# Assignment #3: A-ECMS

**Table of Contents**



## Group information

Group number: 4

Students:

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

## Introduction

As part of the third project, the laboratory activity consists in the realization of a similar controller with respect to the one exploited in the previous project, but with some key differences that allow the control strategy involved to be tested through several homologation cycles and, in general, to be used in a wider and more flexible way.

The speed profile cycles that we are now considering are

- The Artemis Urban cycle (AUDC), repeated five times.
- The Artemis Motorway cycle (AMDC).
- The WLTP.
- The Turin Test cycle (TTC).

and can be found in the "data" folder, while, all the prevoius data and parameters used in the former projects within the simulation system remain unchanged: the car architecture is a series hybrid electric vehicle (HEV), the provided models concern the battery, the car longitudinal dynamics, and the performance of the devices. All the informations are available, respectevely, in the "data" and "models" folder.

The variables within the vehicle system are reported for sake of completeness:

**Exogenous Inputs:**

Vehicle Speed - $VehSpd$

Vehicle Acceleration - $VehAcc$

***State Variables:***

State of Charge - $SOC$

***Control Variables:***

Engine Speed - $EngSpd$

Engine Torque - $EngTrq$

***Stage Cost:***

Fuel Flow Rate - $fuelFlwRate$

As far as the controller is concerned, the ECMS (Equivalent Consumption Minimization Strategy) was originally tested on the ARDC cycle and, as has been demonstrated, the proper and effective use of this local fuel minimization strategy required knowledge of the entire cycle and the tuning of its essential parameter "s", based on the final deviation of the state trajectory from the desired one (the initial State of Charge). The optimization

strategy, in fact, strongly depends on the charge sustaining behavior, and its effectiveness can be assessed according to this criterion by iteratively simulating the entire cycle until the most suitable value of "s" is found.

With reference to the previously employed engine controller, a different tuning of the ECMS equivalent factor "s" can be done in place of the bisectional algorithm as the simulation progresses, leading to a different control approach that can be used to adapt the control logic to different driving conditions. This kind of controller is called Adaptive ECMS, and its working principle is based on the use of a penalty function that tries to adapt the equivalent factor "s" to the current driving condition and cycle features, according to the SOC deviation computed on a periodic basis. The tuning of the parameter "s", in fact, is done dynamically, trying to achieve a convergent behavior of its value by scaling the penalty function with a coefficient called Kp, which must be accurately selected. The correct tuning of this controller, indeed, is affacted by the driving scenario but through a much less sensitive dependence from its design parameters enabling the controller usage in different conditions.

The control strategy still makes use of a point-by-point fuel consumption minimization but, in this case, its logic varies according to the value of the equivalent factor and, consequently, to the sampling of the evolution of the SOC over the simulation made each interval of time $T\_update$ , leading to a more sophisticated and versatile approach.

The specifications and the implications of this approach will be discussed throughout this report, illustrating the distinctive elements of the controller and highlighting the different behaviors of the ECMS control strategy within the proposed scenarios.

## 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 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
```

## Preliminar parameters

These values are common for all the cycles and are preliminary computed.

```
% Initialize the State of Charge

T_update = 60; % (s) Time Period of Update

eqFactor0 = 2.6406; % Calibrated Equivalent Factor
```

## Legend

1a: AUDC - Low Kp

1b: AUDC - Tuned Kp (later alias: Common Kp)

1c: AUDC - High Kp

2a: AMDC - Common Kp

2b: AMDC - Tuned Kp

2c: AMDC - Tuned KP with Improved Strategy

3a: WLTP - Common Kp

3b: WLTP - Tuned Kp

4a: TTC - Common Kp

4b: WLTP - Tuned Kp

### AUDC

#### Load cycle data

The project considers the AUDC, whose corresponding data are stored within the following directory.

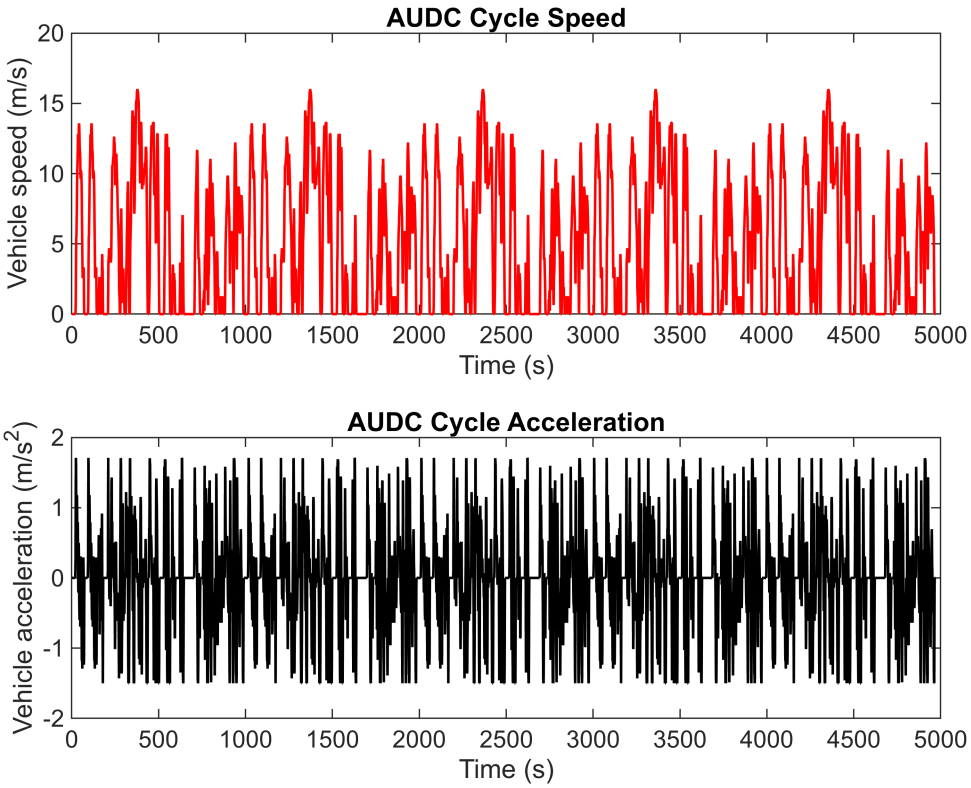
```
mission = load("data\AUDC.mat"); % ECMS data are now contained in the struct
mission variable
time1 = 1:1:5* length(mission.time_s); % (s)  Vector of the time instants within
the considered cycle repeted 5 times
vehSpd1 = []; % (m/s) Empty vector of the instant velocities referred to each time
value
vehAcc1 = []; % (m/s^2) Empty vector of the instant acceleration referred to each
time value
for i = 1:5 % Both the Speed and Acceleration vectors are filled up with the cycle
vector repeted 5 times
```

```
    vehSpd1 = [vehSpd1 mission.speed_km_h./ 3.6];
    vehAcc1 = [vehAcc1 mission.acceleration_m_s2];
end
```

### Plot the mission cycle data

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

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

nexttile(1) % Select the first tile (upper plot)
plot(time1, vehSpd1,"r-",'LineWidth', 1); % Plot vehicle speed over time
title("AUDC Cycle Speed");
xlabel("Time (s)");
ylabel("Vehicle speed (m/s)");

nexttile(2) % Select the second tile (lower plot)
plot(time1, vehAcc1,"k-", 'LineWidth', 1); % Plot vehicle acceleration over time
title("AUDC Cycle Acceleration");
xlabel("Time (s)");
ylabel("Vehicle acceleration (m/s^2)");
```



## Simulation Loop with Proportional SOC feedback - AUDC Cycle

Three simulations were carried out using different values of Kp, in order to obtain the following time-dependent outputs:

- State of Charge (SOC)
- Fuel flow rate
- Unfeasibility indicator
- Struct called "prof"

```
KpAUDC = [2 3.8 5]; % Vector containing the values of Kp used in the simulations
```

### Low Kp value Simulation

```
SOC1a = zeros(length(time1),1); % Create a vector to store SOC values over time

SOC1a(1) = 0.6; % Initial SOC value (60%)

eqFactor_n_1a = [eqFactor0]; % Initialize current equivalent factor

eqFactor_np = eqFactor0; % Initialize preceding equivalent factor

delta_SOC_1a = [0]; % Initialize vector to store SOC deviation

tc = 1; % Time counter for control update

for i = 1: length(time1)

    if i < length(time1)

        if tc > T_update

            % If it's time to update the control strategy
            tc = 1;
            % Compute SOC deviation from initial value
            delta_SOC_1a(end+1) = SOC1a(1) - SOC1a(i);

            % Recalculate control with updated equivalent factor
            [engSpd(i), engTrq(i), eqFactor_ns] = a_ecmsControl(SOC1a(i),
vehSpd1(i), vehAcc1(i), eqFactor_n_1a(end), eqFactor_np, KpAUDC(1),veh);
            % Simulate vehicle model with updated control
            [SOC1a(i+1), fuelFlwRate1a(i), unfeas1a(i), prof1a(i)] =
hev_model(SOC1a(i), [engSpd(i), engTrq(i)], [vehSpd1(i), vehAcc1(i)], veh);

            % Update equivalent factors
            eqFactor_np = eqFactor_n_1a(end);
            eqFactor_n_1a(end+1) = eqFactor_ns;
```

```
        else
            % If no control update is needed
            tc = tc + 1;
            % Keep the last SOC deviation
            delta_SOC_1a(end+1) = delta_SOC_1a(end);

            % Use previous equivalent factor for control
            [engSpd(i), engTrq(i)] = a_ecmsControl(SOC1a(i), vehSpd1(i),
vehAcc1(i), eqFactor_n_1a(end), eqFactor_np, KpAUDC(1), veh);
            [SOC1a(i+1), fuelFlwRate1a(i), unfeas1a(i), prof1a(i)] =
hev_model(SOC1a(i), [engSpd(i), engTrq(i)], [vehSpd1(i), vehAcc1(i)], veh);

            % Keep the current equivalent factor unchanged
            eqFactor_n_1a(end+1) = eqFactor_n_1a(end);
        end

    else
        % Handle last iteration separately to avoid out-of-bounds indexing
        [engSpd, engTrq] = a_ecmsControl(SOC1a(i), vehSpd1(i), vehAcc1(i),
eqFactor_n_1a(end), eqFactor_np, KpAUDC(1),veh);
        [SOC1a(i), fuelFlwRate1a(i), unfeas1a(i), prof1a(i)] = hev_model(SOC1a(i),
[engSpd, engTrq], [vehSpd1(i), vehAcc1(i)], veh);
    end
end
```

**Tuned Kp value Simulation**

```
SOC1b = zeros(length(time1),1); % Create a vector to store SOC values over time

SOC1b(1) = 0.6; % Initial SOC value (60%)

eqFactor_n_1b = [eqFactor0]; % Initialize current equivalent factor

eqFactor_np = eqFactor0;  % Initialize preceding equivalent factor

delta_SOC_1b = [0]; % Initialize vector to store SOC deviation

tc = 1; % Time counter for control update

for i = 1: length(time1)

    if i < length(time1)

        if tc > T_update

            % If it's time to update the control strategy
            tc = 1;
            % Compute SOC deviation from initial value
```

```
            delta_SOC_1b(end+1) = SOC1b(1) - SOC1b(i);

            % Recalculate control with updated equivalent factor
            [engSpd(i), engTrq(i), eqFactor_ns] = a_ecmsControl(SOC1b(i),
vehSpd1(i), vehAcc1(i), eqFactor_n_1b(end), eqFactor_np, KpAUDC(2),veh);
            % Simulate vehicle model with updated control
            [SOC1b(i+1), fuelFlwRate1b(i), unfeas1b(i), prof1b(i)] =
hev_model(SOC1b(i), [engSpd(i), engTrq(i)], [vehSpd1(i), vehAcc1(i)], veh);

            % Update equivalent factors
            eqFactor_np = eqFactor_n_1b(end);
            eqFactor_n_1b(end+1) = eqFactor_ns;

        else
            % If no control update is needed
            tc = tc + 1;

            % Keep the last SOC deviation
            delta_SOC_1b(end+1) = delta_SOC_1b(end);

            % Use previous equivalent factor for control
            [engSpd(i), engTrq(i)] = a_ecmsControl(SOC1b(i), vehSpd1(i),
vehAcc1(i), eqFactor_n_1b(end), eqFactor_np, KpAUDC(2), veh);
            [SOC1b(i+1), fuelFlwRate1b(i), unfeas1b(i), prof1b(i)] =
hev_model(SOC1b(i), [engSpd(i), engTrq(i)], [vehSpd1(i), vehAcc1(i)], veh);

            % Keep the current equivalent factor unchanged
            eqFactor_n_1b(end+1) = eqFactor_n_1b(end);
        end

    else
        % Handle last iteration separately to avoid out-of-bounds indexing
        [engSpd, engTrq] = a_ecmsControl(SOC1b(i), vehSpd1(i), vehAcc1(i),
eqFactor_n_1b(end), eqFactor_np, KpAUDC(2),veh);
        [SOC1b(i), fuelFlwRate1b(i), unfeas1b(i), prof1b(i)] = hev_model(SOC1b(i),
[engSpd, engTrq], [vehSpd1(i), vehAcc1(i)], veh);
    end
end
```

**High Kp value Simulation**

```
SOC1c = zeros(length(time1),1); % Create a vector to store SOC values over time

SOC1c(1) = 0.6; % Initial SOC value (60%)

eqFactor_n_1c = [eqFactor0]; % Initialize current equivalent factor

eqFactor_np = eqFactor0;  % Initialize preceding equivalent factor
```

```
delta_SOC_1c = [0]; % Initialize vector to store SOC deviation

tc = 1; % Time counter for control update

for i = 1: length(time1)

    if i < length(time1)

        if tc > T_update

            % If it's time to update the control strategy
            tc = 1;
            % Compute SOC deviation from initial value
            delta_SOC_1c(end+1) = SOC1c(1) - SOC1c(i);

            % Recalculate control with updated equivalent factor
            [engSpd(i), engTrq(i), eqFactor_ns] = a_ecmsControl(SOC1c(i),
vehSpd1(i), vehAcc1(i), eqFactor_n_1c(end), eqFactor_np, KpAUDC(3),veh);
            % Simulate vehicle model with updated control
            [SOC1c(i+1), fuelFlwRate1c(i), unfeas1c(i), prof1c(i)] =
hev_model(SOC1c(i), [engSpd(i), engTrq(i)], [vehSpd1(i), vehAcc1(i)], veh);

            % Update equivalent factors
            eqFactor_np = eqFactor_n_1c(end);
            eqFactor_n_1c(end+1) = eqFactor_ns;

        else
            % If no control update is needed
            tc = tc + 1;
            % Keep the last SOC deviation
            delta_SOC_1c(end+1) = delta_SOC_1c(end);

            % Use previous equivalent factor for control
            [engSpd(i), engTrq(i)] = a_ecmsControl(SOC1b(i), vehSpd1(i),
vehAcc1(i), eqFactor_n_1c(end), eqFactor_np, KpAUDC(3), veh);
            [SOC1c(i+1), fuelFlwRate1c(i), unfeas1c(i), prof1c(i)] =
hev_model(SOC1c(i), [engSpd(i), engTrq(i)], [vehSpd1(i), vehAcc1(i)], veh);

            % Keep the current equivalent factor unchanged
            eqFactor_n_1c(end+1) = eqFactor_n_1c(end);
        end

    else
        % Handle last iteration separately to avoid out-of-bounds indexing
        [engSpd, engTrq] = a_ecmsControl(SOC1c(i), vehSpd1(i), vehAcc1(i),
eqFactor_n_1c(end), eqFactor_np, KpAUDC(3),veh);
        [SOC1c(i), fuelFlwRate1c(i), unfeas1c(i), prof1c(i)] = hev_model(SOC1c(i),
[engSpd, engTrq], [vehSpd1(i), vehAcc1(i)], veh);
    end
end
```

## AMDC

### Load cycle data

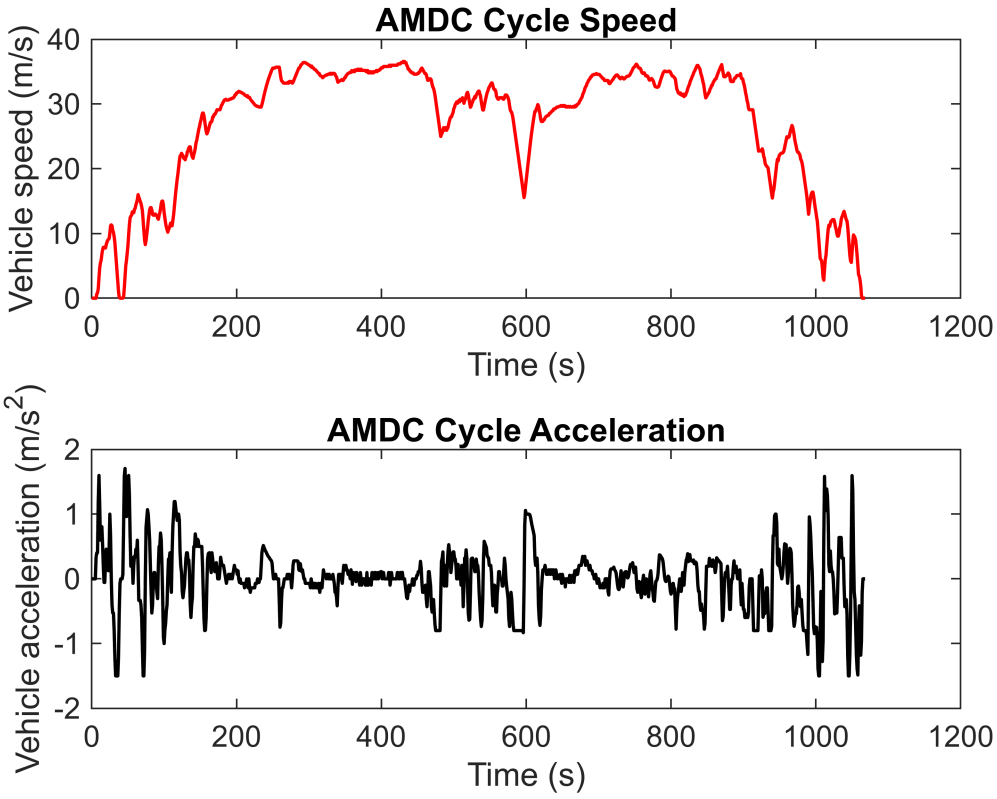
The project considers the AMDC, whose corresponding data are stored within the following directory.

```
mission = load("data\AMDC.mat"); % ECMS data are now contained in the struct
mission variable
time2 = mission.time_s; % (s)  Vector of the time instants within the considered
cycle
vehSpd2 = mission.speed_km_h ./ 3.6; % (m/s) Vector of the instant velocities
referred to each time value
vehAcc2 = 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 AUDC are now being plotted as a function of time.

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

nexttile(1) % Select the first tile (upper plot)
plot(time2, vehSpd2,"r-",'LineWidth', 1); % Plot vehicle speed over time
title("AMDC Cycle Speed");
xlabel("Time (s)");
ylabel("Vehicle speed (m/s)");

nexttile(2) % Select the second tile (lower plot)
plot(time2, vehAcc2,"k-", 'LineWidth', 1); % Plot vehicle acceleration over time
title("AMDC Cycle Acceleration");
xlabel("Time (s)");
ylabel("Vehicle acceleration (m/s^2)");
```




