Assignment #1: Quasi-static HEV model and Rule Based Control

Group information

Group number: 4

Students:

- Giacomo Chiavetta, 348027
- Massimo Stella, s348062
- Tommaso Massone, s349235

Load the cycle and vehicle data

Project starts with the provided folders acquisition by means of the following commands:

```
clear;clc;close all

addpath("models");      % Add the "models" directory to the search path
addpath("data");        % Add the "data" directory to the search path
addpath("utilities");   % Add the "utilities" directory to the search path
```

Load cycle data

The project considers the *WLTP* cycle, whose corresponding data are stored within the following directory.

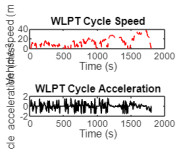
```
mission = load("data\WLTP.mat"); % WLTP data are now contained in the struct
mission variable
time = mission.time_s; % (s)  Vector of the time instants within the considered
cycle
vehSpd = mission.speed_km_h ./ 3.6; % (m/s) Vector of the instant velocities
referred to each time value
vehAcc = mission.acceleration_m_s2; % (m/s^2) Vector of the instant acceleration
referred to each time value
```

Plot the mission cycle data

The velocity and acceleration profiles of the WLTP cycle are now being plotted as a function of time.

```
t = tiledlayout(2,1); % Create a 2-row, 1-column tiled layout for multiple plots

nexttile(1) % Select the first tile (upper plot)
```

```
plot(time, vehSpd, LineStyle="--", Color="r", LineWidth=1); % Plot vehicle speed
over time
title("WLPT Cycle Speed");
xlabel("Time (s)");
ylabel("Vehicle speed (m/s)");

nexttile(2) % Select the second tile (lower plot)
plot(time, vehAcc, LineStyle="-", Color="k", LineWidth=1); % Plot vehicle
acceleration over time
title("WLPT Cycle Acceleration");
xlabel("Time (s)");
ylabel("Vehicle acceleration (m/s^2");
```



Load the vehicle data

The vehicle parameters have been retrieved and the relevant data pertaining to our workgroup have been substituted through the provided function "scaleVehData".

```
veh = load("data\vehData.mat"); % Vehicle data are now contained in the struct veh
variable

% Scale vehicle data according to group parameters

veh = scaleVehData(veh, 66e3, 82e3, 945); % Rated motor power, Rated engine power,
Battery nominal energy, respectively
```

Simulation loop

This code simulates the operation of a series hybrid electric vehicle (HEV), updating the state of charge (SOC) and engine state over time. It calculates engine speed, torque by means of the "thermostatControl" and consequently the fuel flow rate, finally it updates the SOC based on the vehicle longitudinal dynamics considering the parameters of the drivetrain using a backward quasi-static model.

```
% Initialize the state of charge
SOC = zeros(length(time),1); % Create a vector to store SOC values over time
SOC(1) = 0.6; % Initial SOC value (60%)
engState = zeros(length(time),1); % Create a vector for engine state (0 = OFF, 1 =
ON)
engState(1) = 0; % Initial engine state (engine OFF)


for n = 1 : length(time)
    % Compute engine speed and torque using the thermostat control strategy
```

```
    [engSpd(n), engTrq(n)] = thermostatControl(SOC(n), engState(n), vehSpd(n),
vehAcc(n), veh);

    if n < length(time)
        % Update SOC, fuel flow rate, feasibility, and profile for the current and
next time step
        [SOC(n+1), fuelFlwRate(n), unfeas(n), prof(n)] = hev_model(SOC(n),
[engSpd(n), engTrq(n)], [vehSpd(n), vehAcc(n)], veh);
        % Update engine state: 1 if engine is on (torque > 0), else 0
        engState(n+1) = engTrq(n) > 0;
    else
        % At the last time step, update the current SOC and other parameters
without advancing
        [SOC(n),fuelFlwRate(n), unfeas(n), prof(n)] = hev_model(SOC(n), [engSpd(n),
engTrq(n)], [vehSpd(n), vehAcc(n)], veh);
        % Assign the engine state for the final time step
        engState(n) = engTrq(n) > 0;
    end
end
```

Results analysis

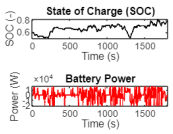
Battery Power and State of Charge (SOC)

The first result obtained is a graphical representation of the battery's state of charge (SOC), compared with the power absorbed and released by the battery throughout the simulation. It can be observed that during discharging intervals, the battery power is positive, whereas during charging, it is negative. Regarding the SOC trend, it should be clarified that, as intended by the engine controller, its value oscillates within the imposed limits, varying according to the engine start-up and shutdown strategy. In fact, certain scenarios force the engine to shut down during charging and turn on during pure electric mode changing the vehicle operating mode.

