low-level controller works in much the same way as the high-level controller with a few differences.

The first thing the code does is read the actual road velocity. In this case, FTP75 is used again. In other cases, the real-time velocity of the road could be used in the case of connected vehicles. Next, it reads the solution provided by the high-level controller. Then, it defines the functions of the system and initializes the solution vectors.

Next, a “for loop” begins, which has the objective of moving the computation horizon of the optimizer. What the code intends is to imitate what would happen if this code were applied to vehicles connected in real-time.

Next, optimize the problem for that short time horizon. Once the time horizon has traveled the full path and the optimizer has calculated for each of these movements, the optimized state and control variables are stored in a .csv file.

5 Developed Code

From Figure 25 to Figure 37 the code developed for the implementation of the energy management strategy is shown.

5.1 High-level Controller for general dynamics model

```
using JuMP
using CSV
using Ipopt
using DataFrames
using Plots
using XLSX


# Read .csv with FTP75 data

    # File path of the .csv
    file_path = "driving_cycle_data.csv"

    # Load data from the Excel file
    data = CSV.File(file_path,header=false)

    # Convert to a DataFrame
    data_df = DataFrame(data)

    # Display the DataFrame
    #println(data_df)

    # Access the data as needed
    Time = data_df[:,1]             # Load Time
    real_s_profile = data_df[:,2]   # Load Positon
    real_v_profile = data_df[:,3]   # Load Velocity
    lower_vel_bound= data_df[:,4]   # Load Velocity lower bound
    upper_vel_bound= data_df[:,5]   # Load Velocity upper bound

# Constants
Tamb = 25.0     # Ambient temperature [°C]
t_horz = 598    # Driving cycle time [s]

# FUNCTIONS: dynamics of the model
    # v_dot function
    function v_dot(v, F_acc, F_break)
        me = 2063 * 1.08  # kg inertial losses
        ro = 1.293
        A = 2.22
        Cd = 0.23
        # EQN
        v_dot = real( (1 / me) * (F_acc - F_break - (0.5 * ro * A * Cd * v^2)) )
        return v_dot
    end

```

Figure 25 Code, high-lvl controller general dyn 1

```julia
    # soc_dot function
    function soc_dot(v, F_acc, F_break)
        Voc = 400
        C_bat = 205 * 3600  # battery capacity A*s
        T_0 = 20  # ºC Ref temp for battery capacity
        regen_eff = 0.75
        Q_eff = 0.75
        Rbatt=0.04;
        # EQN
        soc_dot = 100 * real( -(Voc - sqrt(Voc^2 - 4 * Rbatt * (F_acc * v - F_break * v * regen_eff ))) / (2 * Rbatt * C_bat ) )
        return soc_dot
    end

# Create JuMP optimization model
model = Model(optimizer_with_attributes(Ipopt.Optimizer, "print_level" => 0))

#Register the functions
register(model, :v_dot, 3, v_dot; autodiff = true)
register(model, :soc_dot, 3, soc_dot; autodiff = true)

# Variables to be optimzed
@variable(model, xh[1:3, 1:t_horz+1])
@variable(model, uh[1:2, 1:t_horz])

# Set initial conditions for the first time step
initial_conditions = [0.0, 0.0, 90.0]  # s_0=0m v_0=0 m/s soc_0=90%

# Initial Conditions
for k in 1:3
    @constraint(model, xh[k, 1] == initial_conditions[k])
end

# Weights in the objective function
const_1 = 0.01
const_2 = 0.01
const_3 = 0.01

# Constraints
    for k in 1:t_horz

        @NLconstraint(model, xh[1, k+1] == xh[1, k] + xh[2, k] + 0.5 * v_dot(xh[2, k], uh[1, k], uh[2, k]))
        @NLconstraint(model, xh[2, k+1] == xh[2, k] + v_dot(xh[2, k], uh[1, k], uh[2, k]))
        @NLconstraint(model, xh[3, k+1] == xh[3, k] + soc_dot(xh[2, k], uh[1, k], uh[2, k]))
```

Figure 26 Code, high-lvl controller general dyn 2

```julia

        @constraint(model, lower_vel_bound[k] <= xh[2, k+1] <= upper_vel_bound[k])
        @constraint(model, 0 <= xh[2, k+1] <= 50)
        @constraint(model, 100 >= xh[3, k+1] >= 0)


        @constraint(model, 0 <= uh[1, k] <= 15000)
        @constraint(model, 0 <= uh[2, k] <= 15000)


        @NLconstraint(model, uh[1, k] * uh[2, k] == 0)


    end

    @constraint(model, real_s_profile[end]-2 <= xh[1, end] <= real_s_profile[end]+2)

# Objective function
@objective(model, Min, const_1  * sum(uh[1, k]*xh[2, k] for k in 1:t_horz) + const_2 * sum(uh[1, k]^2 for k in 1:t_horz) ...
...+ const_3 * sum(uh[2, k]^2 for k in 1:t_horz) - sum(xh[3, t_horz] for k in 1:t_horz))

# Solve the optimization problem
optimize!(model)

# Print solution
    #println("Objective value: ", objective_value(model))
    for i in 1:3
        for j in 1:t_horz
            #println("State variables xh: ", "value", i,",",j, "=", value(xh[i,j]))
        end
    end

    for i in 1:2
        for j in 1:t_horz
            #println("Control variables uh: ", "value", i,",",j, "=",  value.(uh[i,j]))
        end
    end

# Print solution in .csv
    # Transpose the result data
    position_data = value.(xh[1,:]);
    velocity_data = value.(xh[2,:]);
    soc_data = value.(xh[3,:]);
    f_acc_data = value.(uh[1,:]);
    f_brake_data = value.(uh[2,:]);
```

Figure 27 Code, high-lvl controller general dyn 3

```
134
135     # Write data to CSV file by columns
136     f_1 = open("high_lvl_solved_gen_dyn_xh.csv", "w");
137     for i in 1:length(position_data)
138         println(f_1, join([position_data[i], velocity_data[i], soc_data[i]], ","));
139     end
140     close(f_1);
141
142     # Write data to CSV file by columns
143     f_2 = open("high_lvl_solved_gen_dyn_uh.csv", "w");
144     for i in 1:length(f_acc_data)
145         println(f_2, join([f_acc_data[i], f_brake_data[i]], ","));
146     end
147     close(f_2);
148
```