## Simulation Loop with Proportional SOC feedback - AMDC Cycle

Three simulations were carried out using different values of Kp, in order to obtain the following time-dependent outputs:

- State of Charge (SOC)
- Fuel flow rate
- Unfeasibility indicator
- Struct called "prof"

```
KpAMDC = [3.8 0.1 5]; % Vector containing the values of Kp used in the simulations
```

### Common Kp value Simulation

```
SOC2a = zeros(length(time2),1); % Create a vector to store SOC values over time

SOC2a(1) = 0.6; % Initial SOC value (60%)

eqFactor_n_2a = [eqFactor0]; % Initialize current equivalent factor

eqFactor_np = eqFactor0; % Initialize preceding equivalent factor
```

```
delta_SOC_2a = [0]; % Initialize vector to store SOC deviation

tc = 1; % Time counter for control update

for i = 1: length(time2)

    if i < length(time2)

        if tc > T_update

            % If it's time to update the control strategy
            tc = 1;
            % Compute SOC deviation from initial value
            delta_SOC_2a(end+1) = SOC2a(1) - SOC2a(i);

            % Recalculate control with updated equivalent factor
            [engSpd(i), engTrq(i), eqFactor_ns] = a_ecmsControl(SOC2a(i),
vehSpd2(i), vehAcc2(i), eqFactor_n_2a(end), eqFactor_np, KpAMDC(1),veh);
            % Simulate vehicle model with updated control
            [SOC2a(i+1), fuelFlwRate2a(i), unfeas2a(i), prof2a(i)] =
hev_model(SOC2a(i), [engSpd(i), engTrq(i)], [vehSpd2(i), vehAcc2(i)], veh);

            % Update equivalent factors
            eqFactor_np = eqFactor_n_2a(end);
            eqFactor_n_2a(end+1) = eqFactor_ns;

        else
            % If no control update is needed
            tc = tc + 1;
            % Keep the last SOC deviation
            delta_SOC_2a(end+1) = delta_SOC_2a(end);

            % Use previous equivalent factor for control
            [engSpd(i), engTrq(i)] = a_ecmsControl(SOC2a(i), vehSpd2(i),
vehAcc2(i), eqFactor_n_2a(end), eqFactor_np, KpAMDC(1), veh);
            [SOC2a(i+1), fuelFlwRate2a(i), unfeas2a(i), prof2a(i)] =
hev_model(SOC2a(i), [engSpd(i), engTrq(i)], [vehSpd2(i), vehAcc2(i)], veh);

            % Keep the current equivalent factor unchanged
            eqFactor_n_2a(end+1) = eqFactor_n_2a(end);
        end

    else
        % Handle last iteration separately to avoid out-of-bounds indexing
        [engSpd, engTrq] = a_ecmsControl(SOC2a(i), vehSpd2(i), vehAcc2(i),
eqFactor_n_2a(end), eqFactor_np, KpAMDC(1),veh);
        [SOC2a(i), fuelFlwRate2a(i), unfeas2a(i), prof2a(i)] = hev_model(SOC2a(i),
[engSpd, engTrq], [vehSpd2(i), vehAcc2(i)], veh);
    end
```

```
end
```

### Tuned Kp value Simulation

```
SOC2b = zeros(length(time2),1); % Create a vector to store SOC values over time

SOC2b(1) = 0.6; % Initial SOC value (60%)

eqFactor_n_2b = [eqFactor0]; % Initialize current equivalent factor

eqFactor_np = eqFactor0;  % Initialize preceding equivalent factor

delta_SOC_2b = [0]; % Initialize vector to store SOC deviation

tc = 1; % Time counter for control update

for i = 1: length(time2)

    if i < length(time2)

        if tc > T_update

            % If it's time to update the control strategy
            tc = 1;
            % Compute SOC deviation from initial value
            delta_SOC_2b(end+1) = SOC2b(1) - SOC2b(i);

            % Recalculate control with updated equivalent factor
            [engSpd(i), engTrq(i), eqFactor_ns] = a_ecmsControl(SOC2b(i),
vehSpd2(i), vehAcc2(i), eqFactor_n_2b(end), eqFactor_np, KpAMDC(2),veh);
            % Simulate vehicle model with updated control
            [SOC2b(i+1), fuelFlwRate2b(i), unfeas2b(i), prof2b(i)] =
hev_model(SOC2b(i), [engSpd(i), engTrq(i)], [vehSpd2(i), vehAcc2(i)], veh);

            % Update equivalent factors
            eqFactor_np = eqFactor_n_2b(end);
            eqFactor_n_2b(end+1) = eqFactor_ns;

        else
            % If no control update is needed
            tc = tc + 1;
            % Keep the last SOC deviation
            delta_SOC_2b(end+1) = delta_SOC_2b(end);

            % Use previous equivalent factor for control
            [engSpd(i), engTrq(i)] = a_ecmsControl(SOC2b(i), vehSpd2(i),
vehAcc2(i), eqFactor_n_2b(end), eqFactor_np, KpAMDC(2), veh);
            [SOC2b(i+1), fuelFlwRate2b(i), unfeas2b(i), prof2b(i)] =
hev_model(SOC2b(i), [engSpd(i), engTrq(i)], [vehSpd2(i), vehAcc2(i)], veh);
```

```
            % Keep the current equivalent factor unchanged
            eqFactor_n_2b(end+1) = eqFactor_n_2b(end);
        end

    else
        % Handle last iteration separately to avoid out-of-bounds indexing
        [engSpd, engTrq] = a_ecmsControl(SOC2b(i), vehSpd2(i), vehAcc2(i),
eqFactor_n_2b(end), eqFactor_np, KpAMDC(2),veh);
        [SOC2b(i), fuelFlwRate2b(i), unfeas2b(i), prof2b(i)] = hev_model(SOC2b(i),
[engSpd, engTrq], [vehSpd2(i), vehAcc2(i)], veh);
    end
end
```

**Tuned Kp with Improved controller Simulation**

```
SOC2c = zeros(length(time2),1); % Create a vector to store SOC values over time

SOC2c(1) = 0.6; % Initial SOC value (60%)

eqFactor_n_2c = [eqFactor0]; % Initialize current equivalent factor

eqFactor_np = eqFactor0;  % Initialize preceding equivalent factor

delta_SOC_2c = [0]; % Initialize vector to store SOC deviation

tc = 1; % Time counter for control update

for i = 1: length(time2)

    if i < length(time2)

        if tc > T_update

            % If it's time to update the control strategy
            tc = 1;
            % Compute SOC deviation from initial value
            delta_SOC_2c(end+1) = SOC2c(1) - SOC2c(i);

            % Recalculate control with updated equivalent factor
            [engSpd(i), engTrq(i), eqFactor_ns] = improved_a_ecmsControl(SOC2c(i),
vehSpd2(i), vehAcc2(i), eqFactor_n_2c(end), eqFactor_np, KpAMDC(3),veh);
            % Simulate vehicle model with updated control
            [SOC2c(i+1), fuelFlwRate2c(i), unfeas2c(i), prof2c(i)] =
hev_model(SOC2c(i), [engSpd(i), engTrq(i)], [vehSpd2(i), vehAcc2(i)], veh);

            % Update equivalent factors
            eqFactor_np = eqFactor_n_2c(end);
            eqFactor_n_2c(end+1) = eqFactor_ns;
```

```
        else
            % If no control update is needed
            tc = tc + 1;
            % Keep the last SOC deviation
            delta_SOC_2c(end+1) = delta_SOC_2c(end);

            % Use previous equivalent factor for control
            [engSpd(i), engTrq(i)] = improved_a_ecmsControl(SOC2c(i), vehSpd2(i),
vehAcc2(i), eqFactor_n_2c(end), eqFactor_np, KpAMDC(3), veh);
            [SOC2c(i+1), fuelFlwRate2c(i), unfeas2c(i), prof2c(i)] =
hev_model(SOC2c(i), [engSpd(i), engTrq(i)], [vehSpd2(i), vehAcc2(i)], veh);

            % Keep the current equivalent factor unchanged
            eqFactor_n_2c(end+1) = eqFactor_n_2c(end);
        end

    else
        % Handle last iteration separately to avoid out-of-bounds indexing
        [engSpd, engTrq] = improved_a_ecmsControl(SOC2c(i), vehSpd2(i), vehAcc2(i),
eqFactor_n_2c(end), eqFactor_np, KpAMDC(3),veh);
        [SOC2c(i), fuelFlwRate2c(i), unfeas2c(i), prof2c(i)] = hev_model(SOC2c(i),
[engSpd, engTrq], [vehSpd2(i), vehAcc2(i)], veh);
    end
end
```

## WLTP

### Load cycle data

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

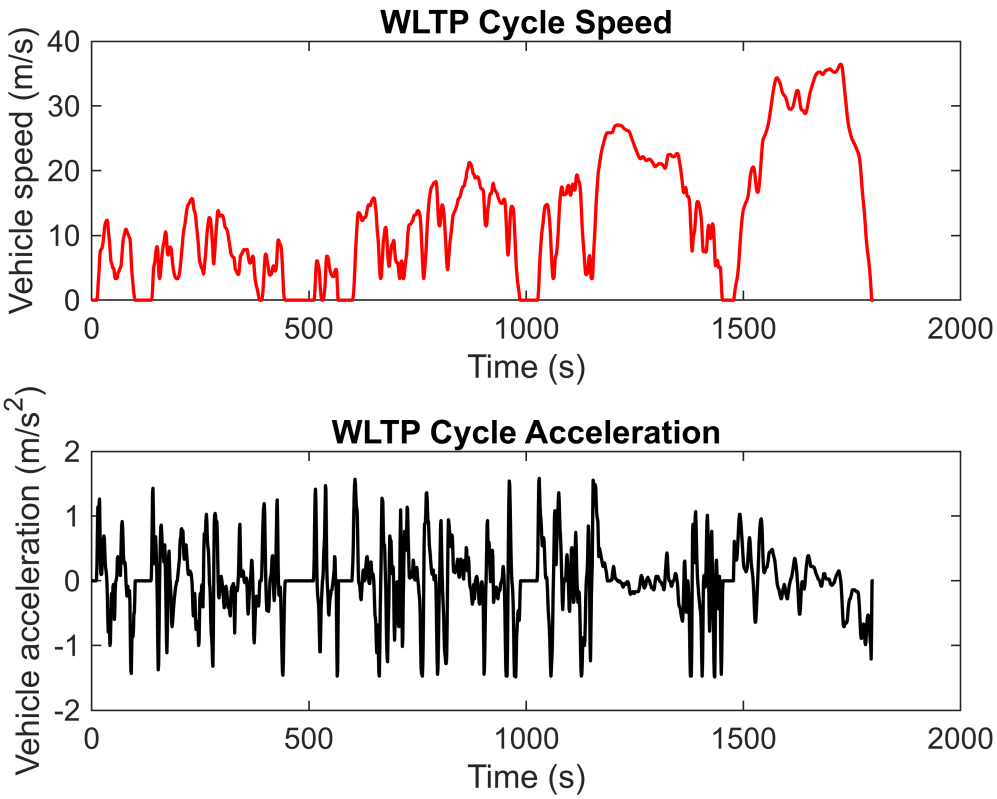
```
mission = load("data\WLTP.mat"); % WLTP data are now contained in the struct
mission variable
time3 = mission.time_s; % (s)  Vector of the time instants within the considered
cycle
vehSpd3 = mission.speed_km_h ./ 3.6; % (m/s) Vector of the instant velocities
referred to each time value
vehAcc3 = 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 are now being plotted as a function of time.

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

```
nexttile(1) % Select the first tile (upper plot)
plot(time3, vehSpd3,"r-",'LineWidth', 1); % Plot vehicle speed over time
title("WLTP Cycle Speed");
xlabel("Time (s)");
ylabel("Vehicle speed (m/s)");

nexttile(2) % Select the second tile (lower plot)
plot(time3, vehAcc3,"k-", 'LineWidth', 1); % Plot vehicle acceleration over time
title("WLTP Cycle Acceleration");
xlabel("Time (s)");
ylabel("Vehicle acceleration (m/s^2)");
```




## Simulation Loop with Proportional SOC feedback - WLTP Cycle

Two simulations were carried out using different values of Kp, in order to obtain the following time-dependent outputs:

- State of Charge (SOC)
- Fuel flow rate
- Unfeasibility indicator
- Struct called "`prof`"

```
KpWLTP = [3.8 1]; % Vector containing the values of Kp used in the simulations
```

**Common Kp value Simulation**

```
SOC3a = zeros(length(time3),1); % Create a vector to store SOC values over time

SOC3a(1) = 0.6; % Initial SOC value (60%)

eqFactor_n_3a = [eqFactor0]; % Initialize current equivalent factor

eqFactor_np = eqFactor0;  % Initialize preceding equivalent factor

delta_SOC_3a = [0]; % Initialize vector to store SOC deviation

tc = 1; % Time counter for control update

for i = 1: length(time3)

    if i < length(time3)

        if tc > T_update

            % If it's time to update the control strategy
            tc = 1;
            % Compute SOC deviation from initial value
            delta_SOC_3a(end+1) = SOC3a(1) - SOC3a(i);

            % Recalculate control with updated equivalent factor
            [engSpd(i), engTrq(i), eqFactor_ns] = a_ecmsControl(SOC3a(i),
vehSpd3(i), vehAcc3(i), eqFactor_n_3a(end), eqFactor_np, KpWLTP(1),veh);
            % Simulate vehicle model with updated control
            [SOC3a(i+1), fuelFlwRate3a(i), unfeas3a(i), prof3a(i)] =
hev_model(SOC3a(i), [engSpd(i), engTrq(i)], [vehSpd3(i), vehAcc3(i)], veh);

            % Update equivalent factors
            eqFactor_np = eqFactor_n_3a(end);
            eqFactor_n_3a(end+1) = eqFactor_ns;

        else
            % If no control update is needed
            tc = tc + 1;
            % Keep the last SOC deviation
            delta_SOC_3a(end+1) = delta_SOC_3a(end);

            % Use previous equivalent factor for control
            [engSpd(i), engTrq(i)] = a_ecmsControl(SOC3a(i), vehSpd3(i),
vehAcc3(i), eqFactor_n_3a(end), eqFactor_np, KpWLTP(1), veh);
            [SOC3a(i+1), fuelFlwRate3a(i), unfeas3a(i), prof3a(i)] =
hev_model(SOC3a(i), [engSpd(i), engTrq(i)], [vehSpd3(i), vehAcc3(i)], veh);
```

```
            % Keep the current equivalent factor unchanged
            eqFactor_n_3a(end+1) = eqFactor_n_3a(end);
        end

    else
        % Handle last iteration separately to avoid out-of-bounds indexing
        [engSpd, engTrq] = a_ecmsControl(SOC3a(i), vehSpd3(i), vehAcc3(i),
eqFactor_n_3a(end), eqFactor_np, KpWLTP(1),veh);
        [SOC3a(i), fuelFlwRate3a(i), unfeas3a(i), prof3a(i)] = hev_model(SOC3a(i),
[engSpd, engTrq], [vehSpd3(i), vehAcc3(i)], veh);
    end
end
```

**Tuned Kp value Simulation**

```
SOC3b = zeros(length(time3),1); % Create a vector to store SOC values over time

SOC3b(1) = 0.6; % Initial SOC value (60%)

eqFactor_n_3b = [eqFactor0]; % Initialize current equivalent factor

eqFactor_np = eqFactor0;  % Initialize preceding equivalent factor

delta_SOC_3b = [0]; % Initialize vector to store SOC deviation

tc = 1; % Time counter for control update

for i = 1: length(time3)

    if i < length(time3)

        if tc > T_update

            % If it's time to update the control strategy
            tc = 1;
            % Compute SOC deviation from initial value
            delta_SOC_3b(end+1) = SOC3b(1) - SOC3b(i);

            % Recalculate control with updated equivalent factor
            [engSpd(i), engTrq(i), eqFactor_ns] = a_ecmsControl(SOC3b(i),
vehSpd3(i), vehAcc3(i), eqFactor_n_3b(end), eqFactor_np, KpWLTP(2),veh);
            % Simulate vehicle model with updated control
            [SOC3b(i+1), fuelFlwRate3b(i), unfeas3b(i), prof3b(i)] =
hev_model(SOC3b(i), [engSpd(i), engTrq(i)], [vehSpd3(i), vehAcc3(i)], veh);

            % Update equivalent factors
            eqFactor_np = eqFactor_n_3b(end);
            eqFactor_n_3b(end+1) = eqFactor_ns;
```