The SOC trend can be judged flawed since, as it will be demostreted in the subsequent results, the engine strategy leads to a overproduction of generator energy that keep the battery state of charge close to the upper boundary (SOC = 0.7) which is, consequently, often exceeded during regenerative braking.

```
t = tiledlayout(2,1); % Create a 2-row, 1-column tiled layout for multiple plots

ax1 = nexttile; % Select the first tile (upper plot)
plot(time, SOC, Color="k", LineWidth=1) % Plot the state of charge (SOC) over time
title("State of Charge (SOC)");
xlabel("Time (s)");
ylabel("SOC (-)");

finalSOC = SOC(n);

ax2 = nexttile; % Select the second tile (lower plot)
% Plot the instantaneous battery power over time
```

```
plot(time, [prof.battVolt] .* [prof.battCurr], Color="r", LineWidth=1)
title("Battery Power");
xlabel("Time (s)");
ylabel("Power (W)");

linkaxes([ax1,ax2], 'x') % Synchronizes zoom on the X-axis between ax1 and ax2
```

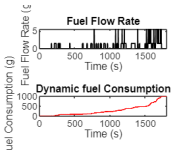


Fuel flow rate and total fuel consumed

Subsequently an insight of the instantaneous engine fuel flow rate and the total fuel consumed have been provided. To better analyse the trend of the fuel consumed over the simulation the so called "Dynamic Fuel consumption" has been defined and used to strengthen the graphical representation of the instantaneous fuel flow rate by showing how the fuel is consumed throughtout the engine operation. Thanks to the first figure it is possible to notice what is the amount of the fuel flow rate during maximum power utilization compared to the one during enigne optimal opearation. The vastly usage of this energy-intensive working point leads to a large fuel consumptions during the relative intervals. By means of the obtained results the requested outputs have been retrieved.

```
% Calculate cumulative fuel consumption using numerical integration (trapz)
FlwCnsp_dyn = 0; % Initialize the "Dynamic Fuel consumption" variable

for n = 1: length(time)-1
    % Perfom iterative summation of each instant fuel consumption contribution
    FlwCnsp_dyn(n+1) = trapz(fuelFlwRate(n:n+1)) + FlwCnsp_dyn(n);
end

% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption = trapz(fuelFlwRate); % (g)

% Convert the fuel consumed from mass to volume quantity
Fuel_l = fuelConsumption/(veh.eng.fuelDensity*1e3); % (l)

% Calculate the cycle total dinstance using numerical integration (trapz)
distance = trapz(vehSpd)/1e3; % (Km)

fuelEconomy = Fuel_l/distance * 100; % (l/100km)

figure
t = tiledlayout(2,1); % Create a 2-row, 1-column tiled layout for multiple plots

ax1 = nexttile; % Select the first tile (upper plot)
plot(time, fuelFlwRate, Color="k", LineWidth=1) % Plot instantaneous fuel flow rate
over time
```

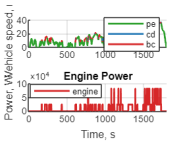
```
title("Fuel Flow Rate");
xlabel("Time (s)");
ylabel("Fuel Flow Rate (g/s)");

ax2 = nexttile; % Select the second tile (lower plot)
plot(time,FlwCnsp_dyn, Color="r", LineWidth=1) % Plot the cumulative fuel
consumption over time
title("Dynamic fuel Consumption");
xlabel("Time (s)");
ylabel("Fuel Consumption (g)");

linkaxes([ax1,ax2], 'x') % Synchronizes zoom on the X-axis between ax1 and ax2
```



Vehicle speed profile and engine power

The following figures highlight how the engine has been controlled, showing through the second plot the power requested to the engine, whose values varies between zero, the optimal power and the maximum one . The first plot, instead, shows the velocity profile over the simulation marking the intervals according to the vehicle operating mode demostrating that, on the basis of the engine operation, the vehicle moves in: (p.e.) pure electric mode when the engine is off, (b.c.) battery charging mode when the engine is generating power that, either contributing to vehicle traction or not, increase the state of charge (SOC) of the battery and (c.d.) charge depleting mode when the engine is used for traction but the requested power from the traction motor is relatevely large and the coperation of both engine and battery is needed to meet the load. The engine power trend follows the one of the fuel flow rate already commented.

```
% Plot the speed profile and the engine power over time
fig = powerProfiles(prof, 'eng');
title("Engine Power")
```