Figure 28 Code, high-lvl controller general dyn 4

5.2 High-level Controller for thermal dynamics model

```
using JuMP
using CSV
using Ipopt
using DataFrames
using Plots
using XLSX


# Read .csv with FTP75 data

    # File path of the .csv
    file_path = "driving_cycle_data.csv"

    # Load data from the Excel file
    data = CSV.File(file_path,header=false)

    # Convert to a DataFrame
    data_df = DataFrame(data)

    # Display the DataFrame
    #println(data_df)

    # Access the data as needed
    Time = data_df[:,1]             # Load Time
    real_s_profile = data_df[:,2]   # Load Positon
    real_v_profile = data_df[:,3]   # Load Velocity
    lower_vel_bound= data_df[:,4]   # Load Velocity lower bound
    upper_vel_bound= data_df[:,5]   # Load Velocity upper bound

# Constants
Tamb = 25.0     # Ambient temperature [°C]
t_horz = 598    # Driving cycle time [s]

# FUNCTIONS: dynamics of the model
    # v_dot function
    function v_dot(v, F_acc, F_break)
        me = 2063 * 1.08  # kg inertial losses
        ro = 1.293
        A = 2.22
        Cd = 0.23
        # EQN
        v_dot = real( (1 / me) * (F_acc - F_break - (0.5 * ro * A * Cd * v^2)) )
        return v_dot
    end

```

Figure 29 Code, high-lvl controller thermal dyn 1

```
    # soc_dot function
    function soc_dot(v, F_acc, F_break,Q_dt)
        Voc = 400
        C_bat = 205 * 3600  # battery capacity A*s
        T_0 = 20  # ºC Ref temp for battery capacity
        regen_eff = 0.75
        Q_eff = 0.75
        Rbatt=0.04;
        # EQN
        soc_dot = 100 * real( -(Voc - sqrt(Voc^2 - 4 * Rbatt * (F_acc * v - F_break * v * regen_eff + Q_dt))) / (2 * Rbatt * C_bat ) )
        return soc_dot
    end

    # T_dot function
    function T_dot(v,T,F_acc,F_break,Q_dt)
        Voc=400;                    #Open-circuit voltage [V]
        Rbatt=0.04;                #Internal battery resistance [Ohm]
        me=2063*1.08;               #EV mass (inertial losses) [kg]
        Cnom=1000;                  #J/kg*K
        m_batt=497;                 #EV battery mass [kg]
        T_amb=25;                   #ambient temp [ºC]
        regen_eff = 0.75
        Q_eff = 0.75
        #EQN
        #Q_amb = 2.38*(v)*(T-T_amb)*2
        Q_amb = 2.38*(0.6336*v+1.3712)*(T-T_amb)*2
        T_dot = real( (1/(m_batt*Cnom))*((((F_acc*v)/Voc)^2+((F_break*regen_eff*v)/Voc)^2)*Rbatt-Q_dt*Q_eff-Q_amb));
        return T_dot
    end

# Create JuMP optimization model
model = Model(optimizer_with_attributes(Ipopt.Optimizer, "print_level" => 0))

#Register the functions
register(model, :v_dot, 3, v_dot; autodiff = true)
register(model, :soc_dot, 4, soc_dot; autodiff = true)
register(model, :T_dot, 5, T_dot; autodiff = true)


# Variables to be optimzed
@variable(model, xh[1:4, 1:t_horz+1])
@variable(model, uh[1:3, 1:t_horz])

# Set initial conditions for the first time step
initial_conditions = [0.0, 0.0, 90.0, 25.0]  # s_0=0m v_0=0 m/s soc_0=90% T_0=25.0 °C

```

Figure 30 Code, high-lvl controller thermal dyn 2

```
# Initial Conditions
for k in 1:4
    @constraint(model, xh[k, 1] == initial_conditions[k])
end

# Weights in the objective function
const_1 = 0.01
const_2 = 0.01
const_3 = 0.01
const_4 = 0.01


# Constraints
    for k in 1:t_horz

        @NLconstraint(model, xh[1, k+1] == xh[1, k] + xh[2, k] + 0.5 * v_dot(xh[2, k], uh[1, k], uh[2, k]))
        @NLconstraint(model, xh[2, k+1] == xh[2, k] + v_dot(xh[2, k], uh[1, k], uh[2, k]))
        @NLconstraint(model, xh[3, k+1] == xh[3, k] + soc_dot(xh[2, k], uh[1, k], uh[2, k], uh[3, k]))
        @NLconstraint(model, xh[4, k+1] == xh[4, k] + T_dot(xh[2, k], xh[4, k], uh[1, k], uh[2, k], uh[3, k]))



        @constraint(model, lower_vel_bound[k] <= xh[2, k+1] <= upper_vel_bound[k])
        @constraint(model, 0 <= xh[2, k+1] <= 50)
        @constraint(model, 100 >= xh[3, k+1] >= 0)
        @constraint(model, 40 >= xh[4, k+1] >= 10)


        @constraint(model, 0 <= uh[1, k] <= 15000)
        @constraint(model, 0 <= uh[2, k] <= 15000)
        @constraint(model, 0 <= uh[3, k] <= 1000)


        @NLconstraint(model, uh[1, k] * uh[2, k] == 0)

    end

    @constraint(model, real_s_profile[end]-2 <= xh[1, end] <= real_s_profile[end]+2)

# Objective function
@objective(model, Min, const_1  * sum(uh[1, k]*xh[2, k] for k in 1:t_horz) + const_2 * sum(uh[1, k]^2 for k in 1:t_horz)
+ const_3 * sum(uh[2, k]^2 for k in 1:t_horz) + const_4 * sum(uh[3, k]^2 for k in 1:t_horz) - sum(xh[3, t_horz] for k in 1:t_horz))

# Solve the optimization problem
optimize!(model)

```

Figure 31 Code, high-lvl controller thermal dyn 3

```
# Solve the optimization problem
optimize!(model)

# Print solution
    println("Objective value: ", objective_value(model))
    for i in 1:4
        for j in 1:t_horz
            println("State variables xh: ", "value", i,",",j, "=", value(xh[i,j]))
        end
    end

    for i in 1:3
        for j in 1:t_horz
            println("Control variables uh: ", "value", i,",",j, "=",  value.(uh[i,j]))
        end
    end

# Print solution in .csv
    # Transpose the result data
    position_data = value.(xh[1,:]);
    velocity_data = value.(xh[2,:]);
    soc_data = value.(xh[3,:]);
    temp_data = value.(xh[4,:]);
    f_acc_data = value.(uh[1,:]);
    f_brake_data = value.(uh[2,:]);
    Q_dt_data = value.(uh[3,:]);


    # Write data to CSV file by columns
    f_1 = open("high_lvl_solved_thm_dyn_xh.csv", "w");
    for i in 1:length(position_data)
        println(f_1, join([position_data[i], velocity_data[i], soc_data[i], temp_data[i]], ","));
    end
    close(f_1);


    # Write data to CSV file by columns
    f_2 = open("high_lvl_solved_thm_dyn_uh.csv", "w");
    for i in 1:length(f_acc_data)
        println(f_2, join([f_acc_data[i], f_brake_data[i], Q_dt_data[i]], ","));
    end
    close(f_2);
```