```
        else
            % If no control update is needed
            tc = tc + 1;
            % Keep the last SOC deviation
            delta_SOC_3b(end+1) = delta_SOC_3b(end);

            % Use previous equivalent factor for control
            [engSpd(i), engTrq(i)] = a_ecmsControl(SOC3b(i), vehSpd3(i),
vehAcc3(i), eqFactor_n_3b(end), eqFactor_np, KpWLTP(2), veh);
            [SOC3b(i+1), fuelFlwRate3b(i), unfeas3b(i), prof3b(i)] =
hev_model(SOC3b(i), [engSpd(i), engTrq(i)], [vehSpd3(i), vehAcc3(i)], veh);

            % Keep the current equivalent factor unchanged
            eqFactor_n_3b(end+1) = eqFactor_n_3b(end);
        end

    else
        % Handle last iteration separately to avoid out-of-bounds indexing
        [engSpd, engTrq] = a_ecmsControl(SOC3b(i), vehSpd3(i), vehAcc3(i),
eqFactor_n_3b(end), eqFactor_np, KpWLTP(2),veh);
        [SOC3b(i), fuelFlwRate3b(i), unfeas3b(i), prof3b(i)] = hev_model(SOC3b(i),
[engSpd, engTrq], [vehSpd3(i), vehAcc3(i)], veh);
    end
end
```

## TURIN TEST CYCLE

### Load cycle data

The project considers the TTC, whose corresponding data are stored within the following directory.

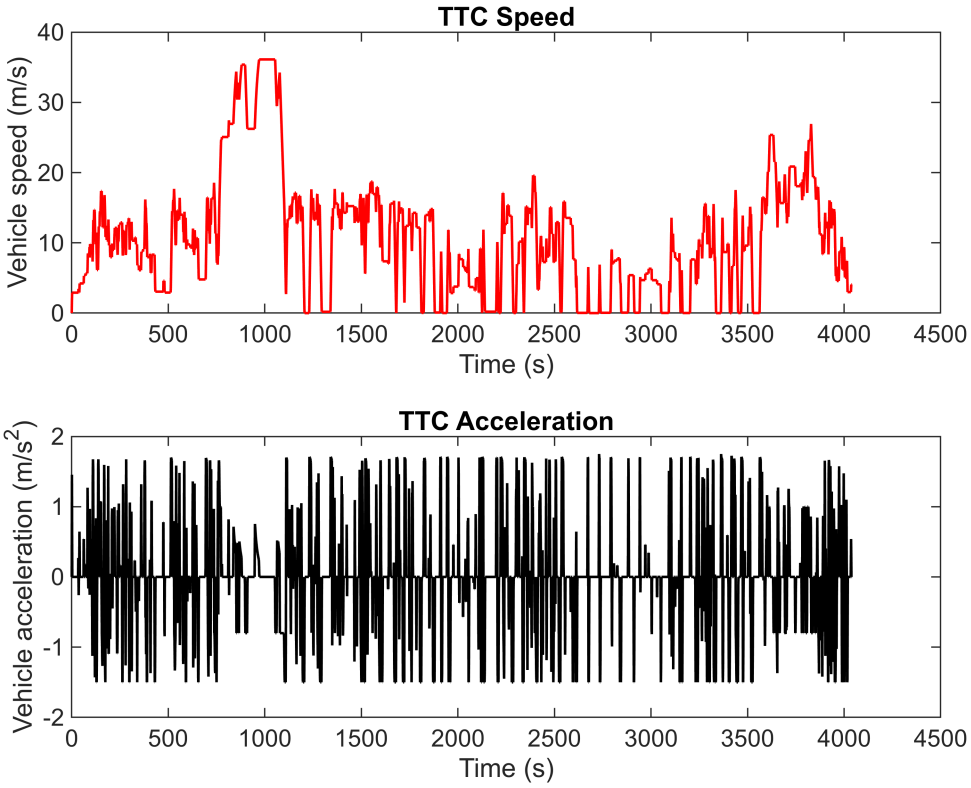
```
mission = load("data\TurinTest.mat"); % ECMS data are now contained in the struct
mission variable
time4 = mission.time_s; % (s)  Vector of the time instants within the considered
cycle
vehSpd4 = mission.speed_km_h ./ 3.6; % (m/s) Vector of the instant velocities
referred to each time value
vehAcc4 = 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 Turin Test Cycle are now being plotted as a function of time.

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

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

```
plot(time4, vehSpd4,"r-",'LineWidth', 1); % Plot vehicle speed 4 over time
title("TTC Speed");
xlabel("Time (s)");
ylabel("Vehicle speed (m/s)");

nexttile(2) % Select the second tile (lower plot)
plot(time4, vehAcc4,"k-", 'LineWidth', 1); % Plot vehicle acceleration 4 over time
title("TTC Acceleration");
xlabel("Time (s)");
ylabel("Vehicle acceleration (m/s^2)");
```




## Simulation Loop with Proportional SOC feedback - Turin Test Cycle

Two simulations were carried out using different values of Kp, in order to obtain the following time-dependent outputs:

- State of Charge (SOC)
- Fuel flow rate
- Unfeasibility indicator
- Struct called "`prof`"

```
KpTTC = [3.8 9]; % Vector containing the values of Kp used in the simulations
```

**Common Kp value Simulation**

```
SOC4a = zeros(length(time4),1); % Create a vector to store SOC values over time

SOC4a(1) = 0.6; % Initial SOC value (60%)

eqFactor_n_4a = [eqFactor0]; % Initialize current equivalent factor

eqFactor_np = eqFactor0;  % Initialize preceding equivalent factor

delta_SOC_4a = [0]; % Initialize vector to store SOC deviation

tc = 1; % Time counter for control update

for i = 1: length(time4)

    if i < length(time4)

        if tc > T_update

            % If it's time to update the control strategy
            tc = 1;
            % Compute SOC deviation from initial value
            delta_SOC_4a(end+1) = SOC4a(1) - SOC4a(i);

            % Recalculate control with updated equivalent factor
            [engSpd(i), engTrq(i), eqFactor_ns] = a_ecmsControl(SOC4a(i),
vehSpd4(i), vehAcc4(i), eqFactor_n_4a(end), eqFactor_np, KpTTC(1),veh);
            % Simulate vehicle model with updated control
            [SOC4a(i+1), fuelFlwRate4a(i), unfeas4a(i), prof4a(i)] =
hev_model(SOC4a(i), [engSpd(i), engTrq(i)], [vehSpd4(i), vehAcc4(i)], veh);

            % Update equivalent factors
            eqFactor_np = eqFactor_n_4a(end);
            eqFactor_n_4a(end+1) = eqFactor_ns;

        else
            % If no control update is needed
            tc = tc + 1;
            % Keep the last SOC deviation
            delta_SOC_4a(end+1) = delta_SOC_4a(end);

            % Use previous equivalent factor for control
            [engSpd(i), engTrq(i)] = a_ecmsControl(SOC4a(i), vehSpd4(i),
vehAcc4(i), eqFactor_n_4a(end), eqFactor_np, KpTTC(1), veh);
            [SOC4a(i+1), fuelFlwRate4a(i), unfeas4a(i), prof4a(i)] =
hev_model(SOC4a(i), [engSpd(i), engTrq(i)], [vehSpd4(i), vehAcc4(i)], veh);
```

```
            % Keep the current equivalent factor unchanged
            eqFactor_n_4a(end+1) = eqFactor_n_4a(end);
        end

    else
        % Handle last iteration separately to avoid out-of-bounds indexing
        [engSpd, engTrq] = a_ecmsControl(SOC4a(i), vehSpd4(i), vehAcc4(i),
eqFactor_n_4a(end), eqFactor_np, KpTTC(1),veh);
        [SOC4a(i), fuelFlwRate4a(i), unfeas4a(i), prof4a(i)] = hev_model(SOC4a(i),
[engSpd, engTrq], [vehSpd4(i), vehAcc4(i)], veh);
    end
end
```

**Tuned Kp value Simulation**

```
SOC4b = zeros(length(time4),1); % Create a vector to store SOC values over time

SOC4b(1) = 0.6; % Initial SOC value (60%)

eqFactor_n_4b = [eqFactor0]; % Initialize current equivalent factor

eqFactor_np = eqFactor0;  % Initialize preceding equivalent factor

delta_SOC_4b = [0]; % Initialize vector to store SOC deviation

tc = 1; % Time counter for control update

for i = 1: length(time4)

    if i < length(time4)

        if tc > T_update

            % If it's time to update the control strategy
            tc = 1;
            % Compute SOC deviation from initial value
            delta_SOC_4b(end+1) = SOC4b(1) - SOC4b(i);

            % Recalculate control with updated equivalent factor
            [engSpd(i), engTrq(i), eqFactor_ns] = a_ecmsControl(SOC4b(i),
vehSpd4(i), vehAcc4(i), eqFactor_n_4b(end), eqFactor_np, KpTTC(2),veh);
            % Simulate vehicle model with updated control
            [SOC4b(i+1), fuelFlwRate4b(i), unfeas4b(i), prof4b(i)] =
hev_model(SOC4b(i), [engSpd(i), engTrq(i)], [vehSpd4(i), vehAcc4(i)], veh);

            % Update equivalent factors
            eqFactor_np = eqFactor_n_4b(end);
            eqFactor_n_4b(end+1) = eqFactor_ns;
```

```
        else
            % If no control update is needed
            tc = tc + 1;
            % Keep the last SOC deviation
            delta_SOC_4b(end+1) = delta_SOC_4b(end);

            % Use previous equivalent factor for control
            [engSpd(i), engTrq(i)] = a_ecmsControl(SOC4b(i), vehSpd4(i),
vehAcc4(i), eqFactor_n_4b(end), eqFactor_np, KpTTC(2), veh);
            [SOC4b(i+1), fuelFlwRate4b(i), unfeas4b(i), prof4b(i)] =
hev_model(SOC4b(i), [engSpd(i), engTrq(i)], [vehSpd4(i), vehAcc4(i)], veh);

            % Keep the current equivalent factor unchanged
            eqFactor_n_4b(end+1) = eqFactor_n_4b(end);
        end

    else
        % Handle last iteration separately to avoid out-of-bounds indexing
        [engSpd, engTrq] = a_ecmsControl(SOC4b(i), vehSpd4(i), vehAcc4(i),
eqFactor_n_4b(end), eqFactor_np, KpTTC(2),veh);
        [SOC4b(i), fuelFlwRate4b(i), unfeas4b(i), prof4b(i)] = hev_model(SOC4b(i),
[engSpd, engTrq], [vehSpd4(i), vehAcc4(i)], veh);
    end
end
```

# POST PROCESSING

**State of Charge (SOC) evolution**

The plot illustrates the evolution of the battery State of Charge (SOC) over time during an AUDC (Artemis Urban Driving Cycle) for a hybrid electric vehicle. Three different lines are shown, each corresponding to a different coefficient called `Kp` used in the feedback controller that adjusts the equivalent fuel consumption factor in the ECMS.

Initial SOC condition: All simulations start with the same initial SOC of 60%, providing a common baseline for comparison.

- Low Kp (Kp = 2)**:**The SOC shows a gradual and less oscillatory trend over time. The system reacts more slowly to changes in driving conditions, which indicates a lower control aggressiveness. It can be interpreted that the feedback control is not aggressive enough to maintain the SOC close to its initial value.
- Medium Kp (Kp = 3.8)**:** With a tuned Kp value, the SOC remains more stable and closer to the initial level throughout the cycle. This suggests a better trade-off between fuel consumption and battery usage. The control system successfully corrects deviations in SOC more efficiently compared to the low Kp case, minimizing cumulative error without overreacting.

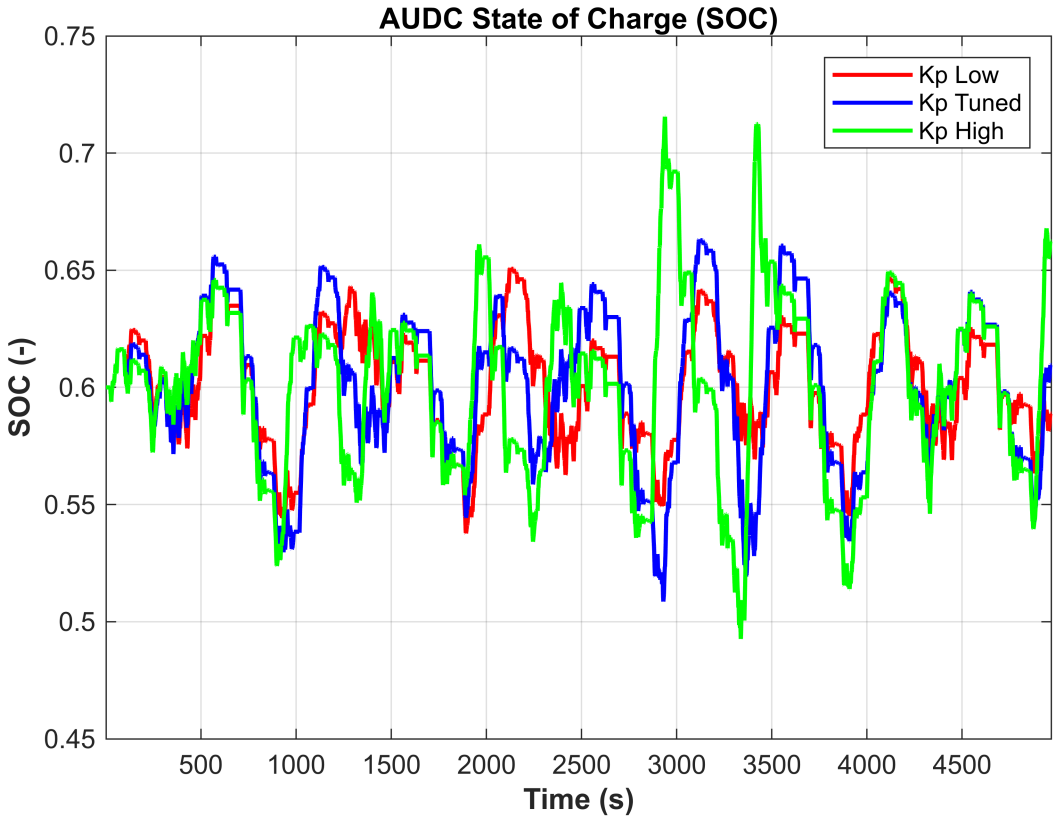
- High Kp (Kp = 5)**:** In the high Kp scenario, the SOC trajectory shows rapid corrections and tends to oscillate slightly due to the controller's increased sensitivity. Although it brings the SOC back to the initial target more aggressively, this might introduce inefficiencies or reduce smoothness in real-time operation.

The State of Charge (SOC) profile exhibits a behavior that is much more consistent with an optimal energy management strategy: the SOC does not remain fixed near a boundary nor does it increase excessively, but instead oscillates in a controlled manner around the target value. This indicates a more balanced and responsive distribution of power between the internal combustion engine and the battery.

The value of Kp is not overly sensitive, which allows it to deliver good results even with different Kp settings. This highlights the controller's robustness in this particular cycle, which is significantly longer than the others, in addittion to that, the controller manages to ensure a fairly rapid dynamic response.

To sum up, this plot demonstrates the impact of proportional gain tuning on energy management. A well-tuned Kp helps maintain SOC stability, ensuring the vehicle operates within optimal energy boundaries. Too low a gain leads to poor regulation, while too high a gain may cause overcorrection. The medium Kp scenario appears to strike the best balance for this specific driving cycle, the AUDC.

```
%State of Charge AUDC
figure;
t = tiledlayout(1,1,'TileSpacing','Compact','Padding','Compact');
ax1 = nexttile;
plot(ax1, time1, SOC1a, "r-", 'LineWidth', 1.5)
grid on
hold on
plot(ax1, time1, SOC1b, "b-", 'LineWidth', 1.5)
hold on
plot(ax1, time1, SOC1c, "g-", 'LineWidth', 1.5)
title("AUDC State of Charge (SOC)", 'FontWeight', 'bold');
legend("Kp Low", "Kp Tuned", "Kp High")
xlabel("Time (s)", 'FontWeight','bold');
ylabel("SOC (-)", 'FontWeight','bold');
xlim([time1(1) time1(end)])
```




This plot presents the State of Charge (SOC) evolution during the Artemis Motorway Driving Cycle (AMDC) for two different proportional feedback gains:

- A common Kp = 3.8, originally tuned for the urban AUDC cycle.
- A tuning attemp with Kp = 0.1,  consdering the AMDC conditions.
- A tuned Kp = 5,  specifically optimized for AMDC using the improved controller.

All simulations start from an initial SOC of 60%.

Common Kp = 3.8 (Not Tuned for AMDC):

The SOC trajectory shows a clear and continuous upward trend. This indicates that the battery is being charged excessively during the motorway cycle. Since this Kp value was tuned for the more dynamic urban cycle (AUDC), where frequent accelerations and decelerations occur, it leads to an overreaction in the relatively steady high-speed AMDC conditions. The control system becomes too conservative, relying more on the engine and underusing the electric drive. As a result, energy from braking or low-load operation is not efficiently used, leading to unnecessary SOC accumulation.