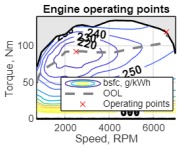


ICE efficiency map

As already explained the engine working points are three and the following figure illustrates the efficiency of the internal combustion engine (ICE) while operating in these points. It is relevant to note that the break specific fuel consumption (bsfc) of the optimal point is 212.32 Wh/kg while the one of the maximum power amounts to 245.83 Wh/kg resulitng in an increase of the consumptions of 15.8 % for each unit mass of fuel.

```
% Plot engine fuel consumption map in which all exploited working points are marked
```

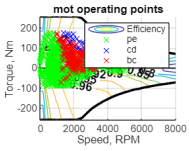
```
engMapWithPF(veh.eng, prof, 'bsfc');
```



Traction motor effiiciency map

In the following graph, instead, the efficiecy map of traction motor is represented and its operating working points are shown. The performance of this device depends completely on the exogeneous variables linked to the WLTP cycle but it is worth to notice that the vehicle operation modes are marked in the same way as before and it is possible to asses some relationships between them and the motor operating points. It can be noticed, for instance, that the charge depleting mode occours for high power requesting points and pure elettric one for lower ones.

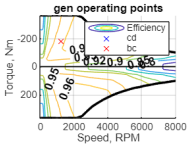
```
% Plot traction motor efficiency map with all exploited working points marked
[fig, ax] = emMapWithPF(veh.mot, prof, "mot", ["all"]);
```



Generator efficiency map

In this figure the generator efficiency map is represented and the engine working points are marked considering the torque and speed ratio existing bewtween the two machines. The choice of these points is linked to the controlled variables of the combustion engine, which are well known. Nevertheless the considered efficiency used in the engine controller was fixed to a costant value of 0.9.

```
% Plot generator efficiency map in which all working points are marked
[fig, ax] = emMapWithPF(veh.gen, prof, "gen", ["all"]);
```



Save results

In this section, the data related to the WLTP operating cycle has been acquired. The "prof" structure stores the parameters related to the vehicle dynamics and the performance of the main subsystems of the architecture, including the internal combustion engine, generator, electric motor, and battery pack. Additionally, the data on fuel consumption (kg), fuel economy (l/100 km), and the final State of Charge (SOC) of the battery at the end of the cycle have been recorded.

```
% Store results
save("results.mat", "prof", "fuelConsumption", "fuelEconomy", "finalSOC")
```

Simulation loop with implemented controller

Now the simulation loop is repeated exploiting the implemented version of the controller : "awesomeControl".

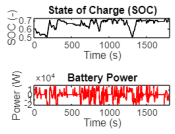
```
% Initialize the state of charge
SOC = zeros(length(time),1); % Create a vector to store SOC values over time
SOC(1) = 0.6; % Initial SOC value (60%)
engState = zeros(length(time),1); % Create a vector for engine state (0 = OFF, 1 =
ON)
engState(1) = 0; % Initial engine state (engine OFF)

for n = 1 : length(time)
    % Compute engine speed and torque using the thermostat control strategy
    [engSpd(n), engTrq(n)] = awesomeControl(SOC(n), engState(n), vehSpd(n),
vehAcc(n), veh);

    if n < length(time)
        % Update SOC, fuel flow rate, feasibility, and profile for the current and
next time step
        [SOC(n+1), fuelFlwRate(n), unfeas(n), prof(n)] = hev_model(SOC(n),
[engSpd(n), engTrq(n)], [vehSpd(n), vehAcc(n)], veh);
        % Update engine state: 1 if engine is on (torque > 0), else 0
        engState(n+1) = engTrq(n) > 0;
    else
        % At the last time step, update the current SOC and other parameters
without advancing
        [SOC(n),fuelFlwRate(n), unfeas(n), prof(n)] = hev_model(SOC(n), [engSpd(n),
engTrq(n)], [vehSpd(n), vehAcc(n)], veh);
        % Assign the engine state for the final time step
        engState(n) = engTrq(n) > 0;
    end
end
```

Results analysis

Battery Power and State of Charge (SOC)

After the controller implementation many other enigne working points have been exploited leading to a much more constrained state of charge boundaries.

```
figure
t1 = tiledlayout(2,1); % Create a 2-row, 1-column tiled layout for multiple plots

ax1 = nexttile; % Select the first tile (upper plot)
plot(time, SOC, Color="k", LineWidth=1) % Plot the state of charge (SOC) over time
title("State of Charge (SOC)");
xlabel("Time (s)");
ylabel("SOC (-)");

finalSOC = SOC(n);

ax2 = nexttile; % Select the second tile (lower plot)
% Plot the instantaneous battery power over time
plot(time, [prof.battVolt] .* [prof.battCurr], Color="r", LineWidth=1)
title("Battery Power");
xlabel("Time (s)");
ylabel("Power (W)");

linkaxes([ax1,ax2], 'x') % Synchronizes zoom on the X-axis between ax1 and ax2
```




```

```

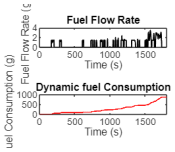
Fuel flow rate and total fuel consumed

As it will be discuss later the engine working points vary continuously but never approaching energy-intensive points. This result in a lower and more efficient energy generation and, consequently, an appreciable fuel saving.