Figure 32 Code, high-lvl controller thermal dyn 4

5.3 Low-level Controller for general dynamics model

```
using JuMP
using CSV
using Ipopt
using DataFrames
using Plots
using XLSX


# Constants
Tamb = 25.0         # Ambient temperature [°C]
t_horz = 20        # Low-lvl controller horizon [s]
t_horz_high = 50    # Driving cycle time [s]


# Read .csv with FTP75 data

    # File path of the .csv
    file_path = "driving_cycle_data.csv"

    # Load data from the Excel file
    data = CSV.File(file_path,header=false)

    # Convert to a DataFrame
    data_df = DataFrame(data)

    # Display the DataFrame
    #println(data_df)

    # Access the data as needed
    Time = data_df[:,1]             # Load Time
    real_s_profile = data_df[:,2]   # Load Positon
    real_v_profile = data_df[:,3]   # Load Velocity
    lower_vel_bound= data_df[:,4]   # Load Velocity lower bound
    upper_vel_bound= data_df[:,5]   # Load Velocity upper bound
# Read .csv with first solution from high level controller data

    # File path of the .csv
    file_path_2 = "high_lvl_solved_gen_dyn_xh.csv"

    # Load data from the Excel file
    data_2 = CSV.File(file_path_2,header=false)

    # Convert to a DataFrame
    data_df_2 = DataFrame(data_2)

    # Display the DataFrame
    #println(data_df_2)
```

Figure 33 Code, low-lvl controller general dyn 1

```
    #println(data_df_2)

    # Access the data as needed
    opt_pos = data_df_2[:,1]    # Load Positon
    opt_vel = data_df_2[:,2]    # Load Velocity
    opt_soc = data_df_2[:,3]     # Load SOC

# Read .csv with first solution from high level controller data

    # File path of the .csv
    file_path_3 = "high_lvl_solved_gen_dyn_uh.csv"

    # Load data from the Excel file
    data_3 = CSV.File(file_path_3,header=false)

    # Convert to a DataFrame
    data_df_3 = DataFrame(data_3)

    # Display the DataFrame
    #println(data_df_2)

    # Access the data as needed
    opt_f_acc = data_df_2[:,1]    # Load F_acc
    opt_f_brake = data_df_2[:,2]    # Load F_brake


# FUNCTIONS: dynamics of the model
    # v_dot function
    function v_dot(v, F_acc, F_break)
        me = 2063 * 1.08  # kg inertial losses
        ro = 1.293
        A = 2.22
        Cd = 0.23
        # EQN
        v_dot = real( (1 / me) * (F_acc - F_break - (0.5 * ro * A * Cd * v^2)) )
        return v_dot
    end

    # soc_dot function
    function soc_dot(v, F_acc, F_break)
        Voc = 400
        C_bat = 205 * 3600  # battery capacity A*s
        T_0 = 20  # ºC Ref temp for battery capacity
        regen_eff = 0.75
        Q_eff = 0.75
        Rbatt=0.04;
        # EQN
        soc_dot = 100 * real( -(Voc - sqrt(Voc^2 - 4 * Rbatt * (F_acc * v - F_break * v * regen_eff ))) / (2 * Rbatt * C_bat ) )
```

Figure 34 Code, low-lvl controller general dyn 2

```
        soc_dot = 100 * real( -(Voc - sqrt(Voc^2 - 4 * Rbatt * (F_acc * v - F_break * v * regen_eff ))) / (2 * Rbatt * C_bat ) )
        return soc_dot
    end

# Create JuMP optimization model
model = Model(optimizer_with_attributes(Ipopt.Optimizer, "print_level" => 0))

# Register the functions
register(model, :v_dot, 3, v_dot; autodiff = true)
register(model, :soc_dot, 3, soc_dot; autodiff = true)

# Initialize solution vectors
optimal_low_pos = Vector{Float64}(undef, t_horz_high+t_horz)
optimal_low_vel = Vector{Float64}(undef, t_horz_high+t_horz)
optimal_low_soc = Vector{Float64}(undef, t_horz_high+t_horz)
optimal_low_f_acc = Vector{Float64}(undef, t_horz_high+t_horz)
optimal_low_f_brake = Vector{Float64}(undef, t_horz_high+t_horz)

# Initial condition of solution vectors
optimal_low_pos[1]=opt_pos[1]
optimal_low_vel[1]=opt_vel[1]
optimal_low_soc[1]=opt_soc[1]
optimal_low_f_acc[1]=opt_f_acc[1]
optimal_low_f_brake[1]=opt_f_brake[1]

# Moving horizon through the driving cycle
for j in 2:t_horz_high

    # Variables
    @variable(model, xh[1:3, 1:t_horz+1])
    @variable(model, uh[1:2, 1:t_horz])

    # Set initial conditions for the first time step
    initial_conditions = [optimal_low_pos[j-1], optimal_low_vel[j-1], optimal_low_soc[j-1]]  # Replace with your initial values


    # Initial Conditions
    for k in 1:3
        @constraint(model, xh[k, 1] == initial_conditions[k])
    end

    # Weights in the objective function
    const_1 = 0.01
    const_2 = 0.01
    const_3 = 0.01
    const_4 = 1
```

Figure 35 Code, low-lvl controller general dyn 3

```
    # Constraints
        for k in 1:t_horz

            @NLconstraint(model, xh[1, k+1] == xh[1, k] + xh[2, k] + 0.5 * v_dot(xh[2, k], uh[1, k], uh[2, k]))
            @NLconstraint(model, xh[2, k+1] == xh[2, k] + v_dot(xh[2, k], uh[1, k], uh[2, k]))
            @NLconstraint(model, xh[3, k+1] == xh[3, k] + soc_dot(xh[2, k], uh[1, k], uh[2, k]))



            @constraint(model, lower_vel_bound[j+k-1] <= xh[2, k+1] <= upper_vel_bound[j+k-1])
            @constraint(model, 0 <= xh[2, k+1] <= 50)
            @constraint(model, 100 >= xh[3, k+1] >= 0)


            @constraint(model, 0 <= uh[1, k] <= 15000)
            @constraint(model, 0 <= uh[2, k] <= 15000)


            @NLconstraint(model, uh[1, k] * uh[2, k] == 0)



        end

        @constraint(model, real_s_profile[j+t_horz-1]-2 <= xh[1, end] <= real_s_profile[j+t_horz-1]+2)

    # Objective
    @objective(model, Min, const_4*sum((opt_vel[m]-xh[2, k])^2 for k in 1:t_horz for m in j:j+t_horz) ...
    ...+ const_1  * sum(uh[1, k]*xh[2, k] for k in 1:t_horz) + const_2 * sum(uh[1, k]^2 for k in 1:t_horz) + const_3 * sum(uh[2, k]^2 for k in 1:t_horz)...
    ... - sum(xh[3, t_horz] for k in 1:t_horz))

    # Solve the optimization problem
    optimize!(model)

    # Update and save the low-lvl solution vector
    optimal_low_pos[j]=value.(xh[1,2])
    optimal_low_vel[j]=value.(xh[2,2])
    optimal_low_soc[j]=value.(xh[3,2])
    optimal_low_f_acc[j]=value.(uh[1,2])
    optimal_low_f_brake[j]=value.(uh[2,2])

    # Start new optimization
    unregister(model, :xh)
    unregister(model, :uh)

    # Print progress
    println("Solved low control for t= ", j)
end
```