Tuned Kp = 0.1 (Optimization attemp for AMDC):

With a lower Kp, the controller reacts more smoothly and slowly to SOC changes. This is more suitable for the AMDC, which involves fewer stop-and-go events and longer cruising phases. The SOC remains closer to its initial value throughout the cycle, indicating better energy balance. The battery is used more effectively, and

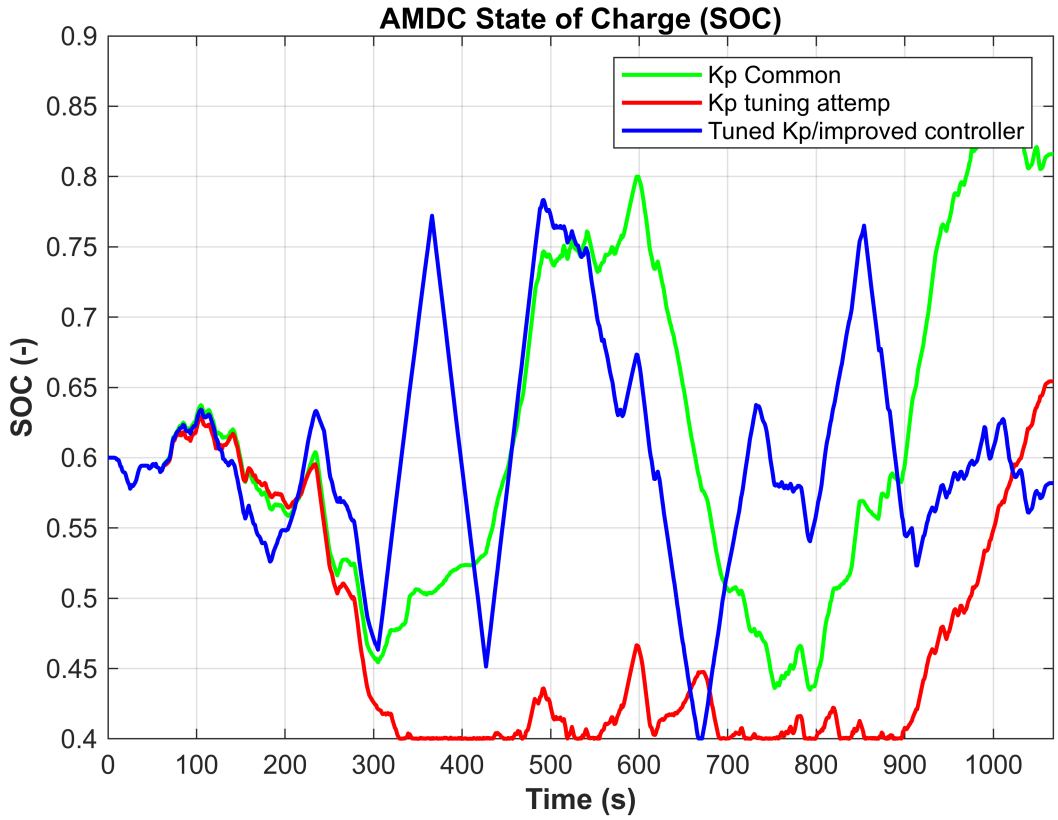
the electric motor assists in a more controlled way, avoiding excessive charging. This results in a more efficient power split between the engine and electric components.

In the previous AUDC simulation, Kp = 3.8 performed well by stabilizing the SOC due to frequent speed changes. However, in the AMDC, this same Kp causes overcharging. This demonstrates that controller tuning must be cycle-specific. What works well in urban driving does not necessarily perform efficiently on the highway, where dynamics are smoother and energy recovery opportunities are different.

Tuned Kp = 5 (improved controller):

The State of Charge (SOC) profile exhibits a behavior that is much more consistent with an optimal energy management strategy: the SOC does not remain fixed near a boundary nor does it increase excessively, but instead oscillates in an acceptable manner around the target value. This indicates a more balanced and responsive distribution of power between the internal combustion engine and the battery. This result demonstrates that the use of the advanced controller enables a more accurate regulation of engine activation and battery recharging phases, in addition to the achivement of the charge sustaining behaviour.

```
%State of Charge AMDC
figure;
t = tiledlayout(1,1,'TileSpacing','Compact','Padding','Compact');
ax1 = nexttile;
plot(ax1,time2, SOC2a, "g-", 'LineWidth', 1.5)
legend("Kp Common")
grid on
hold on
plot(ax1, time2, SOC2b, "r-", 'LineWidth', 1.5)
hold on
plot(ax1, time2, SOC2c, "b-", 'LineWidth', 1.5)
title("AMDC State of Charge (SOC)", 'FontWeight', 'bold');
legend("Kp Common","Kp tuning attemp", "Tuned Kp/improved controller")
xlabel("Time (s)", 'FontWeight','bold');
ylabel("SOC (-)", 'FontWeight','bold');
xlim([time2(1) time2(end)])
```




This plot shows the battery State of Charge (SOC) behavior during the WLTP (Worldwide Harmonized Light Vehicles Test Procedure), using two different proportional feedback gains in the ECMS controller:

- Common Kp = 3.8: This is the same coefficient previously tuned for the urban AUDC.
- Tuned Kp = 1: Specifically optimized for the WLTP.

Both profiles start from an SOC of 60%.

Common Kp = 3.8 (Not Tuned for WLTP):

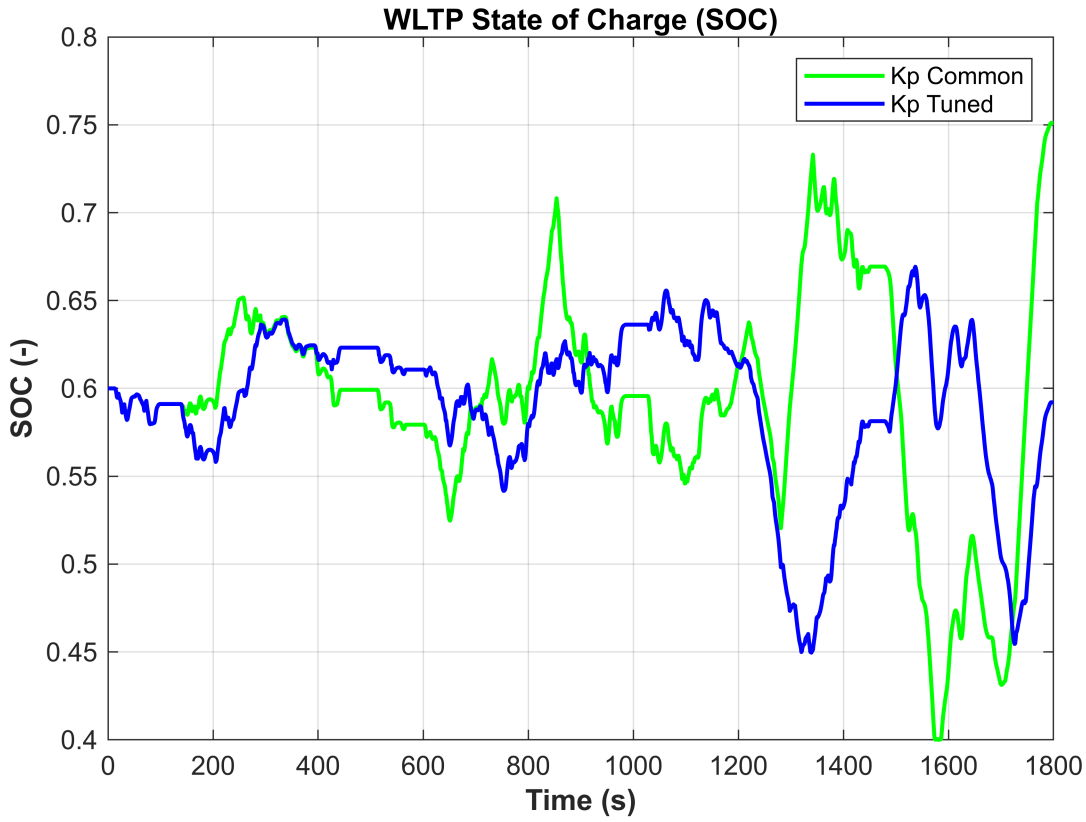
The SOC steadily increases over time, indicating excessive battery charging throughout the WLTP. This behavior is similar to what was observed in the AMDC with the same Kp value. The controller, originally tuned for more frequent changes in vehicle speed and acceleration (as in the AUDC), reacts too strongly in the WLTP, which has more balanced and moderate dynamics. As a result, the control system limits electric drive usage and favors engine operation, which leads to underuse of regenerative braking and unnecessary energy accumulation in the battery.

Tuned Kp = 1 (Optimized for WLTP):

The tuned Kp results in an SOC trajectory that remains much closer to the initial value. This shows that the controller is well-calibrated to match the typical speed and acceleration patterns of the WLTP. It enables a more effective split between engine and electric motor use, ensuring the battery neither overcharges nor depletes excessively. The vehicle likely makes better use of regenerative braking and electric assist, leading to improved energy efficiency.

While the same Kp = 3.8 performs well in the urban AUDC, its application to WLTP and AMDC leads to overcharging and inefficient energy balance. The WLTP, which blends urban, rural, and highway segments, benefits from a moderate Kp like 1, which provides a good balance between responsiveness and smooth SOC control.

```
%State of Charge WLTP
figure;
t = tiledlayout(1,1,'TileSpacing','Compact','Padding','Compact');
ax1 = nexttile;
plot(ax1,time3, SOC3a, "g-", 'LineWidth', 1.5)
grid on
hold on
plot(ax1, time3, SOC3b, "b-", 'LineWidth', 1.5)
title("WLTP State of Charge (SOC)", 'FontWeight', 'bold');
legend("Kp Common","Kp Tuned")
xlabel("Time (s)", 'FontWeight','bold');
ylabel("SOC (-)", 'FontWeight','bold');
xlim([time3(1) time3(end)])
```




This plot illustrates the State of Charge (SOC) profile of a hybrid vehicle over the Turin Test Cycle (TTC) using two proportional feedback gains in the ECMS controller:

- Common Kp = 3.8: This is the same mid-range value used in previous cycles (AUDC, WLTP, AMDC), not specifically tuned for the TTC.
- Tuned Kp = 7: A higher gain chosen to match the dynamics of the TTC.

Both simulations begin at the same initial SOC of 60%.

Common Kp = 3.8 (Not Tuned for TTC):

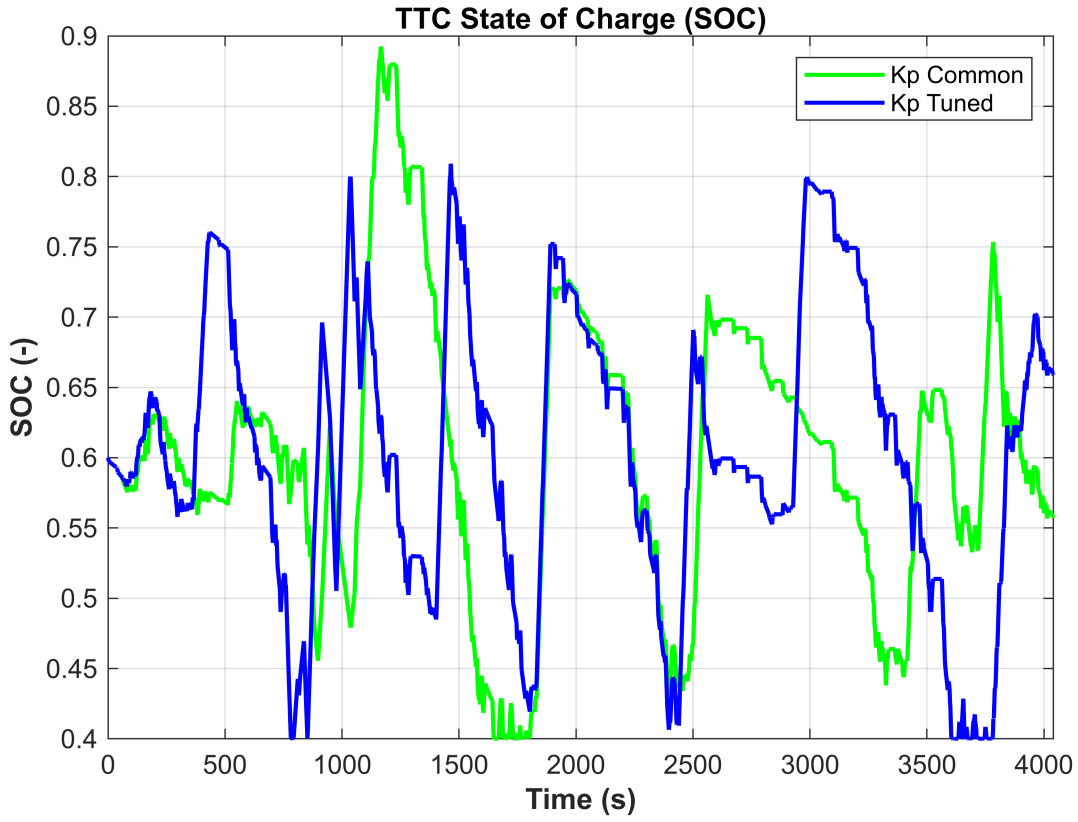
In this configuration, the SOC decreases steadily during the cycle. This suggests that the controller is underreacting to SOC deviations and allows the battery to discharge more than expected. While Kp = 3.8 performed well in the AUDC due to frequent start-stop conditions, it is not aggressive enough for the TTC, which likely contains extended acceleration phases and fewer opportunities for regenerative braking. As a result, the control strategy underutilizes the engine, overly relying on the electric motor and depleting the battery.

Tuned Kp = 7 (Optimized for TTC):

With a higher Kp, the SOC remains much more stable over time. The controller responds more strongly to changes in SOC, compensating quickly to avoid deep discharges. This makes it more suitable for the TTC, which appears to demand high propulsion power and less regenerative energy. The tuned controller prioritizes SOC maintenance, achieving better energy balance by adjusting the contribution of engine and motor effectively during high-load phases.

Comparing TTC with the previous cycles: In the AMDC and WLTP, Kp = 3.8 caused the SOC an overcharging behaviour, due to its strong reaction in smoother or mixed driving conditions.On the other hand, in TTC the same Kp results in a declining SOC, meaning the controller is too mild for the high-power nature of this cycle. This contrast highlights the importance of tuning Kp to the cycle's energy demands. A high Kp is effective in maintaining SOC during high-load, low-regeneration profiles like TTC, while a moderate Kp may suit urban or balanced cycles.

```
%State of Charge TTC
figure;
t = tiledlayout(1,1,'TileSpacing','Compact','Padding','Compact');
ax1 = nexttile;
plot(ax1,time4, SOC4a,"g-",'LineWidth', 1.5)
grid on
hold on
plot(ax1, time4, SOC4b, "b-", 'LineWidth', 1.5)
title("TTC State of Charge (SOC)", 'FontWeight', 'bold');
legend("Kp Common","Kp Tuned")
xlabel("Time (s)", 'FontWeight','bold');
ylabel("SOC (-)", 'FontWeight','bold');
xlim([time4(1) time4(end)])
```




**Equivalce Factor Evolution / Delta SOC**

This figure illustrates the evolution of the equivalence factor throughout the AUDC, updated every 60 seconds. In the upper plot, which illustrates the evolution of the equivalence factor, three control strategies are compared, correspond to different coefficient (Kp values) applied in the controller logic:

- In red, the behavior with a low Kp value equal to 2
- In blue, the behavior with a Kp value specifically tuned for this cycle (Kp = 3.8)
- In green, the behaviour with a high Kp value equalt to 5

The traditional controller includes a very straightforward relation between the deltaSOC feedback and the eq. factor variation, it can be appreciated that the two figures are linked through the incremental interaction since the "s" factor increases proportionally for positive deltaSOC amplitudes , and by converse, decreases proportionally for negative values of the deltaSOC amplitude.

The two plots provide a comparative view of how different feedback control strategies influence both the equivalence factor (s) and the delta SOC over the course of the AUDC driving cycle.

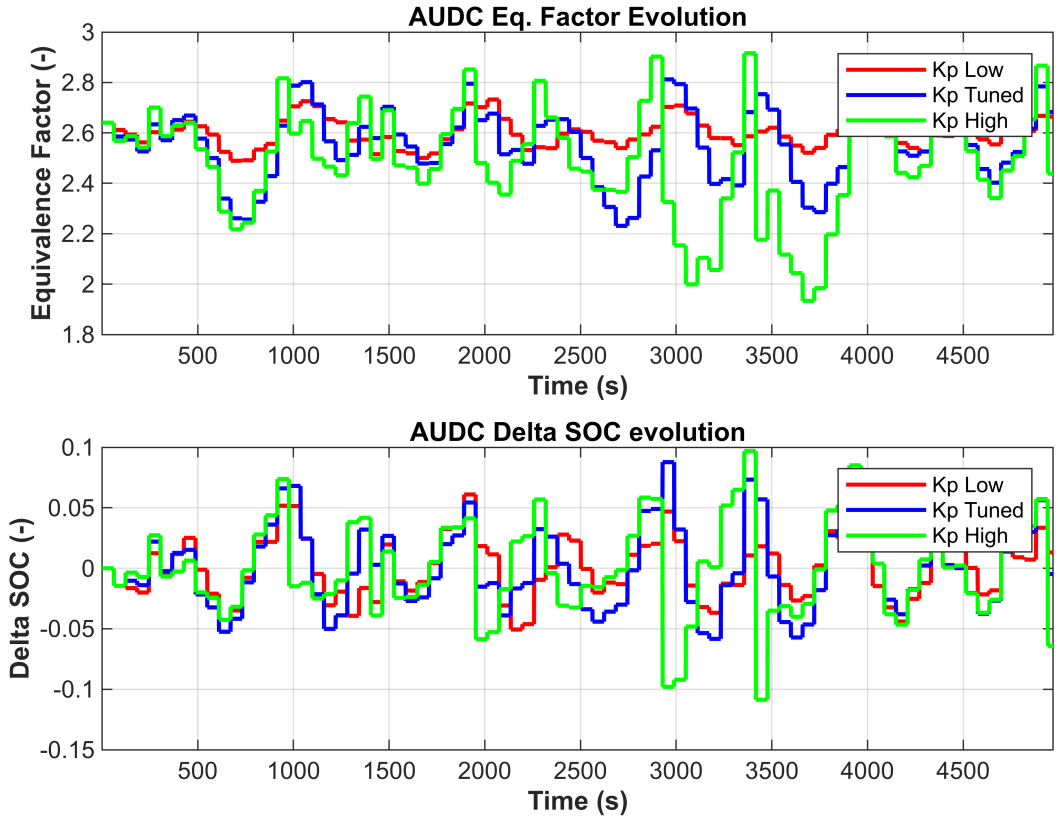
It is possible to observe that the red line (low Kp) exhibits a relatively smooth and slow-changing profile. Its evolution is gradual and shows that the controller reacts only modestly to variations in the deltaSOC. This trend leads to less dynamic correction and, consequently, a slower adaptation of the control strategy. In contrast, the green and blue lines, representing higher coeffcients, demonstrate quicker and more noticeable variations in the equivalence factor. Particularly, the green curve displays rapid rises and drops, suggesting that the controller

is very sensitive to SOC deviations and attempts to immediately correct for them by significantly adjusting the equivalence factor.