The requested outputs have been retrieved:

```
% Calculate cumulative fuel consumption using numerical integration (trapz)
FlwCnsp_dyn = 0; % Initialize the "Dynamic Fuel consumption" variable

for n = 1: length(time)-1
    % Perfom iterative summation of each instant fuel consumption contribution
    FlwCnsp_dyn(n+1) = trapz(fuelFlwRate(n:n+1)) + FlwCnsp_dyn(n);
end

% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption = trapz(fuelFlwRate); % (g)

% Convert the fuel consumed from mass to volume quantity
Fuel_l = fuelConsumption/(veh.eng.fuelDensity*1e3); % (l)
```

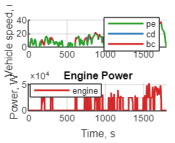
```
% Calculate the cycle total dinstance using numerical integration (trapz)
distance = trapz(vehSpd)/1e3; % (Km)

fuelEconomy = Fuel_l/distance * 100; % (l/100km)

figure
t = tiledlayout(2,1); % Create a 2-row, 1-column tiled layout for multiple plots

ax1 = nexttile; % Select the first tile (upper plot)
plot(time, fuelFlwRate, Color="k", LineWidth=1) % Plot instantaneous fuel flow rate
over time
title("Fuel Flow Rate");
xlabel("Time (s)");
ylabel("Fuel Flow Rate (g/s)");

ax2 = nexttile; % Select the second tile (lower plot)
plot(time,FlwCnsp_dyn, Color="r", LineWidth=1) % Plot the cumulative fuel
consumption over time
title("Dynamic fuel Consumption");
xlabel("Time (s)");
ylabel("Fuel Consumption (g)");

linkaxes([ax1,ax2], 'x') % Synchronizes zoom on the X-axis between ax1 and ax2
```



Vehicle speed profile and engine power

The vehicle speed profile now shows an increased usage of the battery power during traction and a reduce usage of the engine power generation. This results in a more frequent charge depleting mode in place of battery charging one, consinstetly with the reduce engine power generation as demostrated in the second figure.

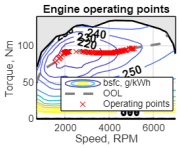
```
% Plot the speed profile and the engine power over time
fig = powerProfiles(prof, 'eng');
title("Engine Power")
```



ICE efficiency map

The engine efficiency map, consistently with the engine control strategy, shows the much more efficient way the enigne operate. The optimal working point remains the main target but many others points have been considered along the optimum operating line. It is important to note that also points less energy-intensive than the optimal one have been considered but still lying one the optimal operating line.

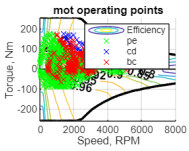
```
% Plot engine fuel consumption map in which all exploited working points are marked
engMapWithPF(veh.eng, prof, 'bsfc');
```



Traction motor effiiciency map

Similarly to what already commented before the traction motor efficiency map shows the working point related to the vehicle dynamics. The marked vehicle mode points changes according to the speed profile ones.

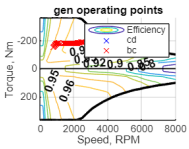
```
% Plot traction motor efficiency map in which all exploited working points are
marked
[fig, ax] = emMapWithPF(veh.mot, prof, "mot", ["all"]);
```



Generator effiiciency map

The generator efficiency map now show a much more different operating point distribution. The information of the actual generator efficiency increase the complexity of the code since the engine performance should be defined according this information in advance with respect to the designation of the generator efficiency. This last consideration reveals the incompleteness of the produced engine controller since just one value was considered for sake of simplicity. This lack of refinement could be solved using an iterative cycle procedure over the generator efficiency.

```
% Plot generator efficiency map in which all engine working points are
% marked considering the torque and speed ratio existing bewtween the two
% machines
[fig, ax] = emMapWithPF(veh.gen, prof, "gen", ["all"]);
```



(Optional) Improving the controller

The developed controller governs the operation of the internal combustion engine in a hybrid electric vehicle (HEV), optimizing its performance based on operating conditions. The analysis of the WLTP cycle revealed a fuel consumption of 1.003 kg, with a specific consumption of 5.49 liters per 100 km. Subsequently, an optimized version of the engine management algorithm was implemented to enhance the overall efficiency of the propulsion system. This optimization resulted in a reduction in fuel consumption, with recorded values of 0.872 kg and an average consumption of 4.78 liters per 100 km. This improvement is attributable to the advanced management of the engine's operating points.