Figure 36 Code, low-lvl controller general dyn 4

```
# Print solution in .csv
    # Transpose the result data
    position_data = optimal_low_pos[:];
    velocity_data = optimal_low_vel[:];
    soc_data = optimal_low_soc[:];
    f_acc_data = optimal_low_f_acc[:];
    f_brake_data = optimal_low_f_brake[:];


    # Write data to CSV file by columns
    f_1 = open("low_lvl_solved_gen_dyn_xh.csv", "w");
    for i in 1:length(position_data)
        println(f_1, join([position_data[i], velocity_data[i], soc_data[i]], ","));
    end
    close(f_1);

    # Write data to CSV file by columns
    f_2 = open("low_lvl_solved_gen_dyn_uh.csv", "w");
    for i in 1:length(f_acc_data)
        println(f_2, join([f_acc_data[i], f_brake_data[i]], ","));
    end
    close(f_2);
```

Figure 37 Code, low-lvl controller general dyn 5

6 Results and Conclusion

6.1 Results

6.1.1 Energy Management Strategy: high-level controller

These results have been obtained from the following experiments:

The high-level controller has been tested with the FTP75 as driving cycle input.

In addition, the system model used in the functions is the general dynamics one.

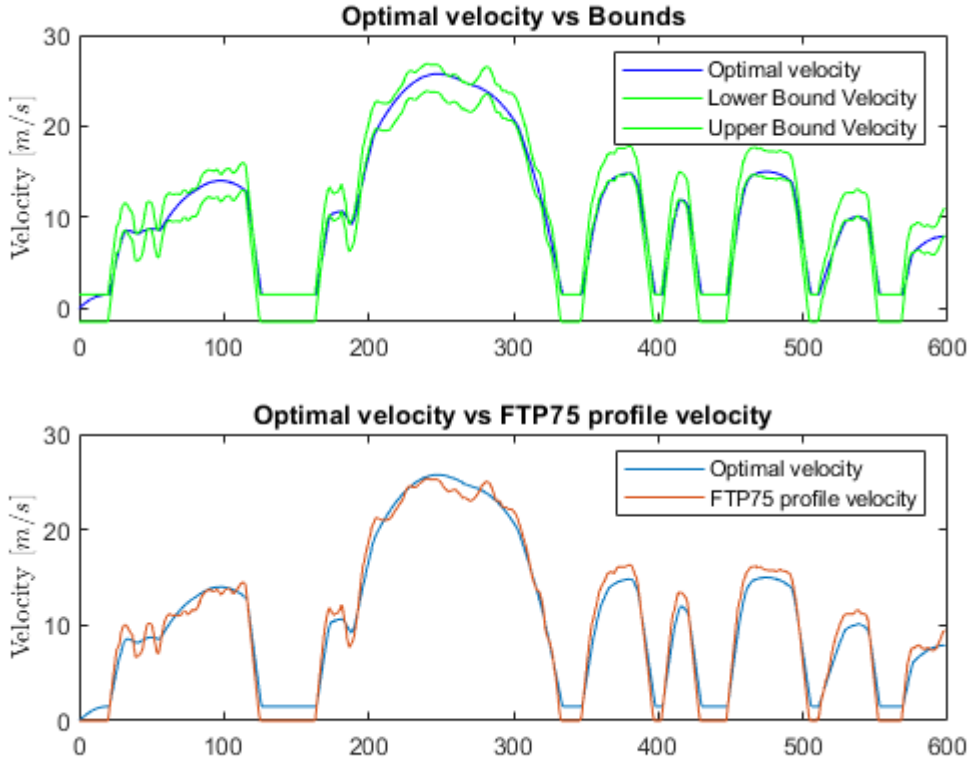

Figure 38 Results, high-lvl general dynamics 1

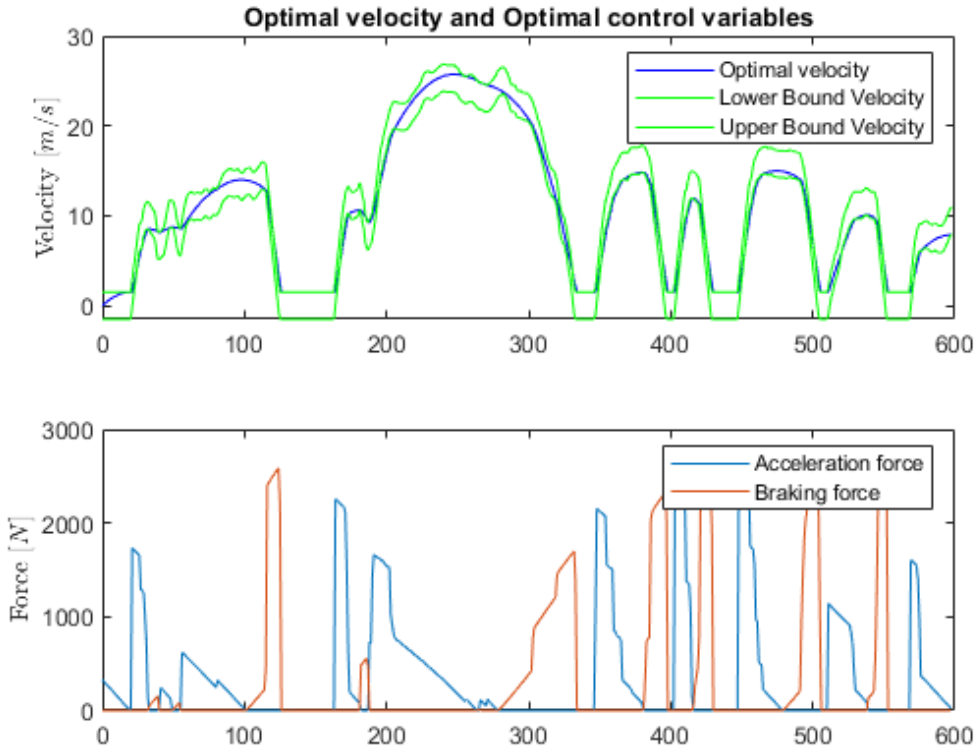

Figure 39 Results, high-lvl general dynamics 2

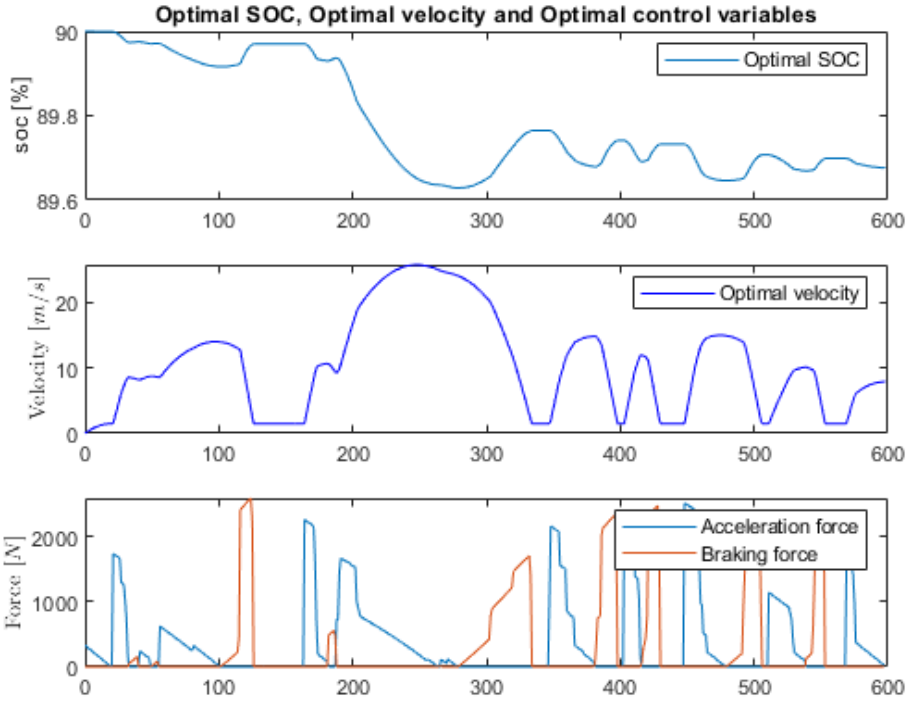

Figure 40 Results, high-lvl general dynamics 3

Figure 38, shows the optimized velocity profile along the path and is compared to the constraints. It also shows a comparison between the optimized velocity profile and the velocity profile of the FTP75.

Figure 39, like the previous figure, shows the optimized velocity profile, but this time the optimized control variables are shown.

Finally, Figure 40 shows the evolution of the state of charge throughout the experiment. The optimized velocity profile and the optimized control variables are also shown to be able to see relationships between them.

In view of these results, it can be said that they are reasonable, since, as it can be seen in the different figures, the optimizer provides a smoothed velocity profile while respecting the constraints.

This second experiment is very similar to the first. The only difference is that it has been applied to the model with thermal dynamics.