The lower plot shows the evolution of the deltaSOC, defined as the variation of the state of charge with respect to its initial value, over the same update interval.

As already explained, the deltaSOC oscillates around zero, which demonstrates that the controller is both fast and highly robust. The best results are achieved with this specific cycle, as the Kp value is not extremely sensitive, making the tuning process relatively straightforward.

```
% AUDC PLOTS

figure;
t = tiledlayout(2,1,'TileSpacing','Compact','Padding','Compact');

% Equivalence factor Evolution
ax1 = nexttile;
plot(time1, eqFactor_n_1a, "r-", 'LineWidth', 1.5)
grid on
hold on
plot(time1, eqFactor_n_1b, "b-", 'LineWidth', 1.5)
hold on
plot(time1, eqFactor_n_1c, "g-", 'LineWidth', 1.5)
title("AUDC Eq. Factor Evolution", 'FontWeight', 'bold');
legend("Kp Low", "Kp Tuned", "Kp High")
xlabel("Time (s)", 'FontWeight','bold');
ylabel("Equivalence Factor (-)", 'FontWeight','bold');
xlim([time1(1) time1(end)])

% Delta SOC evolution
ax2 = nexttile; % Select the second tile (lower plot)
plot(time1, delta_SOC_1a, 'r-', 'LineWidth', 1.5)
grid on
hold on
plot(time1, delta_SOC_1b, 'b-', 'LineWidth', 1.5)
hold on
plot(time1, delta_SOC_1c, 'g-', 'LineWidth', 1.5)
title("AUDC Delta SOC evolution", 'FontWeight', 'bold')
legend("Kp Low", "Kp Tuned", "Kp High")
xlabel("Time (s)", 'FontWeight', 'bold')
ylabel("Delta SOC (-)", 'FontWeight', 'bold')
xlim([time1(1) time1(end)])

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




This figure shows the evolution of the equivalent factor over the AMDC driving cycle, updated every 60 seconds. Two profiles are reported:

- In green, the evolution obtained with a common Kp value (Kp = 3.8);
- In red, the evolution obtained with a Kp = 0.1 which represents the tuning attemp.made with the traditional controller
- In blu, the evolution obtained with the improved controller calibrated using Kp = 5

The first and common value of Kp exhibits an oscillating trend, it basically doesn't converge to any value but rather tries to approximate in a very aggressive way the value of the eq. factor. Altough the deltaSOC demostrates a non-monotonic trend the the "s" parameter widely diverges at the end of the cycle. As already explained the two figures are linked by an incremental relationship but, as it will explained later, this will apply only for the first two curves since the last one uses a different logic and represents an exception in our analysis.

The second Kp value represents the best result from the Kp tuning procedure and, somehow, the testimony that is not always possible to find a suitable vaue for it when it is requested to ensure both charge sustaining bahaviour and an effective use of the control straregy.

The need of crossing the value of 0.6 at the end leads to an unreasonable Kp tuning: on one hand the Kp is needed to be low enough not to increased too much a value already close to the calibrated one (risking to overcorrect the "s" value and exceed the final target), on the other hand, the controller must be fast enough to increased a "s" value which is too low and would bring to a over-utilization of the battery and, conseguently, to a lower saturation of the SOC (as accours in the Kp = 0.1).

This second result shows the effect of a too robust and slow controller, the "s" parameter grows at a very low pace but, as a result of the feedback nature, its adaptation fails to converge or approximate to the optimum the eq. factor.

The third case represents the exception: the controller used is the improved one and is aimed to fix the malfunctions of the traditional one. It's operating principle, in our very preliminar usage, it's cycle specific and, therefore, the simulation focuses with the cycle in question.

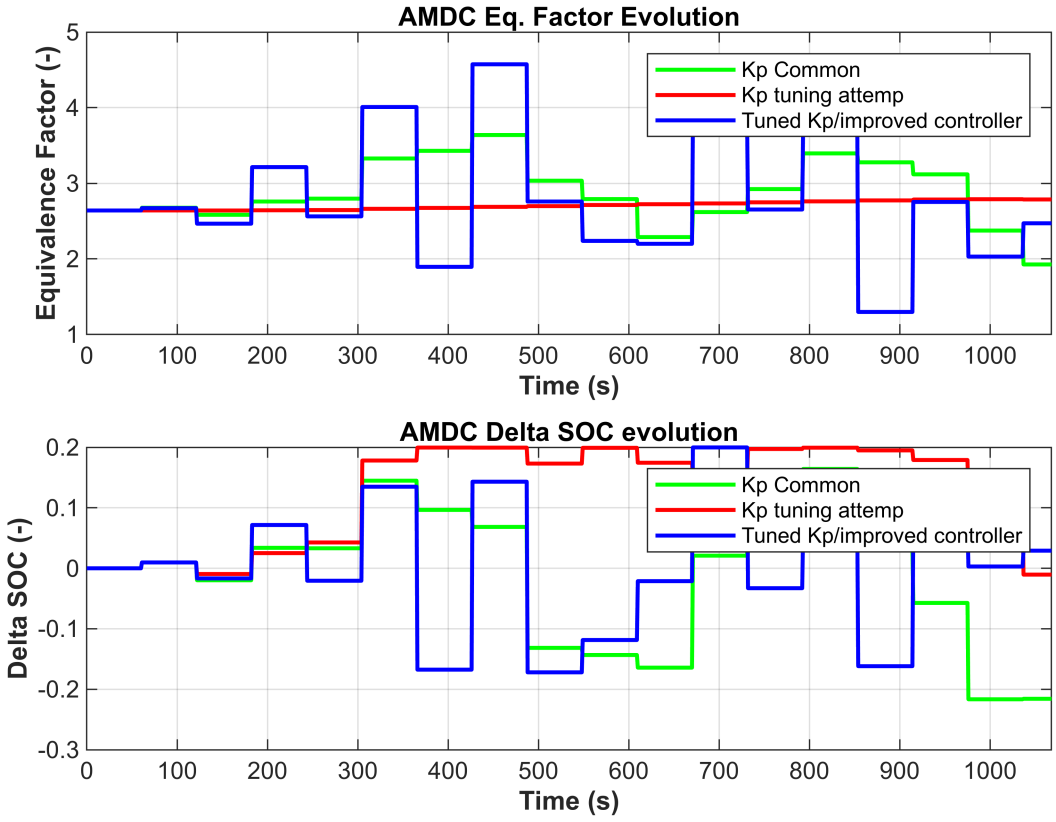
As it will be explained later, this controller doesn't follow anymore the numerous depicted relationship between the eq. factor and deltaSOC, but includes a more sophisticated criterion so that only some updating of the "s" parameter follows it, therefore, the comparison between graphs is no more reliable.

The tuning has been made in a very similar way, even tough with some evident differences as the increased sensitivity. The tuning succeded in ensuring charge sustaining and a good enough optimization strategy but without endowing any convergence to the equivalence value.

The "s" parameter shows a very oscillatory behaviour demostrating that in case of this improved controller the tuning and the robustness of the system degradate.

```
% AMDC PLOTS

figure;
t = tiledlayout(2,1,'TileSpacing','Compact','Padding','Compact');

% Equivalence factor Evolution
ax1 = nexttile;
plot(time2, eqFactor_n_2a, "g-", 'LineWidth', 1.5)
grid on
hold on
plot(time2, eqFactor_n_2b, "r-", 'LineWidth', 1.5)
hold on
plot(time2, eqFactor_n_2c, "b-", 'LineWidth', 1.5)
title("AMDC Eq. Factor Evolution", 'FontWeight', 'bold');
legend("Kp Common","Kp tuning attemp", "Tuned Kp/improved controller")
xlabel("Time (s)", 'FontWeight','bold');
ylabel("Equivalence Factor (-)", 'FontWeight','bold');
xlim([time2(1) time2(end)])

% Delta SOC evolution
ax2 = nexttile; % Select the second tile (lower plot)
plot(time2, delta_SOC_2a, 'g-', 'LineWidth', 1.5)
grid on
hold on
plot(time2, delta_SOC_2b, 'r-', 'LineWidth', 1.5)
hold on
plot(time2, delta_SOC_2c, 'b-', 'LineWidth', 1.5)
title("AMDC Delta SOC evolution", 'FontWeight', 'bold')
legend("Kp Common","Kp tuning attemp", "Tuned Kp/improved controller")
```

```
xlabel("Time (s)", 'FontWeight', 'bold')
ylabel("Delta SOC (-)", 'FontWeight', 'bold')
xlim([time2(1) time2(end)])

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




This figure illustrates the evolution of the equivalence factor throughout the WLTP cycle, updated every 60 seconds. Two control strategies are compared:

- In green, the behavior with a common Kp value equal to 3.8;
- In blue, the behavior with a Kp value specifically tuned for this cycle (Kp = 1).

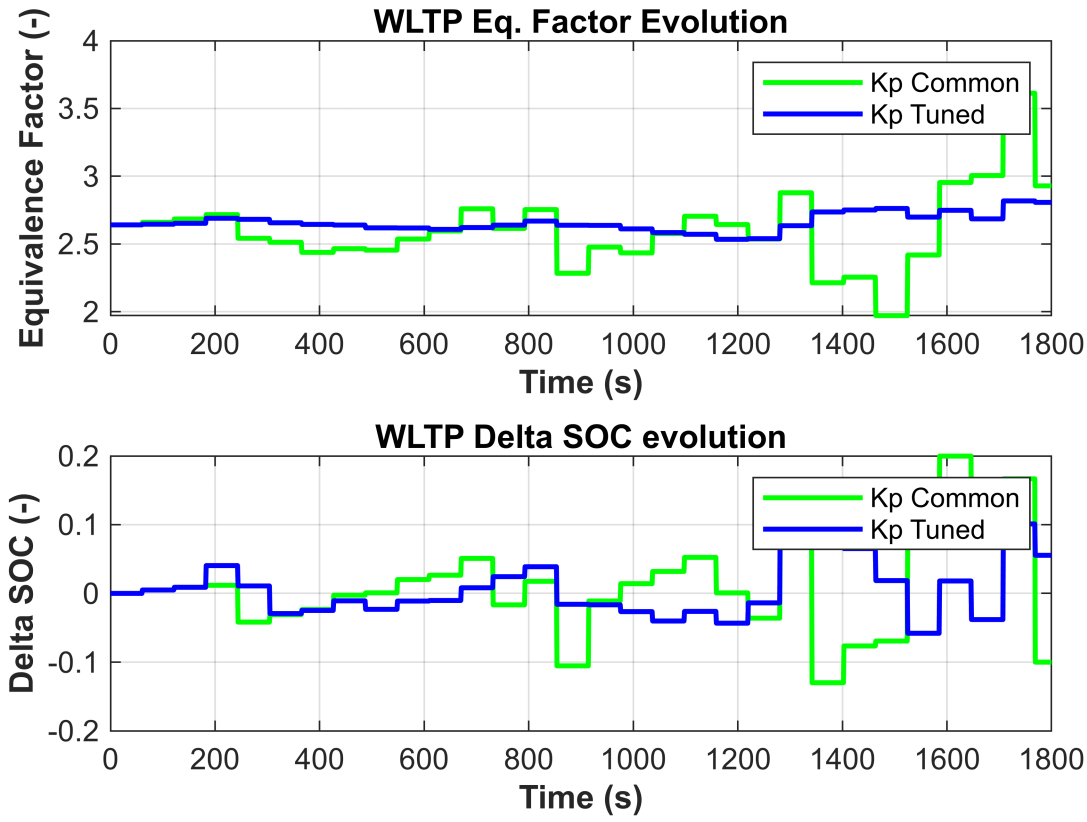
In the first case (Kp = 3.8), the equivalence factor s exhibits larger and less coherent variations, with significant peaks observed during the final stages of the cycle. This indicates a limited adaptability of the controller, which fails to effectively respond to the energy demand dynamics of the WLTP cycle. The strategy appears unable to maintain a balanced trade-off between thermal and electric power contributions under varying conditions.

Conversely, with the tuned Kp value (Kp = 1), the evolution of the equivalence factor is more stable and controlled, highlighting the controller's improved ability to dynamically adjust the energy management policy. This results in a more precise modulation between combustion engine usage and battery contribution.

The lower plot shows the evolution of the delta SOC, defined as the variation of the state of charge with respect to its initial value, over the same update interval.

- With Kp = 3.8, the SOC tends to deviate significantly from the target value, with erratic and poorly regulated behavior.
- With Kp = 1, the delta SOC remains within narrower bounds, suggesting that the controller is better able to respect the SOC constraints and to implement a more stable and effective energy management strategy.

```
% WLTP PLOTS

figure;
t = tiledlayout(2,1,'TileSpacing','Compact','Padding','Compact');

% Equivalence factor Evolution
ax1 = nexttile;
plot(time3, eqFactor_n_3a, "g-", 'LineWidth', 1.5)
grid on
hold on
plot(time3, eqFactor_n_3b, "b-", 'LineWidth', 1.5)
title("WLTP Eq. Factor Evolution", 'FontWeight', 'bold');
legend("Kp Common","Kp Tuned")
xlabel("Time (s)", 'FontWeight','bold');
ylabel("Equivalence Factor (-)", 'FontWeight','bold');
xlim([time3(1) time3(end)])

% Delta SOC evolution
ax2 = nexttile; % Select the second tile (lower plot)
plot(time3, delta_SOC_3a, 'g-', 'LineWidth', 1.5)
grid on
hold on
plot(time3, delta_SOC_3b, 'b-', 'LineWidth', 1.5)
title("WLTP Delta SOC evolution", 'FontWeight', 'bold')
legend("Kp Common","Kp Tuned")
xlabel("Time (s)", 'FontWeight', 'bold')
ylabel("Delta SOC (-)", 'FontWeight', 'bold')
xlim([time3(1) time3(end)])

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




This figure shows the evolution of the equivalent factor over the TTC driving cycle, updated every 60 seconds. Two profiles are reported:

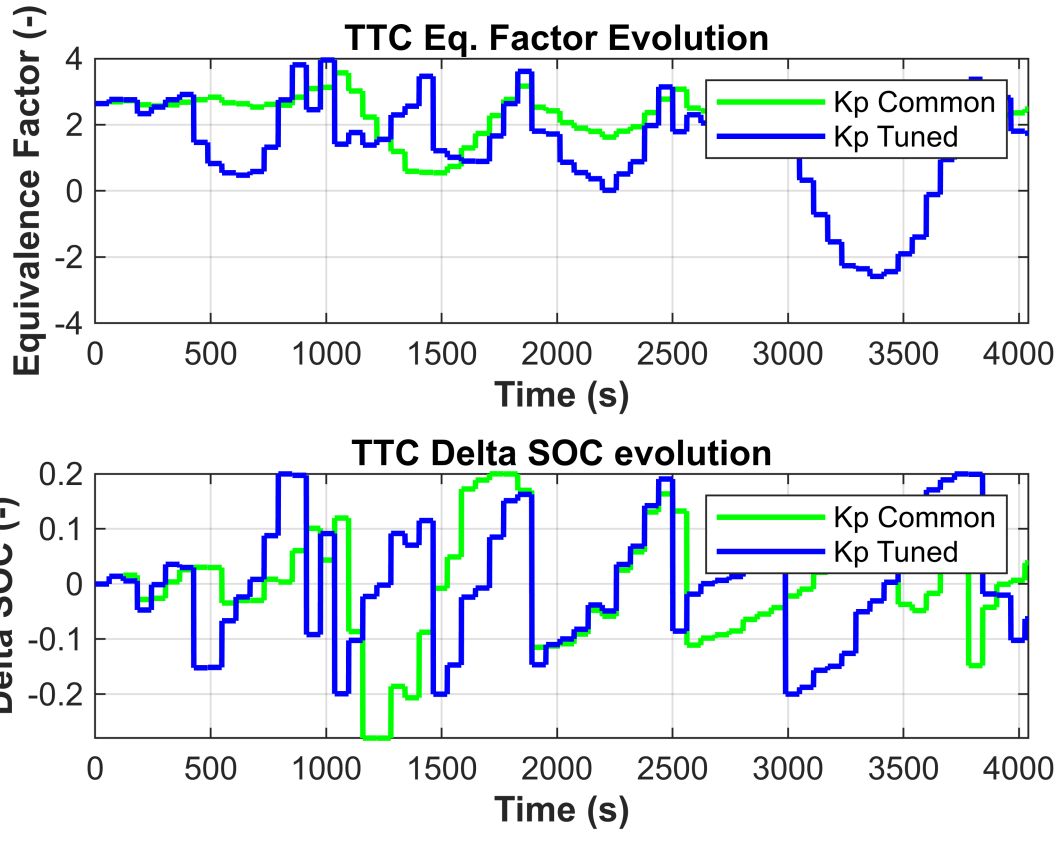
- In green, the evolution obtained with a common Kp value (Kp = 3.8);
- In blue, the evolution obtained with a Kp value specifically tuned for this cycle (Kp = 7).

In the first case, the equivalent factor *s* exhibits very limited variation over time, indicating a reduced adaptability of the controller to the variability of the driving cycle. This behavior suggests that the algorithm fails to effectively adjust the trade-off between the use of the electric and thermal components in response to instantaneous operating conditions. This Kp value leads to a more convergent behavior of the equivalent factor, which, however, does not approximate the calibrated value effectively.

Conversely, with the tuned Kp value, more pronounced oscillations in the equivalent factor *s* are observed, indicating a dynamically adaptive behavior. This tuning leads to a more divergent evolution of *s*, also decreasing up to negative value, nevertheless it better approximates the calibrated value, thus succeding in ensuring the meet of the target SOC final value and a good enough optimization strategy.