It can be observed that there was a noticeable fuel savings, as the final SOC using the first control was 0.78, while with the second control it was 0.73. This demonstrates that, despite the fuel savings, the SOC values are very similar, indicating that the optimization did not significantly affect the battery state of charge, this indicates that the optimization has led to an actual improvement in efficiency without significantly affecting the battery state of charge.

```
% Store results
save("results_extra.mat", "prof", "fuelConsumption", "fuelEconomy", "finalSOC")
```

The thermostat controller

This project is related to an assignment focused on developing a rule-based thermostat controller for a series hybrid electric vehicle (HEV). The goal is to optimize the operation of the internal combustion engine (ICE) to ensure efficient energy management while maintaining battery performance. The controller determines when to turn the engine ON or OFF, adjusting its speed and torque based on the battery's state of charge (SOC) and the power demand from the drivetrain.

The following function is composed by :

Inputs:

- SOC: State of Charge of battery

- engState: Current state of the engine (ON/OFF)

- vehSpd: Vehicle speed

- vehAcc: Vehicle acceleration

- veh: Vehicle parameters and characteristics

Outputs:

- engSpd: Engine speed

- engTrq: Engine torque

- vehPrf: Vehicle performance metrics

```
function [engSpd, engTrq, vehPrf] = thermostatControl(SOC, engState, vehSpd,
vehAcc, veh)

% Define the SOC threshold
SOChi = 0.7; % Upper SOC limit
SOClo = 0.5; % Lower SOC limit

% Define engine characteristics analysing the considered working points through the
use of functions
% The ICE operates within an optimal speed and torque range
w_idle = veh.eng.idleSpd; % Engine idle speed
w_max = veh.eng.maxSpd; % Maximum engine speed
w_speed = linspace(w_idle,w_max,1000); % Speed range of 1000 values for
calculations from w_min to w_max
T_ool = veh.eng.oolTrq(w_speed); % The values of the optimal torque have been
obtained by entering the speed values previously created in the function
T_max =  veh.eng.maxTrq(w_speed); % The values of the maximal torque have been
obtained by entering the speed values previously created in the function
P_max = T_max .* w_speed; % Maximum power values
```

In this section is possible to defines key engine operating points in order to efficiency and maximum power.

The optimal operating point (*w_optimal, T_optimal, P_optimal*) ensures fuel efficiency, while the maximum power point (*w_Pmax, T_Pmax*) allows the engine to deliver peak performance. The commented plotting lines would visualize engine efficiency and power characteristics if activated.

```
% Determine the optimal engine operating point for fuel efficiency
[a,b] = min(veh.eng.bsfcMap(w_speed,T_ool)); % The values of the engine speed
(w_optimal) that minimizes the brake specific fuel consumption have been founded
w_optimal = w_speed(b); % Optimal engine speed for maximum effiency
T_optimal = veh.eng.oolTrq(w_optimal); % Retrieve the corresponding torque at the
optimal speed
P_optimal = T_optimal * w_optimal; % Compute the power at the optimal efficiency
point

% Determine the maximum power oprating point
[c,d] = max(P_max); % The index corresponding to the maximum power output have been
founded
w_Pmax = w_speed(d); % Retrieve the engine speed at which maximum power is delivered
T_Pmax = veh.eng.maxTrq(w_Pmax); % Retrieve the corresponding maximum torque at
peak power
```

It is feasible determines whether the internal combustion engine (ICE) should be turned ON or OFF based on the battery's state of charge (SOC).

If the SOC exceeds the upper threshold (SOChi), the engine is turned off to avoid overcharging. Conversely, if the SOC drops below the lower threshold (SOClo), the engine is turned on to recharge the battery and maintain sufficient energy levels.

```
% Phase 1: Determine whether to turn the engine On or OFF
% The engine state is updated based on the SOC thresholds
if SOC > SOChi

    engState = 0; % If SOC is above upper limit -> Engine is turned OFF

elseif SOC < SOClo

    engState = 1; % If SOC is below lower limit -> Engine is turned ON
end
```

The function then calculates the power demand (demPwr) based on vehicle speed and acceleration.