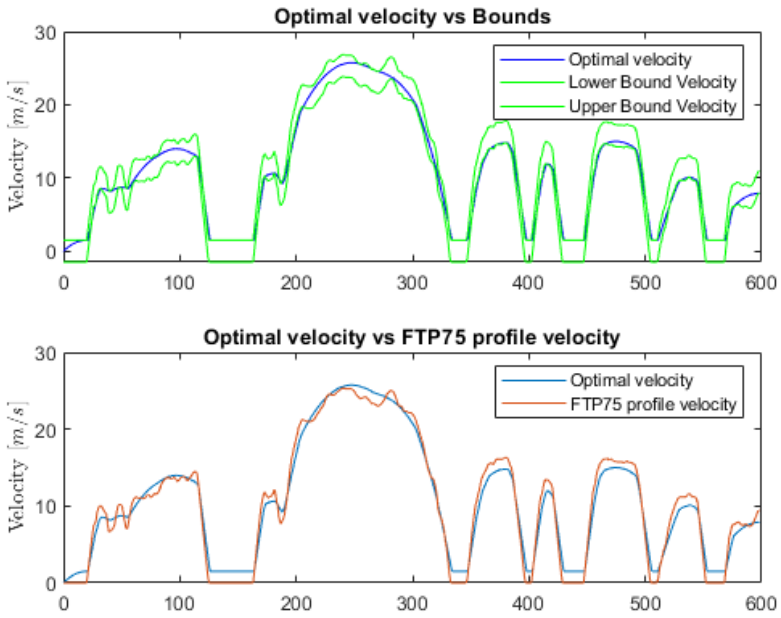

Figure 41 Results, high-lvl thermal dynamics 1

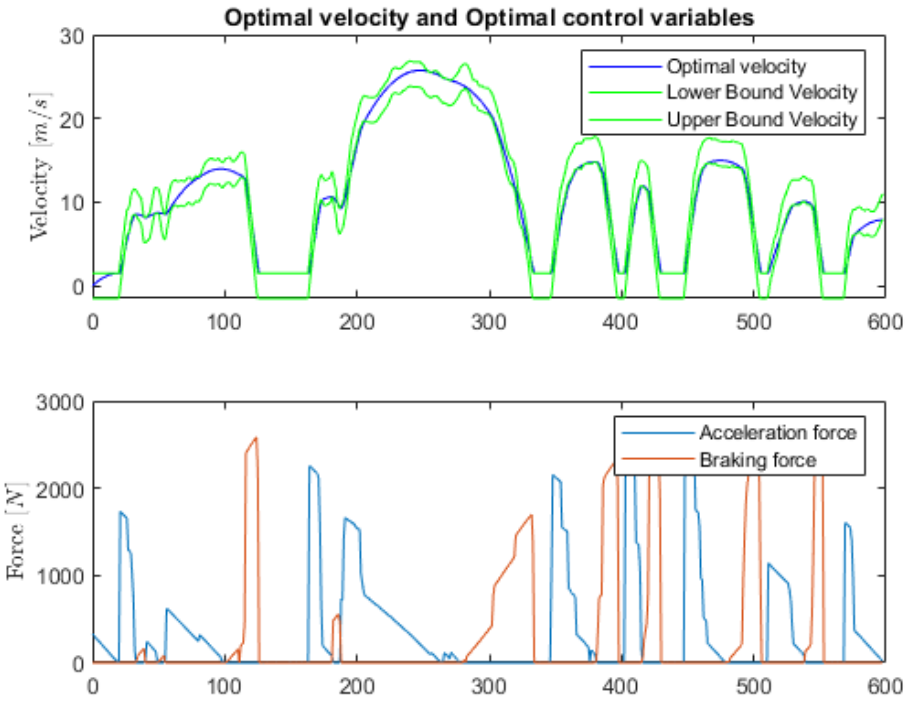

Figure 42 Results, high-lvl thermal dynamics 2

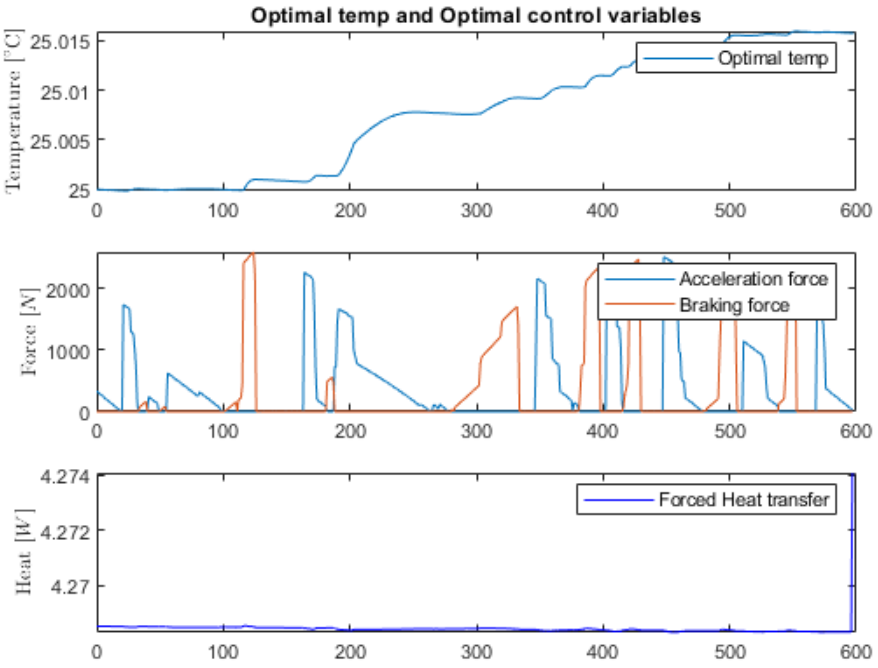

Figure 43 Results, high-lvl thermal dynamics 3

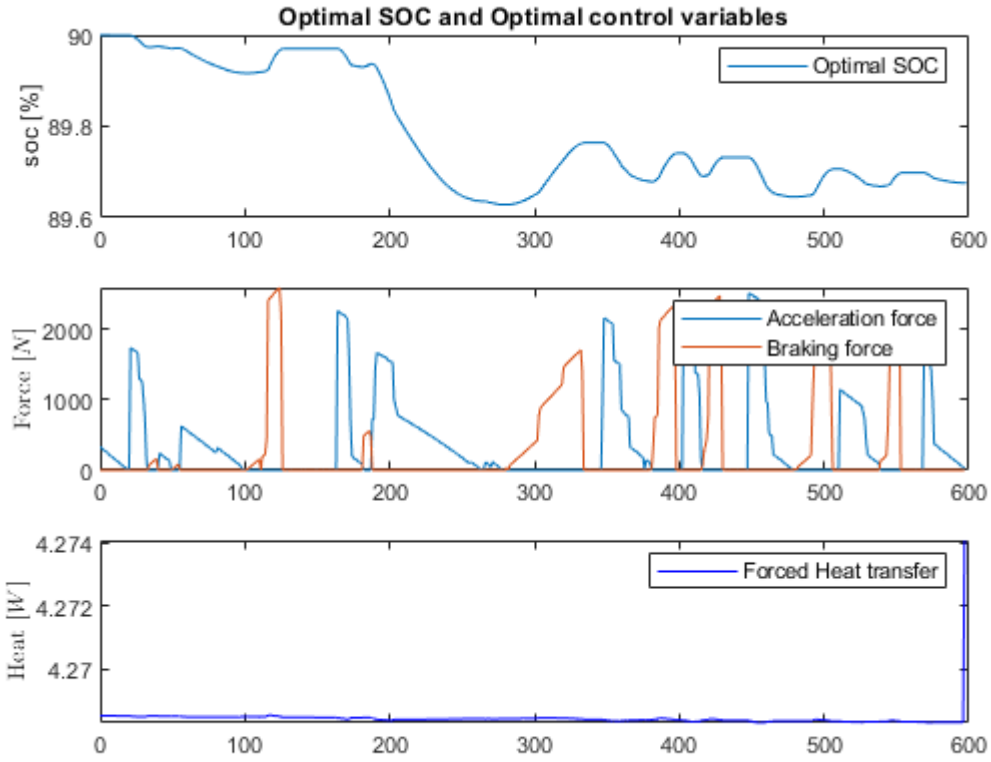

Figure 44 Results, high-lvl thermal dynamics 4

As in the first experiment, Figure 41 shows the optimized velocity profile along the path and is compared to the constraints. It also shows a comparison between the optimized velocity profile and the velocity profile of the FTP75.

Figure 42 like the previous figure, shows the optimized velocity profile, but this time the optimized control variables are shown.

Figure 43 shows the evolution of the state of charge throughout the experiment. The optimized control variables are also shown to be able to see relationships between them.

Finally, Figure 44 shows the evolution of the temperature and optimized control variables.

In view of the results, it can be concluded that the optimizer works correctly for both models. In this last experiment, a reasonable temperature evolution has been shown.

6.1.2 Energy Management Strategy: low-level controller

To test the code developed for the low-level controller, the following experiment has been carried out:

FTP75 has been taken as the current road velocity for simplicity. However, real-time data of the velocity of the road could have been taken. The solution provided by the high-level controller has been

taken as the initial condition and it has also been considered in the objective function. The time horizon for this low-level controller is 20 seconds. These are the results obtained:

Note: The experiment has only been solved for a journey of 50 seconds since each time step the time horizon is moved, the optimization problem is solved, and it is computationally expensive.