```
% TTC PLOTS

figure;
t = tiledlayout(2,1,'TileSpacing','Compact','Padding','Compact');
```

```
% Equivalence factor Evolution
ax1 = nexttile;
plot(time4, eqFactor_n_4a, "g-", 'LineWidth', 1.5)
grid on
hold on
plot(time4, eqFactor_n_4b, "b-", 'LineWidth', 1.5)
title("TTC Eq. Factor Evolution ", 'FontWeight', 'bold');
legend("Kp Common","Kp Tuned")
xlabel("Time (s)", 'FontWeight','bold');
ylabel("Equivalence Factor (-)", 'FontWeight','bold');
xlim([time4(1) time4(end)])

% Delta SOC evolution
ax2 = nexttile; % Select the second tile (lower plot)
plot(time4, delta_SOC_4a, 'g-', 'LineWidth', 1.5)
grid on
hold on
plot(time4, delta_SOC_4b, 'b-', 'LineWidth', 1.5)
title("TTC Delta SOC evolution", 'FontWeight', 'bold')
legend("Kp Common","Kp Tuned")
xlabel("Time (s)", 'FontWeight', 'bold')
ylabel("Delta SOC (-)", 'FontWeight', 'bold')
xlim([time4(1) time4(end)])

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



**Engine Power Profiles**

In the AUDC (Artemis Urban Driving Cycle), the engine power profiles obtained with two different values of the coefficient Kp were compared:

- Kp = 2, a lower value w.r.t. the optimal one
- Kp = 3.8, a generic and commonly used value used for all the cycles (in this case also the optimized one)
- Kp = 5, an upper value w.r.t. the optimal one

The analysis of engine power profiles for the AUDC highlights the influence of the Kp parameter on fuel consumption, average engine power output, and the evolution of the battery's state of charge (SOC). Three distinct Kp values were considered: 2, 3.8 (tuned value), and 5.

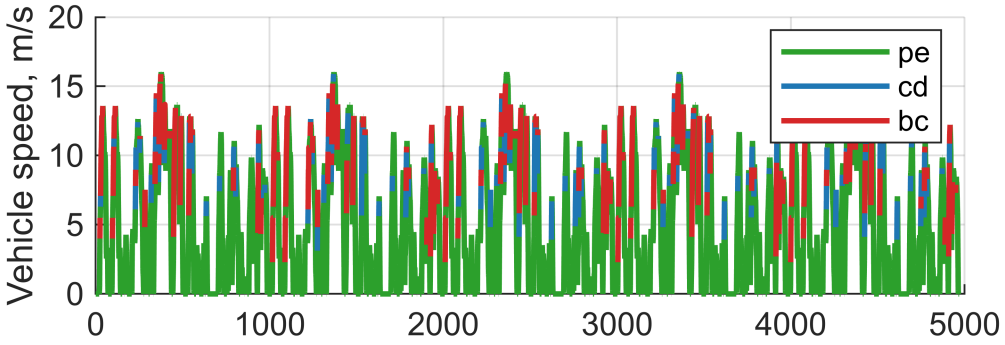
For Kp = 2, the engine power profile is characterized by highly discontinuous behavior, with frequent engine shut-offs. This configuration results in the lowest fuel consumption (0.64 kg) and a relatively low average power output (2167.9 W). However, the final SOC drops to 0.58, indicating a strategy that excessively relies on battery usage, compromising SOC sustainability over time.

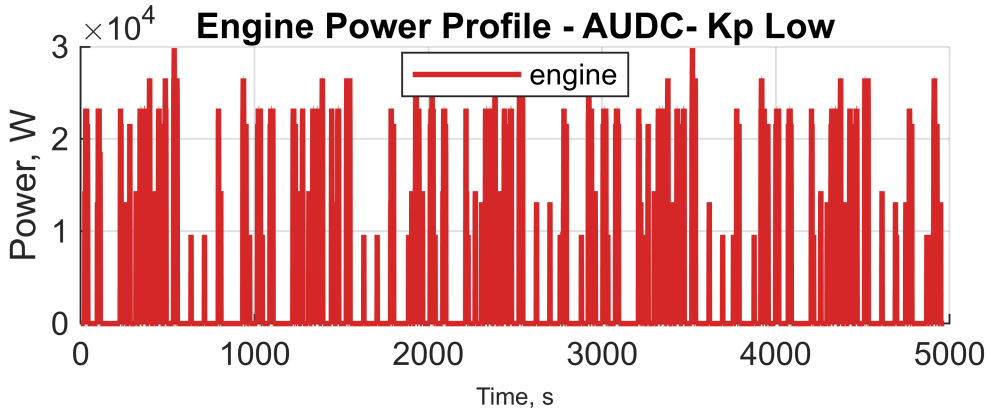
At Kp = 3.8, the tuned value, the engine activation is more balanced. Fuel consumption slightly increases to 0.65 kg, as does the average power (2195.9 W), but the final SOC reaches 0.60, close to the initial value. This suggests that the control strategy successfully maintains the energy balance between the battery and the engine, achieving an effective compromise between fuel efficiency and SOC management.

For Kp = 5, the engine remains active more frequently and delivers higher power peaks. This leads to the highest fuel consumption (0.67 kg) and average power output (2251.8 W). The final SOC increases to 0.63, indicating a strategy highly oriented toward maintaining or increasing the battery charge level, but at the expense of overall energy efficiency.

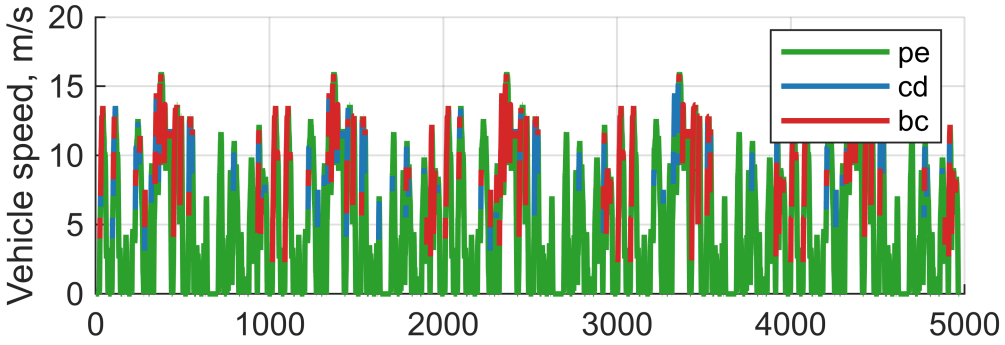
In summary, increasing Kp makes the ECMS strategy more sensitive to SOC variations but less efficient in terms of fuel usage. A low Kp value results in underutilization of the engine and a progressive battery depletion, while a high Kp causes excessive engine usage to recharge the battery. The tuned value Kp = 3.8 represents the best trade-off between fuel savings and SOC stability.

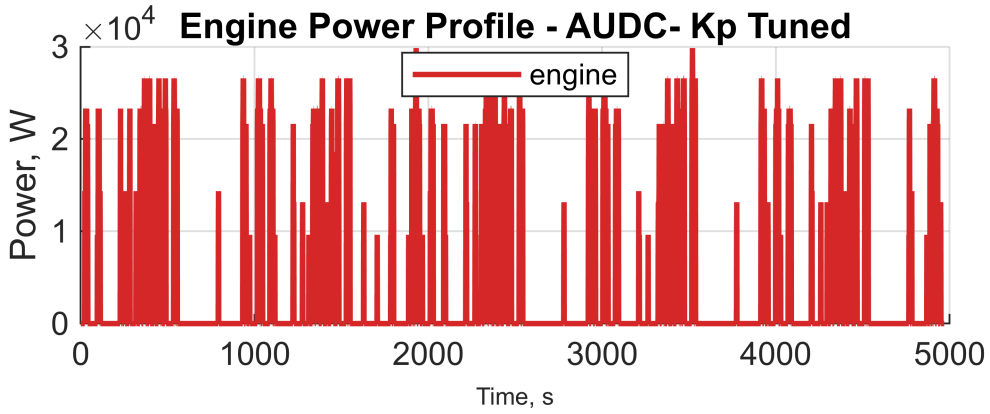
```
% AUDC Engine Power Profiles
% Plot the speed profile and the engine power over time
powerProfiles(prof1a, 'eng');
title("Engine Power Profile - AUDC- Kp Low")
```



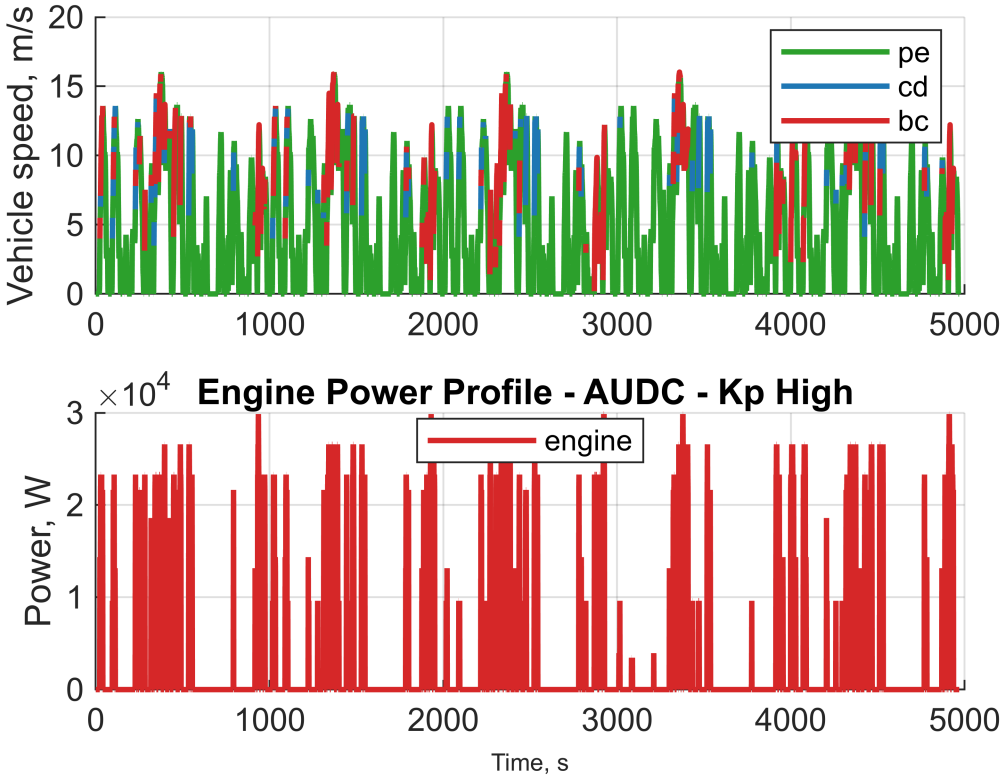




```
powerProfiles(prof1b, 'eng');
title("Engine Power Profile - AUDC- Kp Tuned")
```




```
powerProfiles(prof1c, 'eng');
title("Engine Power Profile - AUDC - Kp High ")
```




In the AMDC (Artemis Motorway Driving Cycle), the engine power profiles obtained with two different values of the coefficient Kp were compared:

- Kp = 3.8, a generic and commonly used value used for all the cycles
- A tuning attemp with Kp = 0.1,  consdering the AMDC conditions.
- A tuned Kp = 5,  specifically optimized for AMDC cycle using the improved controller

Analyzing the power profile relative to the value of Kp = 3.8, the internal combustion engine (ICE) power graph, it can be observed that engine activation is frequent and fragmented, with power peaks occasionally exceeding 60 kW, particularly during the central phase of the cycle. This behavior suggests a conservative energy management strategy, where the controller tends to activate the engine even when the battery alone could sustain the required load.

It can also be observed that during deceleration phases, the internal combustion engine is used to recharge the battery. This behavior is not consistent with an optimal energy management strategy, as battery recharging in such conditions should rely exclusively on the regenerative braking system. Avoiding the use of the ICE in these phases would enhance the overall efficiency of the cycle.

The final State of Charge (SOC) reaches 0.81, well above the initial value. This indicates that the controller not only aimed to maintain the SOC but also excessively recharged the battery, leading to inefficient use of the ICE and an overly cautious control behavior.

This configuration highlights how the Kp parameter amplifies the response of the equivalence factor to SOC variations, resulting in early engine activation and more aggressive recharging phases.

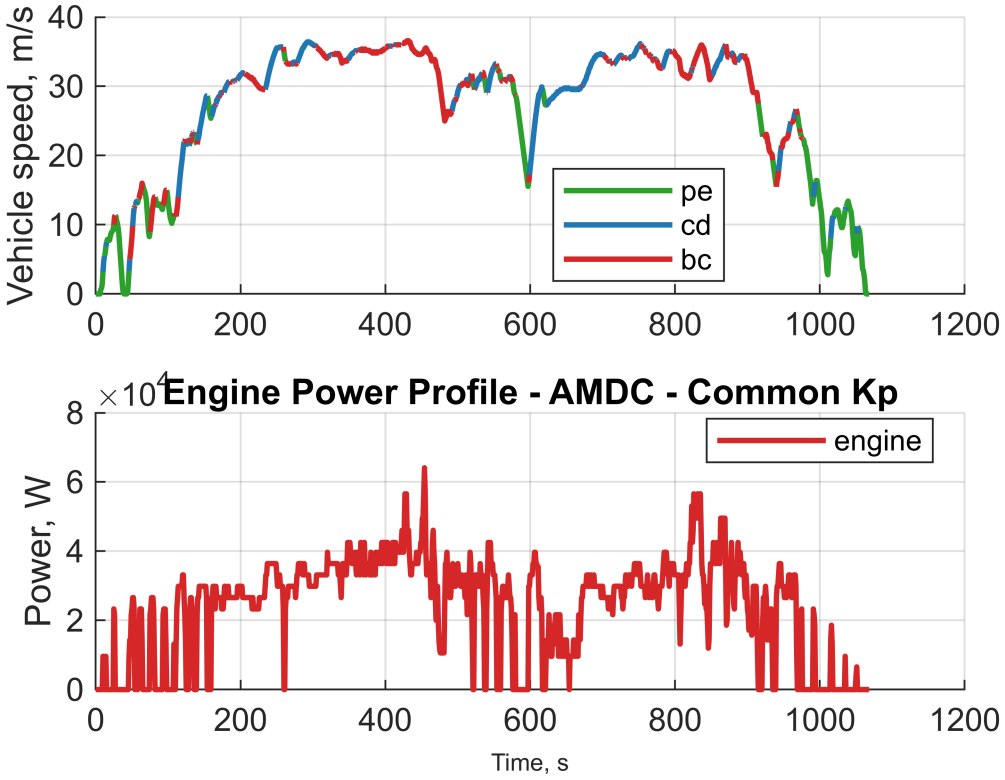
With reference to the optimal value of Kp = 0.1, it can be observed that, although this setting allows the SOC to approach the desired target of 0.6 by the end of the cycle, the final value reaches 0.65, indicating a slight overestimation,so  the overall charging profile is not optimal. In fact, the SOC remains close to the lower bound of 0.4 for the majority of the cycle, increasing only during the final phase. This behavior indicates that Kp = 0.1 represents the limit beyond which the final SOC target can no longer be met: even small deviations from this value result in a final SOC that does not comply with the desired objective.

Nevertheless, an advantage is observed in terms of fuel consumption: the total fuel used is lower than in the case with the non-tuned Kp value (1.49 kg compared to 1.54 kg), despite the final SOC being different. This confirms that the ECMS strategy has effectively prioritized fuel minimization, even if it compromises optimal SOC management.

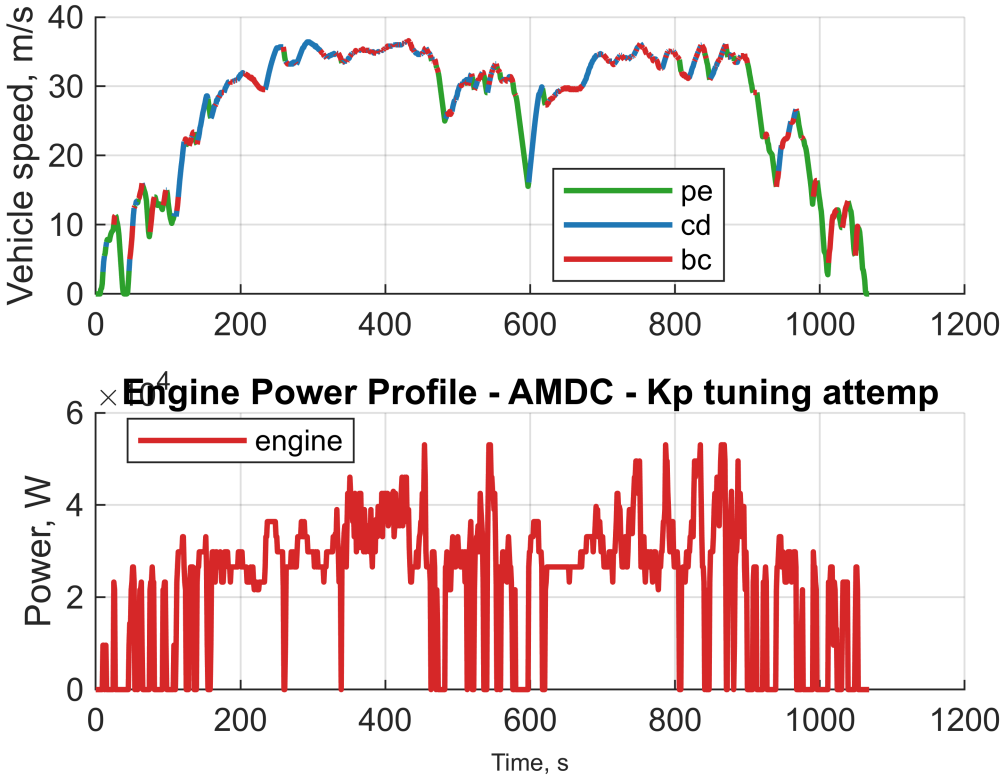
In the case of Kp = 5, obtained through the use of a modified and improved version of the A-ECMS controller, it was finally possible to successfully tune the AMDC cycle, an outcome that could not be achieved with the standard controller. The fuel consumption is the lowest among the three cases analyzed (1.48 kg), confirming a better trade-off between fuel economy and compliance with the SOC constraint.