If the vehicle was previously operating in pure electric mode, the engine starts in the OFF state. If the engine is OFF, speed and torque are set to zero. If the engine is ON, its operating point is determined based on demand: it runs at optimal efficiency if the power requirement is low (demPwr <= P_optimal) or at maximum power if demand exceeds the optimal level.

```
% Phase 2: Compute power demand
% Calculate shaft speed and torque using the vehicle speed, accelleration and
drivetrain parameters
[shaftSpd, shaftTrq, vehPrf] = hev_drivetrain(vehSpd, vehAcc, veh);
demPwr = shaftSpd*shaftTrq; % Compute the power demand from the drivetrain

% Determine the engine speed, torque and power based on demand
if engState == 0
    % If the engine was previously OFF (pure electric mode), keep it OFF
    engSpd = 0; % Engine speed is zero
    engTrq = 0; %Enginje torque is zero
    engPwr = engSpd*engTrq; % Clearly, engine power is zero

elseif engState == 1 & demPwr <= 0
    % If the engine is ON but no power is required (demPwr <= 0), run at optimal
efficiency
    engSpd = w_optimal; % Set engine to the optimal effiency point
    engTrq = T_optimal; % Set torque to the corresponding optimal value
    engPwr = engSpd*engTrq; % Compute engine power

elseif engState == 1 & demPwr > 0
    % If the engine is ON and power is required, adjust engine operation
    if demPwr <= P_optimal
        % If demand is within the optimal power range, operate efficiently
```

```
        engSpd = w_optimal;
        engTrq = T_optimal;
        engPwr = engSpd*engTrq;

    elseif demPwr > P_optimal
        % If demand exceeds the optimal power, operate at max power
           engSpd =  w_Pmax; % Set engine speed to maximum power point
           engTrq = T_Pmax; % Set corresponding torque
           engPwr = engSpd*engTrq; % Compute maximum engine power
    end
end
```

Third phase ensures that the battery power stays within its operating limits by adjusting engine power accordingly.

If the power demand exceeds the battery's maximum limit, the engine compensates by increasing its output. Conversely, if the power demand is negative (regeneration or low load), the system ensures the engine does not overcharge the battery. If excess power cannot be absorbed, the engine is turned off to prevent overcharging.

```
% Phase 3: Battery power constraints
% Ensure that battery power does not exceed its operating limits
eff_gen = 0.9; % Given data of Generator efficiency (90%)
% Assign motor speed and torque based on drivertrain outputs
motSpd = shaftSpd; % Motor speed is equal to the drivertrain shaft speed
motTrq = shaftTrq; % Motor torque is equal to the drivertrain shaft torque

eff_mot = veh.mot.effMap(motSpd, motTrq); % Compute motor efficiency based on its
efficiency map

if  demPwr > 0 & (demPwr/(eff_mot) - engPwr*eff_gen) > veh.batt.maxPwr(SOC)
    % If power demand is positive and exceeeds the battery's maximum proper power
    engState = 1; %Ensure the engine stays ON
    engPwr = max(P_optimal,(demPwr/(eff_mot) - veh.batt.maxPwr(SOC))*(1/eff_gen));
% Adjust engine power to prevent the battery from exceeding its power limit

    if engPwr == P_optimal
        % If the required power is within the optimal range, set the engine at
optimal efficiency
        engSpd = w_optimal;
        engTrq = T_optimal;

    elseif engPwr > P_optimal
        % If more power is needed, operate at maximum power
        engSpd = w_Pmax;
        engTrq = T_Pmax;
    end
```

```
elseif demPwr < 0 & engState == 1 % If power demand is negative (e.g., regenerative
braking) and the engine is ON
    % Adjust engine power to avoid overchanging the battery
    engPwr = min(P_optimal, (demPwr*eff_mot - veh.batt.minPwr(SOC))*(1/eff_gen));

    if (demPwr - eff_gen*engPwr) >= veh.batt.minPwr(SOC) % Check if the calibration
of engine power still keeps the battery within limits

        if engPwr == P_optimal
            % Keep the engine at optimal efficiency if the power remains within
limits
            engSpd =  w_optimal;
            engTrq = T_optimal;

        elseif engPwr < P_optimal
               % If less power is needed, turn OFF the engine in order to prevent
unnecessary fuel consumption
               engState = 0;
               engSpd = 0;
               engTrq = 0;

        end

    elseif (demPwr * eff_mot - eff_gen * engPwr) < veh.batt.minPwr(SOC)
            % If the power demand is too low and cannot be absorbed, turn OFF the
engine
            engState = 0;
            engSpd = 0;
            engTrq = 0;
    end
    end
end
```