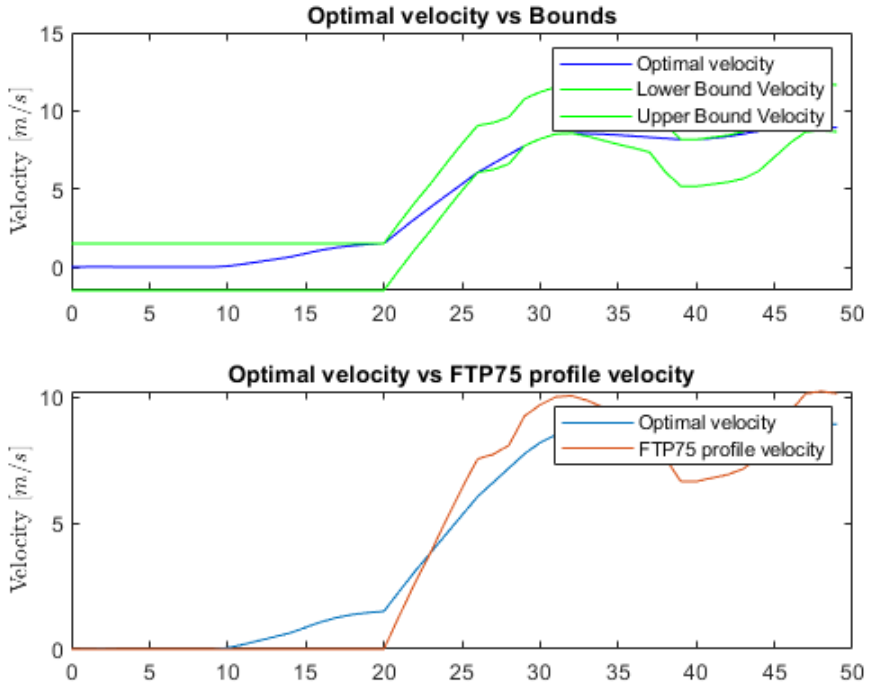

Figure 45 Results, low-lvl thermal dynamics 1

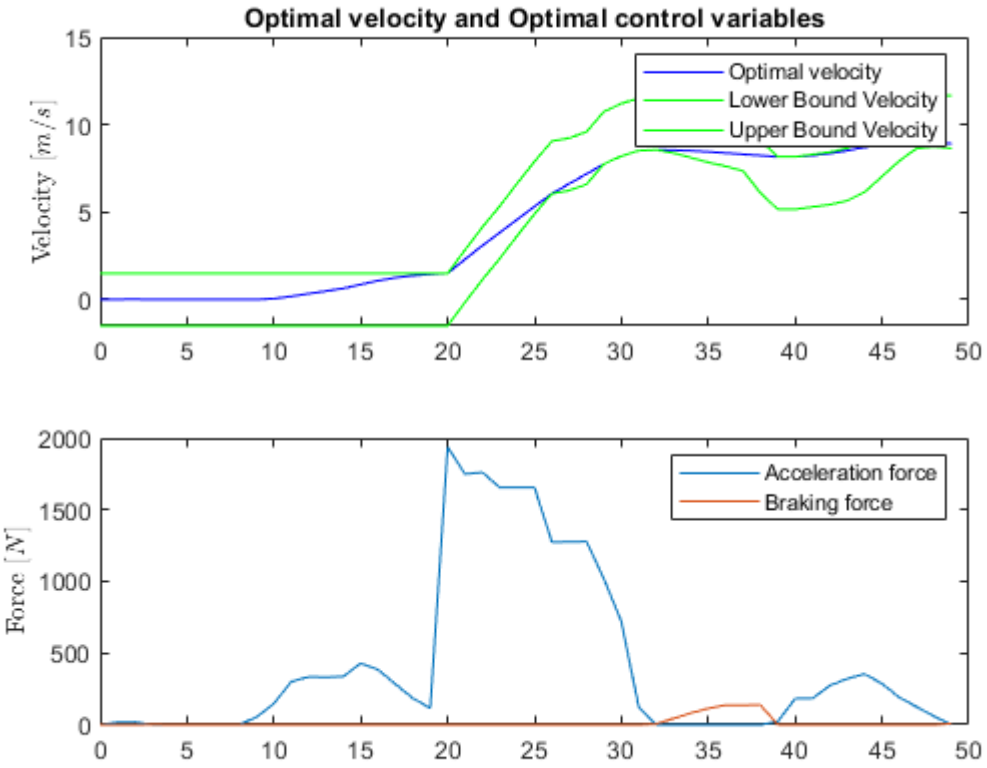

Figure 46 Results, low-lvl thermal dynamics 2

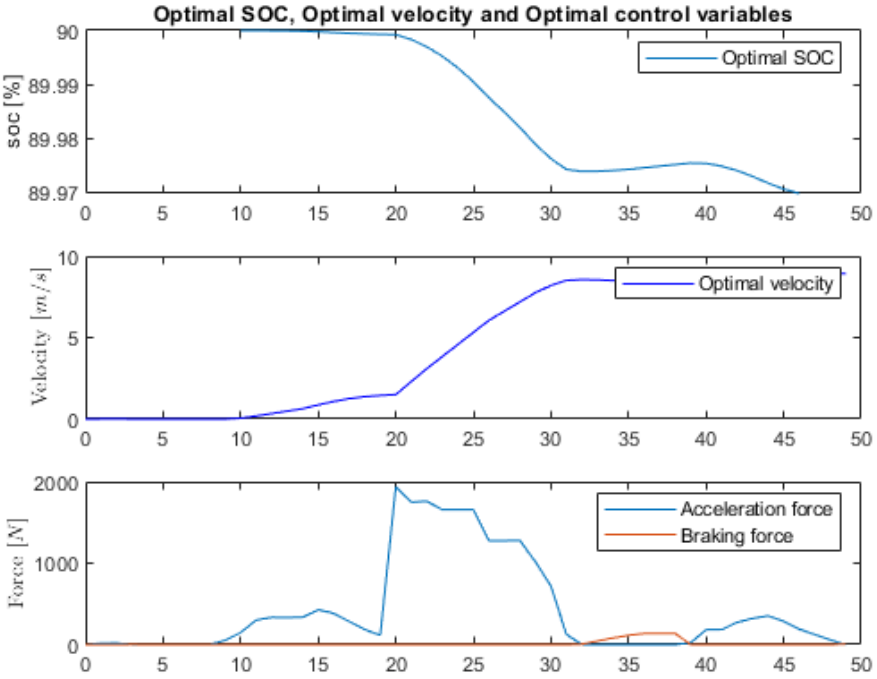

Figure 47 Results, low-lvl thermal dynamics 3

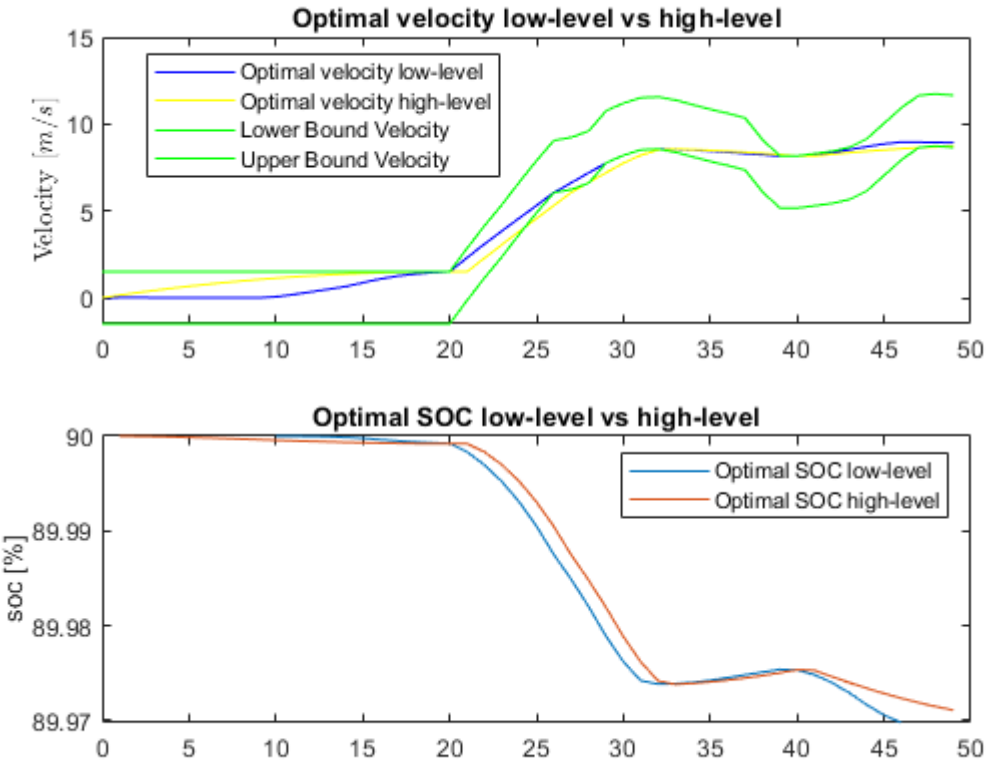

Figure 48 Results, low-lvl thermal dynamics 4

Figure 45, Figure 46, and Figure 47 show similar results than in the previous experiments.

Figure 48 shows a comparison between the high-level controller and the low-level controller.

As expected, the high-level controller gives more optimal results as the time horizon is the entire trip.

6.2 Conclusion

The results obtained in the previous section show success in meeting the objectives, both in the analysis and implementation of an energy management control and in the rest of the objectives.

In my opinion, I believe that I have made valuable advances in the research of Dr. Baisravan HomChaudhuri. As has been explained throughout the project, the need to optimize the performance, efficiency and autonomy of electric vehicles is crucial for the near future, and a control capable of carrying out these improvements has been explored.

In addition, this type of developed control is very versatile and can be applied to other types of vehicles or even processes.

Finally, both the developed system models and explored nonlinear dynamics linearization tools also have various potential uses.

7 Future potential Projects

From this project, it is possible to follow the following lines of research:

- Validate the models with real data from electric vehicles.
- Develop a model with the vehicle motor curves.
- Implement the model developed with the internal resistance of the battery to the optimization program.
- Implement hierarchical control in a real data environment and obtain data in real-time.

8 Budget

8.1 Budget tables

Quantity	Unit	Description	Cost/U (€)	Cost (€)
6	month	MathWorks Matlab/Simulink	70.00	420.00
4	month	Visual Studio Code	0.01	0.04
6	month	Office 365 License	5.53	33.18
			Total	453.22

Table 4 Budget, software used

Quantity	Unit	Description	Cost/U (€)	Cost (€)
300	h	Graduate Engineer (Industrial and Robotics)	40.00	12 000.00
23	h	Ph.D. Engineer (Tutor, Mechanical and Electrical)	50.00	1150.00
			Total	13 150.00

Table 5 Budget, Labor

Budget

Quantity	Unit	Description	Cost/U (€)	Cost (€)