Additionally, it can be observed that the power profile of the ICE is smoother, without abnormal peaks or frequent on-off switching events, further supporting the improved behavior of the system.

```
% AMDC Engine Power Profiles
% Plot the speed profile and the engine power over time
powerProfiles(prof2a, 'eng');
title("Engine Power Profile - AMDC - Common Kp")
```



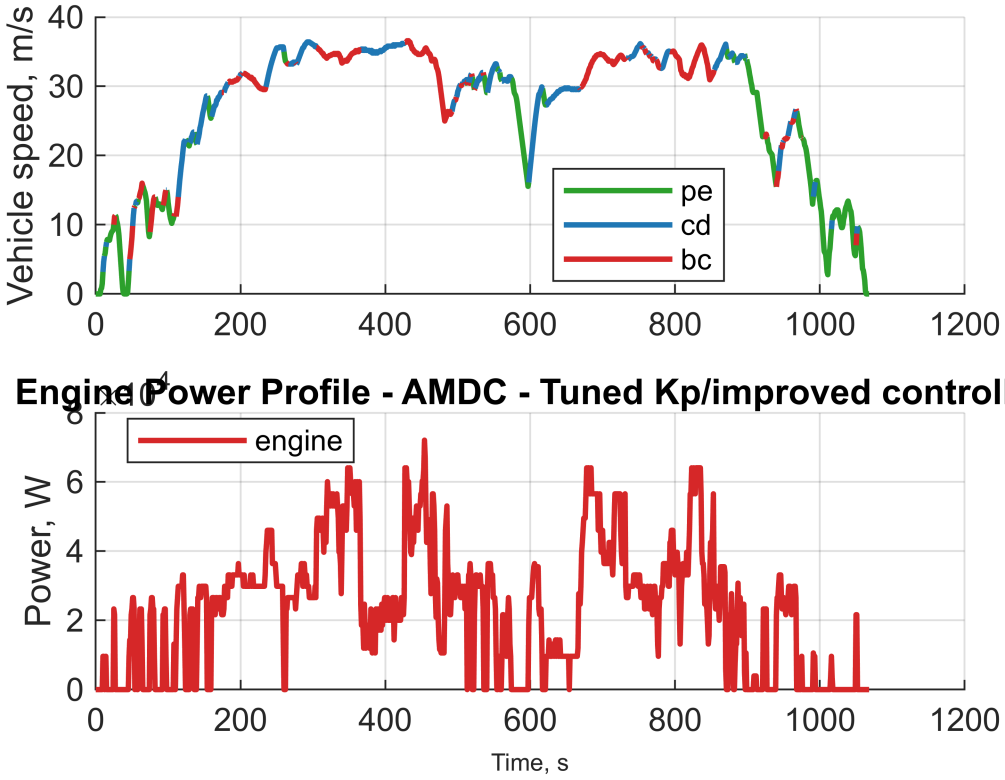

```
powerProfiles(prof2b, 'eng');
title("Engine Power Profile - AMDC - Kp tuning attemp")
```







```
powerProfiles(prof2c, 'eng');
title("Engine Power Profile - AMDC - Tuned Kp/improved controller")
```




In the WLTP (Worldwide Harmonized Light Vehicles Test Procedure) cycle, the engine power profiles obtained with two different values of the coefficient Kp were compared:

- Kp = 3.8, a generic and commonly used value used for all the cycles
- Tuned Kp = 1, Specifically optimized for the WLTP cycle.

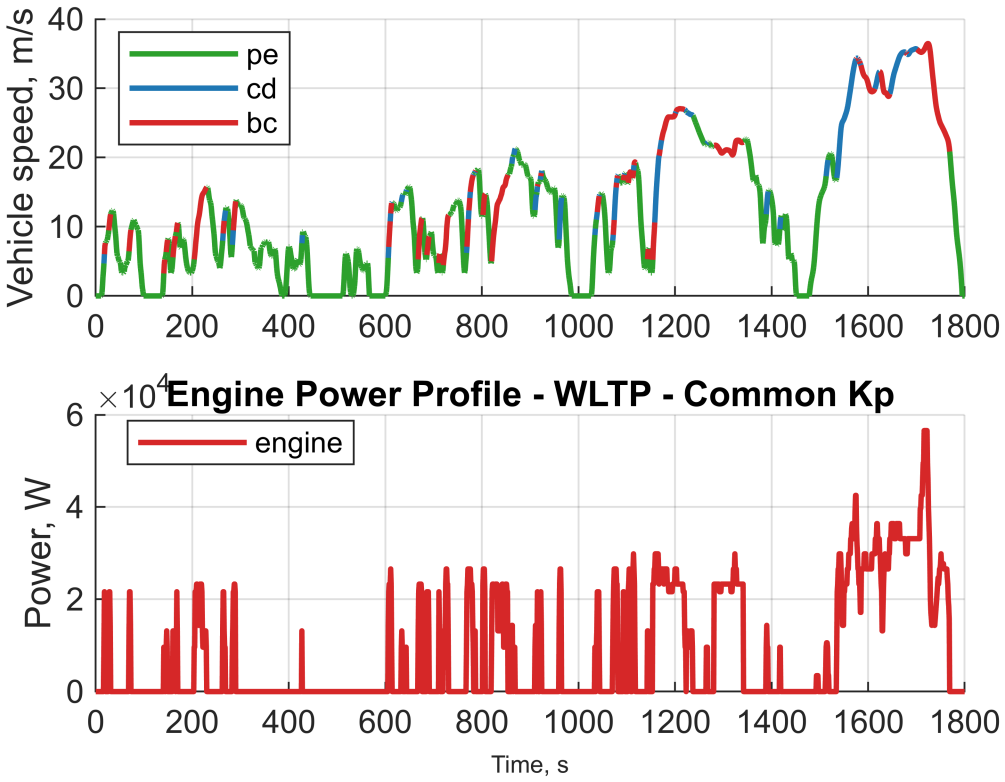
By analyzing the system behavior with Kp = 3.8, it is observed that the internal combustion engine (ICE) is activated more frequently, through short and impulsive power peaks. This indicates an aggressive correction by the controller, with power spikes reaching up to 60 kW. A large multiplication coefficient leads to a strong sensitivity of the equivalence factor to the SOC deviation, driving the controller toward a battery charging strategy: the ICE is turned on even when the battery alone could meet the power demand. As a result, the final SOC increases to 0.75, showing a tendency to restore the battery charge level by forcing the ICE to operate, even during low-demand phases.

With the optimized value Kp = 1, the system exhibits a more balanced behavior. ICE activation is more selective, occurring primarily during high power demand phases, making engine operation more efficient and less fragmented. In this case, the final SOC remains equal to the initial value of 0.6, avoiding unnecessary overcharging of the battery.

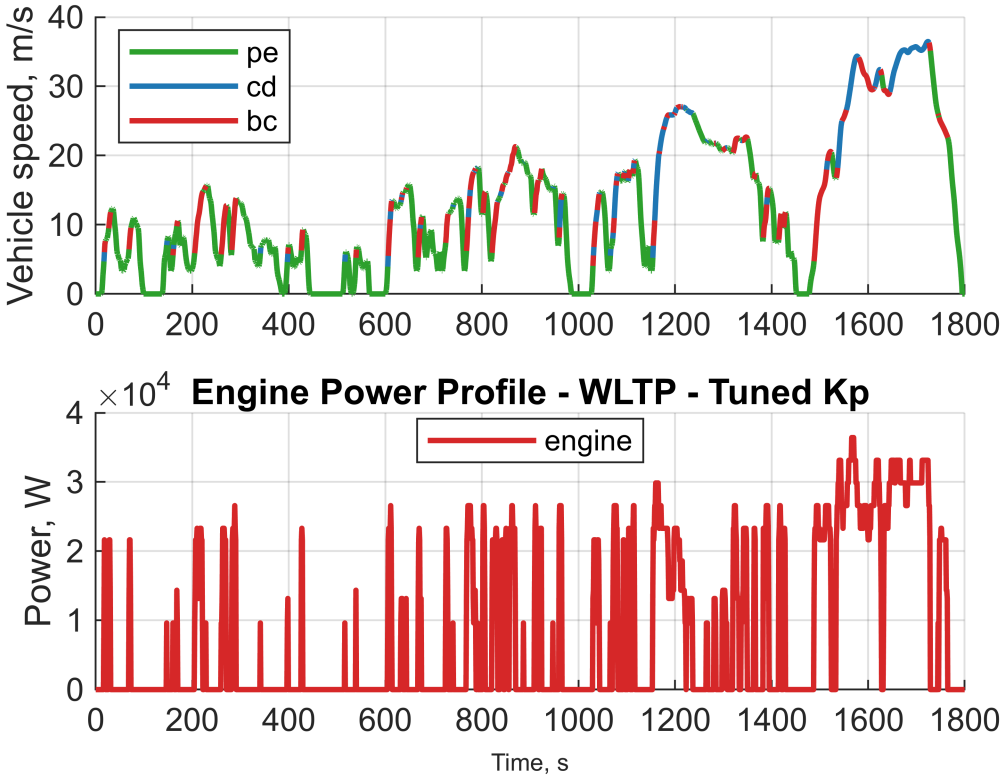
However, it should be noted that using a generic value of Kp = 3.8 may offer greater generalization capabilities across different drive cycles, representing a trade-off between local efficiency (optimized with Kp = 1) and global robustness of the controller, demonstrating that this control logic of the Adaptive ECMS is more effective than

the previous one highlights how it provides greater adaptability to the varying operating conditions of the cycle compared to the conventional ECMS strategy.

```
% WLTP Engine Power Profiles
% Plot the speed profile and the engine power over time
powerProfiles(prof3a, 'eng');
title("Engine Power Profile - WLTP - Common Kp")
```




```
powerProfiles(prof3b, 'eng');
title("Engine Power Profile - WLTP - Tuned Kp")
```




In the Turin Test Cycle, the engine power profiles obtained with two different values of the coefficient Kp were compared:

- Kp = 3.8, a generic and commonly used value used for all the cycles
- Kp = 7, a value specifically optimized for this cycle

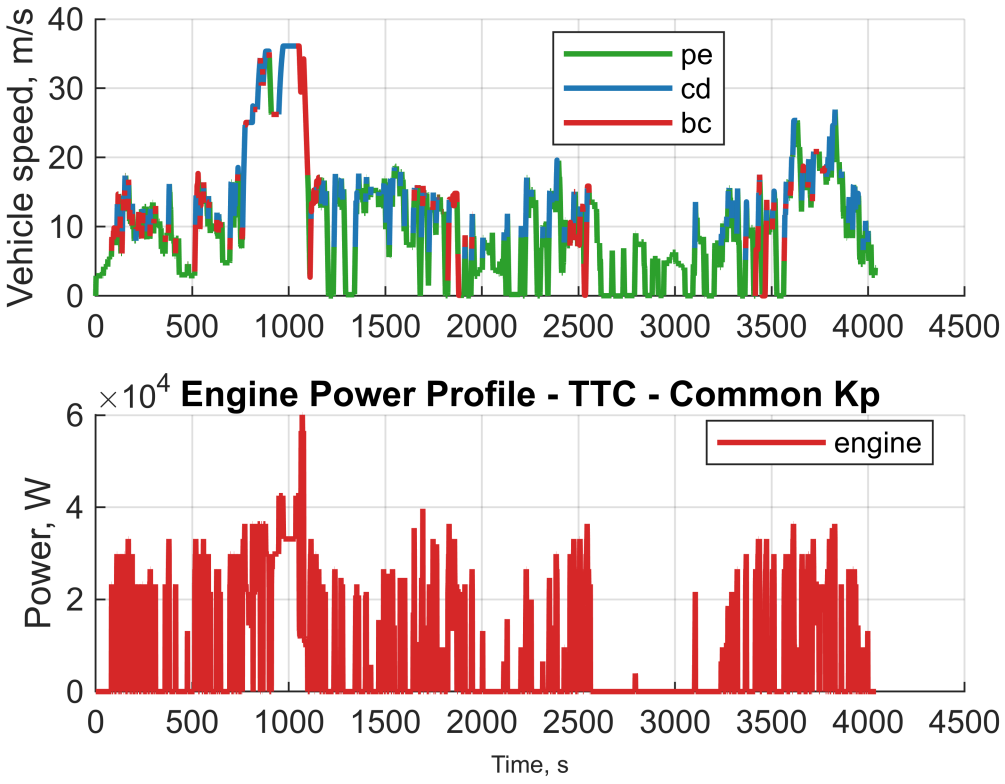
Unlike the previous case, the control corresponding to the common gain value of Kp = 3.8 exhibits less pronounced oscillations in the power output of the internal combustion engine, with a maximum peak reaching 60 kW. This behavior reflects a less aggressive and more balanced control strategy. However, this Kp value is not optimal, as during certain phases of the cycle the State of Charge (SOC) exceeds the predefined upper limit of 0.8, reaching values close to 0.9. Additionally, it is observed that there are fewer segments where the batteries are being recharged, despite the overall delta SOC being comparable.This implies that recharging occurs over a shorter time interval, forcing the engine to operate more frequently in suboptimal conditions and reduced efficiency. Nevertheless, the final SOC value tends to approach the desired target of 0.6 , reaching approximately 0.55, indicating that this common Kp value can still be considered acceptable for the overall cycle.

As for the case of the tuned Kp, the power profile of the internal combustion engine reveals less effective control in maintaining stable power levels, with peaks reaching 72 kW. The increase in mean power from 59.2 kW to 61.4 kW indicates a higher energy usage, which contributes to a better final battery state, as the SOC increases from 0.55 to 0.65. However, as also observed in the SOC analysis, this Kp value allows the battery state of charge to oscillate within a more functional range, ultimately achieving a final value very close to 0.6. This type

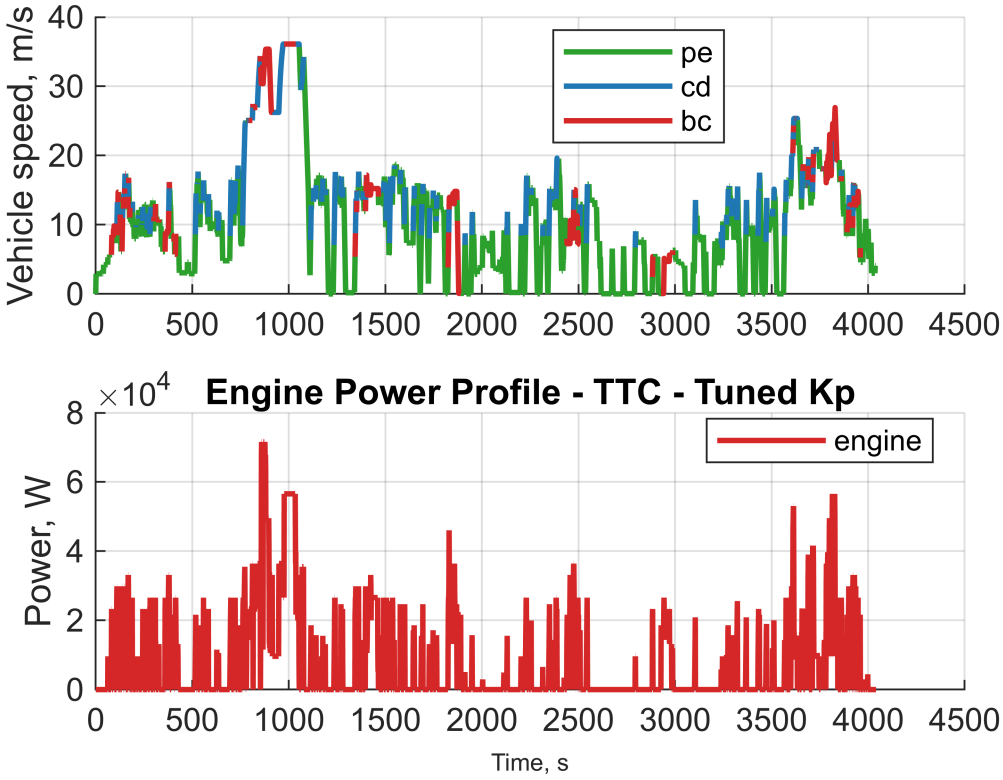
of control is therefore more aggressive, but also more efficient for the specific driving cycle, as it optimizes the engine's operating modes.

In contrast to the previous case, the battery charging through the engine occurs over more limited segments of the cycle. For example, during the braking phase around second 1100, the thermal engine is used to recharge the battery, suggesting a different energy management approach. Ideally, regenerative braking should be prioritized in such phases, while the generator should contribute more significantly during low-load driving phases to improve overall system efficiency.

```
% TTC Engine Power Profiles
% Plot the speed profile and the engine power over time
powerProfiles(prof4a, 'eng');
title("Engine Power Profile - TTC - Common Kp")
```




```
powerProfiles(prof4b, 'eng');
title("Engine Power Profile - TTC - Tuned Kp")
```




## RESULTS EXTRAPOLATION

The following lines of code are reserved to the results computation:

### AUDC RESULTS

This section presents the extrapolated results concerning key performance indicators using different values of Kp, namely:

- Total fuel consumption
- Final State of Charge (SOC)
- Instantaneous engine power
- Average engine power

These metrics were analyzed to assess the effectiveness and consistency of the implemented control strategy under different operating conditions.