A variant of the thermostat controller

The improved version of the Thermostat Control function builds upon the basic implementation, introducing optimizations that enhance its performance. As previously described, this version refines the control strategy by operating at more optimal points, leading to increased efficiency and reduced fuel consumption. These improvements allow for a more effective management of the internal combustion engine, ensuring better adaptability to different operating conditions while minimizing energy losses.

```
function [engSpd, engTrq, vehPrf] = awesomeControl(SOC, engState, vehSpd, vehAcc,
veh)

% Define the SOC threshold
```

```
SOChi = 0.7; % -
SOClo = 0.5; % -

% Define engine characteristics analysing the considered working points through the
use of functions
% The ICE operates within an optimal speed and torque range

w_idle = veh.eng.idleSpd; % Engine idle speed
w_max = veh.eng.maxSpd; % Maximum engine speed
w_speed = linspace(w_idle,w_max,1000); % Speed range of 1000 values for
calculations from w_min to w_max
T_ool = veh.eng.oolTrq(w_speed); % The values of the optimal torque have been
obtained by entering the speed values previously created in the function
P_ool = T_ool .* w_speed; % Optimal Power values
T_max =  veh.eng.maxTrq(w_speed); % The values of the maximal torque have been
obtained by entering the speed values previously created in the function
P_max = T_max .* w_speed; % Maximum power values
```

In this section is possible to defines key engine operating points in order to efficiency and maximum power.

The optimal operating point (*w_optimal, T_optimal, P_optimal*) ensures fuel efficiency, while the maximum power point (*w_Pmax, T_Pmax*) allows the engine to deliver peak performance. The commented plotting lines would visualize engine efficiency and power characteristics if activated.

```
% Determine the optimal engine operating point for fuel efficiency
[a,b] = min(veh.eng.bsfcMap(w_speed,T_ool)); % The values of the engine speed
(w_optimal) that minimizes the brake specific fuel consumption have been founded
w_optimal = w_speed(b); % Optimal engine speed for maximum effiency
T_optimal = veh.eng.oolTrq(w_optimal); % Retrieve the corresponding torque at the
optimal speed
P_optimal = T_optimal * w_optimal; % Compute the power at the optimal efficiency
point

% Determine the maximum power operating point
[c,d] = max(P_max); % The index corresponding to the maximum power output have been
founded
w_Pmax = w_speed(d); % Retrieve the engine speed at which maximum power is delivered
T_Pmax = veh.eng.maxTrq(w_Pmax); % Retrieve the corresponding maximum torque at
peak power
```

It is feasible determines whether the internal combustion engine (ICE) should be turned ON or OFF based on the battery's state of charge (SOC).

If the SOC exceeds the upper threshold (SOChi), the engine is turned off to avoid overcharging. Conversely, if the SOC drops below the lower threshold (SOClo), the engine is turned on to recharge the battery and maintain sufficient energy levels.

```
% Phase 1: Determine whether to turn the engine On or OFF
% The engine state is updated based on the SOC thresholds
if SOC > SOChi
```

```
    engState = 0; % If SOC is above upper limit -> Engine is turned OFF

elseif SOC < SOClo

    engState = 1; % If SOC is below lower limit -> Engine is turned ON
end
```

The function then calculates the power demand (demPwr) based on vehicle speed and acceleration.

```
% Phase 2: Compute power demand
% Calculate shaft speed and torque using the vehicle speed, accelleration and
drivetrain parameters
[shaftSpd, shaftTrq, vehPrf] = hev_drivetrain(vehSpd, vehAcc, veh);
demPwr = shaftSpd*shaftTrq; % Compute the power demand from the drivetrain

motSpd = shaftSpd; % Create an alias of the traction motor speed for clarecy
motTrq = shaftTrq; % Create an alias of the traction motor torque for clarecy

eff_mot = veh.mot.effMap(motSpd, motTrq); % Find the efficiency of the electric
motor based on the working point

% Eff Gen at optimal point is similar to Eff Gen @ max ool point

genSpd_optimal =  w_optimal * veh.gen.tcSpdRatio; % Calculate the velocity of the
generator multiplying the speed of the ICE by the gear ratio of the generator
genTrq_optimal = - T_optimal / veh.gen.tcSpdRatio; % Calculate the input torque of
the generator multiplying the torque of the ICE by the gear ratio of the generator
% The negative sign (-) indicates that the torque generated by the generator is
opposite to that of the internal combustion engine

eff_gen = veh.gen.effMap(genSpd_optimal, genTrq_optimal); % Find the efficiency of
the generator based on the working point
```

For simplicity, a constant efficiency value for the generator is assumed, equal to the one obtained when the internal combustion engine (ICE) operates at its optimal point.