1	u	Software	453.22	453.22
1	u	Labor	13 150.00	13 150.00
			Total	13 603.22

Table 6 Budget, Final budget

The budget amounts to THIRTEEN THOUSAND SIX HUNDRED THREE AND TWENTY-TWO EUROS.

9 References

Amini, M. R., Kolmanovsky, I., & Sun, J. (2020). Hierarchical MPC for Robust Eco-Cooling of Connected and Automated Vehicles and Its Application to Electric Vehicle Battery Thermal Management. *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY*.

Bauer, S., Suchaneck, A., & León, F. P. (2013). Thermal and energy battery management optimization in electric vehicles using Pontryagin's maximum principle. *Journal of Power Sources*.

BenoitH. (2021, January 7). *SIEMENS CORPORATION.* Retrieved from https://blogs.sw.siemens.com/simcenter/race-to-the-clouds-battery-thermal-behavior-simulation-of-a-tesla-model-3/

Bower, G. (2019, October 7). Retrieved from https://insideevs.com/reviews/372819/tesla-model-s-plaid-powertrain-specs-examined/

HomChaudhuri, B., & Bhattacharyya, V. (2022). Distributed Model Predictive Control for Connected and Automated Vehicles in the Presence of Uncertainty. *The American Society of Mechanical Engineers*.

Košuda, M., Novotňák, J., & Fiĺko, M. (2020). Energy-Oriented Trajectory Optimization of Solar Aircraft using fmincon Function in MATLAB. *International Conference on Military Technologies (ICMT)*.

Parliament, E. (2022, November 3). Retrieved from https://www.europarl.europa.eu/news/en/headlines/economy/20221019STO44572/eu-ban-on-sale-of-new-petrol-and-diesel-cars-from-2035-explained

SOTOUDEH, S. M. (2022). PREDICTIVE ENERGY MANAGEMENT OF CONNECTED HYBRID ELECTRIC VEHICLES IN THE PRESENCE OF UNCERTAINTY. *Doctor of Philosophy in Mechanical and Aerospace Engineering in the Graduate College of the Illinois Institute of Technology*.

Sotoudeh, S. M., & HomChaudhuri, B. (2022). A Deep Learning-based Approach to Eco-driving based Energy Management of Hybrid Electric Vehicles. *IEEE TRANSACTIONS ON TRANSPORTATION ELECTRIFICATION*.

Sotoudeh, S. M., & HomChaudhuri, B. (2022). Velocity Optimization and Robust Energy Management of Connected Power-Split Hybrid Electric Vehicles. *The American Society of Mechanical Engineers*.

Wikipedia. (2023, July). Retrieved from https://en.wikipedia.org/wiki/FTP-75

x-engineer. (n.d.). *EV design – electric motor*. Retrieved from x-engineer: https://x-engineer.org/ev-design-electric-motor/