Above the optimal power value, the efficiency remains virtually constant, while below this value, an error is introduced, which tends to increase. However, this error is disregarded in the analysis, as it does not significantly impact the overall result.

```
% We considered the efficiencies of the traction motor and the generator to make
more accurate comparisons,
% taking into account the losses occurring between the traction motor and the
generator.

P_optimal_shaft = P_optimal * (eff_gen*eff_mot); % Multiply by the efficiency to
find a value closest to the reality
P_max_ool_shaft = max(P_ool) * (eff_gen*eff_mot); % Multiply by the efficiency to
find a value closest to the reality
```

```
% Determine the engine speed, torque and power based on demand
if engState == 0
    % If the engine was previously OFF (pure electric mode), keep it OFF
    engSpd = 0; % Engine speed is zero
    engTrq = 0; %Enginje torque is zero
    engPwr = engSpd*engTrq; % Clearly, engine power is zero

elseif engState == 1 & demPwr <= 0
    % If the engine is ON but no power is required (demPwr <= 0), run at optimal
efficiency
    engSpd = w_optimal; % Set engine to the optimal effiency point
    engTrq = T_optimal; % Set torque to the corresponding optimal value
    engPwr = engSpd*engTrq; % Compute engine power

elseif engState == 1 & demPwr > 0
    % If the engine is ON and power is required, adjust engine operation
    if demPwr <= P_optimal
        % If demand is within the optimal power range, operate efficiently
        engSpd = w_optimal;
        engTrq = T_optimal;
        engPwr = engSpd*engTrq;

    elseif P_optimal_shaft < demPwr & demPwr <= P_max_ool_shaft % Condition in
which the power demand is between the optimal value end the maximum value

        engSpd = veh.eng.oolSpd(demPwr/(eff_mot*eff_gen)); % Select the engine
speed on the OOL according to the power demand
        engTrq = veh.eng.oolTrq(engSpd); % Find the engine torque by using the
function with speed as an input
        engPwr = engSpd*engTrq; % Calculate the power delivered by the engine

    elseif demPwr > P_optimal
        % If demand exceeds the optimal power, operate at max power
           engSpd =  w_Pmax; % Set engine speed to maximum power point
           engTrq = T_Pmax; % Set corresponding torque
           engPwr = engSpd*engTrq; % Compute maximum engine power
    end
end
```

The last phase ensures that the battery power stays within its operating limits by adjusting engine power accordingly.

```
% Phase 3: Add limits to the battery power

if  demPwr > 0 & (demPwr - engPwr) > veh.batt.maxPwr(SOC)
    % If power demand is positive and exceeeds the battery's maximum proper power
    engState = 1; %Ensure the engine stays ON
```

```
    engPwr = max(P_optimal,(demPwr - veh.batt.maxPwr(SOC))*(1/eff_gen)); % Adjust
engine power to prevent the battery from exceeding its power limit
    % If the required power is within the optimal range, set the engine at optimal
efficiency
    if engPwr == P_optimal
        engSpd =  w_optimal;
        engTrq = T_optimal;

    elseif P_optimal < engPwr & engPwr <= max(P_ool) % If the power request is
greater than the power optimal and lower than the maximum power of the OOL
        engSpd = veh.eng.oolSpd(engPwr); % Find the engine speed on the OOL by
using the function with power request as an input
        engTrq = veh.eng.oolTrq(engSpd); % Find the engine torque on the OOL by
using the function with speed as an input

    elseif engPwr > max(P_ool) % If the power request is greater than the maximum
power of the OOL
        engSpd =  w_Pmax; % Select the engine speed equal to the velocity referred
to the maximum power that can be delivered
        engTrq = T_Pmax; % Select the engine torque equal to the torque referred to
the maximum power that can be delivered
    end

elseif demPwr < 0 & engState == 1 % If power demand is negative (e.g., regenerative
braking) and the engine is ON
    % Adjust engine power to avoid overchanging the battery
    engPwr = min(P_optimal, (demPwr - veh.batt.minPwr(SOC))*(1/eff_gen));

    if (demPwr - eff_gen*engPwr) >= veh.batt.minPwr(SOC) % Check if the calibration
of engine power still keeps the battery within limits

        if engPwr == P_optimal
            % Keep the engine at optimal efficiency if the power remains within
limits
            engSpd =  w_optimal;
            engTrq = T_optimal;

        elseif engPwr < P_optimal
            % If less power is needed, turn OFF the engine in order to prevent
unnecessary fuel consumption
            engSpd = veh.eng.oolSpd(engPwr); % Using the optimal value of the
engine speed if the power request is less then the optimal power
            engTrq = veh.eng.oolTrq(engSpd); % Taking the torque corresponding to
the speed previously obtained along the OOL

        end

    elseif (demPwr - eff_gen*engPwr) < veh.batt.minPwr(SOC)
        % If the power demand is too low and cannot be absorbed, turn OFF the
engine
```

```
        engState = 0;
        engSpd = 0;
        engTrq = 0;
    end

end

end
```