**Low Kp results**

```
% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption_1a = trapz(fuelFlwRate1a)/1e3 % (Kg)
```

```
fuelConsumption_1a =
0.6431
```

```
finalSOC_1a = SOC1a(end); % Final value of SOC

% Mean engine power
engPower1a = [prof1a.engSpd] .* [prof1a.engTrq]; % Computing the instantaneous
engine power
meanPwr1a = mean(engPower1a) % Computing the mean engine power
```

```
meanPwr1a =
2.1679e+03
```

### High Kp results

```
% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption_1b = trapz(fuelFlwRate1b)/1e3 % (Kg)
```

```
fuelConsumption_1b =
0.6520
```

```
finalSOC_1b = SOC1b(end); % Final value of SOC

% Mean engine power
engPower1b = [prof1b.engSpd] .* [prof1b.engTrq]; % Computing the instantaneous
engine power
meanPwr1b = mean(engPower1b) % Computing the mean engine power
```

```
meanPwr1b =
2.2018e+03
```

### Tuned Kp results

```
% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption_1c = trapz(fuelFlwRate1c)/1e3 % (Kg)
```

```
fuelConsumption_1c =
0.6735
```

```
finalSOC_1c = SOC1c(end); % Final value of SOC

% Mean engine power
engPower1c = [prof1c.engSpd] .* [prof1c.engTrq]; % Computing the instantaneous
engine power
meanPwr1c = mean(engPower1c) % Computing the mean engine power
```

```
meanPwr1c =
2.2729e+03
```

## AMDC RESULTS

This section presents the extrapolated results concerning key performance indicators using different values of Kp, namely:

- Total fuel consumption
- Final State of Charge (SOC)
- Instantaneous engine power
- Average engine power

These metrics were analyzed to assess the effectiveness and consistency of the implemented control strategy under different operating conditions.

### Common Kp results

```
% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption_2a = trapz(fuelFlwRate2a)/1e3 % (Kg)
```

```
fuelConsumption_2a =
1.5429
```

```

finalSOC_2a = SOC2a(end); % Final value of SOC

% Mean engine power
engPower2a = [prof2a.engSpd] .* [prof2a.engTrq]; % Computing the instantaneous
engine power
meanPwr2a = mean(engPower2a) % Computing the mean engine power
```

```
meanPwr2a =
2.3727e+04
```

### Kp tuning attemp results

```
% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption_2b = trapz(fuelFlwRate2b)/1e3 % (Kg)
```

```
fuelConsumption_2b =
1.4904
```

```

finalSOC_2b = SOC2b(end); % Final value of SOC

% Mean engine power
engPower2b = [prof2b.engSpd] .* [prof2b.engTrq]; % Computing the instantaneous
engine power
meanPwr2b = mean(engPower2b) % Computing the mean engine power
```

```
meanPwr2b =
2.3005e+04
```

### Tuned Kp results

```
% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption_2c = trapz(fuelFlwRate2c)/1e3 % (Kg)
```

```
fuelConsumption_2c =
1.5531
```

```
finalSOC_2c = SOC2c(end); % Final value of SOC

% Mean engine power
engPower2c = [prof2c.engSpd] .* [prof2c.engTrq]; % Computing the instantaneous
engine power
meanPwr2c = mean(engPower2c) % Computing the mean engine power
```

```
meanPwr2c =
2.3328e+04
```

## WLTP RESULTS

This section presents the extrapolated results concerning key performance indicators using different values of Kp, namely:

- Total fuel consumption
- Final State of Charge (SOC)
- Instantaneous engine power
- Average engine power

These metrics were analyzed to assess the effectiveness and consistency of the implemented control strategy under different operating conditions.

### Common Kp results

```
% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption_3a = trapz(fuelFlwRate3a)/1e3 % (Kg)
```

```
fuelConsumption_3a =
0.8593
```

```
finalSOC_3a = SOC3a(end); % Final value of SOC

% Mean engine power
engPower3a = [prof3a.engSpd] .* [prof3a.engTrq]; % Computing the instantaneous
engine power
meanPwr3a = mean(engPower3a) % Computing the mean engine power
```

```
meanPwr3a =
7.9341e+03
```

### Tuned Kp results

```
% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption_3b = trapz(fuelFlwRate3b)/1e3 % (Kg)
```

```
fuelConsumption_3b =
0.8075
```

```
finalSOC_3b = SOC3b(end); % Final value of SOC

% Mean engine power
engPower3b = [prof3b.engSpd] .* [prof3b.engTrq]; % Computing the instantaneous
engine power
meanPwr3b = mean(engPower3b) % Computing the mean engine power
```

```
meanPwr3b =
7.5150e+03
```

## TTC RESULTS

This section presents the extrapolated results concerning key performance indicators using different values of Kp, namely:

- Total fuel consumption
- Final State of Charge (SOC)
- Instantaneous engine power
- Average engine power

These metrics were analyzed to assess the effectiveness and consistency of the implemented control strategy under different operating conditions.

### Common Kp results

```
% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption_4a = trapz(fuelFlwRate4a)/1e3 % (Kg)
```

```
fuelConsumption_4a =
1.4470
```

```
finalSOC_4a = SOC4a(end); % Final value of SOC

% Mean engine power
engPower4a = [prof4a.engSpd] .* [prof4a.engTrq]; % Computing the instantaneous
engine power
meanPwr4a = mean(engPower4a) % Computing the mean engine power
```

```
meanPwr4a =
5.9250e+03
```

**Tuned Kp results**

```
% Calculate total fuel consumption using numerical integration (trapz)
fuelConsumption_4b = trapz(fuelFlwRate4b)/1e3 % (Kg)
```

```
fuelConsumption_4b =
1.5371
```

```
finalSOC_4b = SOC4b(end); % Final value of SOC

% Mean engine power
engPower4b = [prof4b.engSpd] .* [prof4b.engTrq]; % Computing the instantaneous
engine power
meanPwr4b = mean(engPower4b) % Computing the mean engine power
```

```
meanPwr4b =
6.1379e+03
```

## The A-ECMS controller

The variables within the controller can be classified as:

***Inputs:***

- *SOC*: Double type variable of current battery state of charge, a scalar between 0 and 1 [-], used to update the SOC next value and fill progressevely the SOC vector.
- *vehSpd*: Double type variable of current vehicle speed [m/s] used to compute the power demand at the cycle time instant.
- *vehAcc*: Double type variable of current vehicle acceleration [m/s^2] used to compute the power demand at the cycle time instant.
- Equivalence factor at time n - `eqFactor_n`
- Equivalence factor at time n-1 - `eqFactor_np`
- Multiplicative coefficient for adaptive update - Kp
- Structure containing vehicle parameters (engine map, battery specs, etc.) - veh

***Outputs Control Variables:***

- Engine Speed - *EngSpd [rad/s]*
- Engine Torque - *EngTrq [Nm]*
- Updated equivalence factor - `eqFactor_ns`

The main input/output differences from the previous controller can be showed by looking the function:

```
%function [engSpd, engTrq, eqFactor_ns] = a_ecmsControl(SOC, vehSpd, vehAcc,
eqFactor_n, eqFactor_np, Kp, veh)
```

The crucial passage of this new controller version can be expressed by the computation of this output variable presented at the beginning:

$$\mathrm{eqFact_{ns}} = \frac{(\mathrm{eqFact}_n + \mathrm{eqFact_{np}})}{2} + \mathrm{kp}\,(\mathrm{SOC_{in}} - \mathrm{SOC}_n)$$

- The equivalence factor trajectory is built combining past and current value of "s" so that its value can be stabilized.
- A correction term is added and it is scaled through the value of Kp

The controller and the simulations loops synergistically implement the following steps to permeform the A-ECMS simulations :

The simulation starts with two inputs values of `eqFactor`, named `eqFactor_n` and `eqFactor_np`, which are both initialized using the `eqFactor` value obtained in the previous project. These values are updated every 60 seconds. After each update, `eqFactor_n` is replaced by `eqFactor_np`, and the new value from the output `eqFactor_ns` is stored in `eqFactor_n` as an input.

As was discussed in the previous project, the code implements a controller based on the A-ECMS (Adaptive Equivalent Consumption Minimization Strategy) approach for the optimal management of the internal combustion engine in our hybrid vehicle model. The additional objective in this case is to determine a mechanism in order to perform the parameter adaptation based exclusively on feedback from SOC.

The main idea is to periodically refresh the control parameter "s" according to the current SOC value, so that the SOC is maintained as close as possible to the reference value, i.e. the initial one, and, so that both the minimization of the fuel consumption and the charge sustaining behaviour can be achieved approximating the calibration of the "s" value.

**Definition of Candidate Controls**

A discrete set of engine speed values is generated by initially adding the value 0 and then uniformly discretizing the interval between the idle speed *`w_idle`* and the maximum speed *`w_max`* using 20 equally spaced points (`linspace`). Similarly, the engine torque *`T_eng`* is discretized from 0 up to the maximum torque calculated over the engine speed range.

These values are then combined using a grid (`ndgrid`) to explore all possible control combinations.

**Equivalent Consumption Calculation**

For each candidate pair (w_eng,T_eng), the function `hev_cell_model` is called, returning:

- The future value of the state of charge (`SOC_next`)
- The instantaneous fuel consumption (`stageCost`)

- A feasibility indicator (`unfeas`)

The equivalent fuel consumption is calculated according to the formula:

$$\dot{m}_{f,\mathrm{eq}} = \dot{m}_f - s \cdot \mathrm{const} \cdot (\mathrm{SOC}_{\mathrm{next}} - \mathrm{SOC})$$

where `const` is an energy-to-fuel conversion factor that accounts for the nominal battery energy (E_b) and the lower heating value of the fuel (*Q_LHV*), and is given by: $\mathrm{const} = \frac{E_b}{Q_{\mathrm{LHV}}} \cdot 1000 \cdot 3600.$

**Constraint Handling**

To avoid physically infeasible solutions:

- Combinations that cause the SOC to exceed the desired limits [0.4, 0.8] are penalized by adding a very large numerical penalty to the equivalent cost.
- Combinations marked as infeasible by the model (`unfeas`) are discarded by assigning them an infinite value.
- Ensures battery protection is always enforced.

**Optimal Control Selection**

Finally, the combination (w_eng,T_eng) that minimizes the equivalent fuel consumption among all feasible options is selected.

```
function [engSpd, engTrq, eqFactor_ns] = a_ecmsControl(SOC, vehSpd, vehAcc,
eqFactor_n, eqFactor_np, Kp, veh)

SOC1 = 0.6; % Inizializing the starting value of SOC (60%)

% The value of the equivalent factor is updated by applying a penalty function
eqFactor_ns = (eqFactor_n + eqFactor_np)/2 + Kp * (SOC1 - SOC);

% Define the engine speed boundaries
w_idle = veh.eng.idleSpd; % Engine idle speed (rad/s)
w_max = veh.eng.maxSpd; % Maximum engine speed (rad/s)

% Generate a discrete set of engine speed values:
% the first value is 0, followed by 20 equally spaced points from w_idle to w_max
w_speed = [0 linspace(w_idle, w_max, 20)]; % (rad/s)

% Compute the maximum engine torque corresponding to the defined speed set
T_max = max(veh.eng.maxTrq(w_speed)); % (Nm)

% Generate 21 equally spaced torque values from 0 to T_max
T_eng = linspace(0, T_max, 21); % (Nm)

% Create a full grid of all (w_eng, T_eng) control combinations
[w_set,T_set] = ndgrid(w_speed,T_eng);
```

```
% Evaluate the HEV model for each control combination:
% returns:
% - SOC_next: predicted future value of the State of Charge
% - stageCost: fuel consumption rate (g/s)
% - unfeas: mask indicating infeasible control combinations
[SOC_next, stageCost, unfeas, ~] = hev_cell_model({SOC}, {w_set, T_set}, {vehSpd,
vehAcc}, veh) ;

% Compute the constant used to convert battery energy change to equivalent fuel
const = ((veh.batt.nomEnergy)/veh.eng.fuelLHV) * 1000 * 3600; % (g)

% Replicate current SOC to match the dimensions of the cost array
SOC = SOC * ones(size(stageCost)); % (-)

% Compute the equivalent fuel consumption:
% actual fuel flow minus electric contribution scaled by equivalence factor
fuelFlwRate_eq = stageCost - eqFactor_n * const * (SOC_next{1} -SOC); % (g/s)

% Penalize controls that cause SOC to exceed safe operational boundaries
fuelFlwRate_eq(SOC_next{1} < 0.4 | SOC_next {1} > 0.8) = fuelFlwRate_eq ( SOC_next
{1} < 0.4 | SOC_next {1} > 0.8) + 10^6;

 % Set cost to infinity for infeasible control combinations
fuelFlwRate_eq (unfeas == 1) = inf;

% Find the control combination that minimizes equivalent fuel consumption
[min_val, idx] = min(fuelFlwRate_eq, [], 'all');

min_fuelFlwRate_eq = min_val;

[a,b] = ind2sub(size(fuelFlwRate_eq), idx);

% Return the optimal engine speed and torque
engSpd = w_speed(a);
engTrq = T_eng(b);
end
```

## The Improved A-ECMS Controller (AMDC cycle specific)

This alternative control strategy allows refining the logic of the A-ECMS controller by implementing a different criterion through updating the equivalent factor and improve the controller behaviour.

As already explained, the original controller failed to operate properly in the AMDC cycle because there was no way to tune the Kp value, leading either to a malfunction of the optimization strategy or to a substantial deviation of the final SOC. This demonstrated the limited effectiveness of the SOC feedback in periodically correcting the

equivalent factor value and thus ensuring a sufficiently accurate approximation through some oscillating values, showing that it is not always possible to find a suitable Kp value that properly scales the SOC deviation error.

The main issue with this approach, according to our analysis, can be traced back to the nature of the delta SOC feedback. Referring to the SOC evolution obtained using a tuned equivalence factor (for example, found by the bisectional algorithm previously exploited), it can be observed that the correct use of the optimization strategy involves a continuous variation of the SOC value over time, rather than a constant value. This implies that comparing the current SOC with the initial one produces a feedback error that is almost always inappropriate, or rather, accounts for some deviations that in reality should be accepted, resulting in an excessive increase or decrease of the equivalent factor.

More specifically, it can be noticed that during simulations where the ECMS control strategy is properly applied, the SOC evolution shows more likely a positive deviation (deltaSOC < 0) during regenerative braking phases, and a negative deviation (deltaSOC > 0) during relatively low power demand phases in which the vehicle travels in Pure Electric mode (PE). The Battery Charging mode (BC) occurs more likely during low/medium power scenarios and, of course, causes a positive SOC deviation as well.

The idea is to build a filter which amplify the deltaSOC feedback when the State of Charge is more likely constant and close to 0.6, that is, when the power demand is higher, and, instead, to attenuate the deltaSOC feedback during low/medium and negative power demand, where the SOC deviation should not be interpreted as a major mistake.

The higher power scenarios correspond to situations where the internal combustion engine works at higher load and tries to meet the requested power alone, so the battery neither recharges nor discharges, and the SOC deviation can be more properly perceived as a feedback error.

Regarding the controller implementation, it has been made without changing any input or output variables and uses the same optimization logic but with a different adaptation criterion compared to before: a new parameter has been introduced, and the equivalent factor updating now follows the logic explained so far.

The new parameter is a power ratio (pwr_ratio) that normalizes and divides by 2 the demanded power with respect to a reference power specific to the AMDC cycle. This reference power has been computed considering the speed and acceleration observed in the SOC evolution with calibrated equivalent factor so that the curve crosses SOC = 0.6 roughly in the middle of the simulation.

Both this reference power and the current power have been computed through the hev_drivetrain function; for the latter, the absolute value is taken to avoid amplifying negative power scenarios. The Kp scaling is finally made through a straight line crossing the origin with a slope equal to the Kp value, from which the updated equivalent factor can then be computed.

This improvement leads to increased complexity and makes Kp tuning more difficult: the reference power should be approximated or computed somehow, and Kp now appears more sensitive since it influences both the multiplication factor and the line's slope. The improvement works specifically for this cycle and has proven difficult to extend to other cycles without additional refinement.

```
function [engSpd, engTrq, eqFactor_ns] = improved_a_ecmsControl(SOC, vehSpd,
vehAcc, eqFactor_n, eqFactor_np, Kp, veh)

SOC1 = 0.6;  % Inizializing the starting value of SOC (60%)
```

```
% Compute the shaft speed and torque based on current vehicle speed and acceleration
[shaftSpd, shaftTrq] = hev_drivetrain(vehSpd, vehAcc, veh);

% Compute shaft speed and torque for a reference operating condition (SOC = 0.6)
[shaftSpd0, shaftTrq0] = hev_drivetrain(33.9, 0.277, veh);

% Calculate the current power demand at the wheels (absolute value)
PwrDemand = abs(shaftTrq*shaftSpd);

% Calculate the power demand at the wheels of the reference operating condition
PwrDemand0 = shaftTrq0*shaftSpd0;

% Compute a normalized power ratio used to scale the control response
pwr_ratio = (PwrDemand/PwrDemand0)/2;

% The value of the equivalent factor is updated by applying a penalty function
eqFactor_ns = (eqFactor_n + eqFactor_np)/2 + Kp * (SOC1 - SOC)*(Kp*pwr_ratio);

% Define the engine speed boundaries
w_idle = veh.eng.idleSpd; % Engine idle speed (rad/s)
w_max = veh.eng.maxSpd; % Maximum engine speed (rad/s)

% Generate a discrete set of engine speed values:
% the first value is 0, followed by 20 equally spaced points from w_idle to w_max
w_speed = [0 linspace(w_idle, w_max, 20)]; % (rad/s)

% Compute the maximum engine torque corresponding to the defined speed set
T_max = max(veh.eng.maxTrq(w_speed)); % (Nm)

% Generate 21 equally spaced torque values from 0 to T_max
T_eng = linspace(0, T_max, 21); % (Nm)

% Create a full grid of all (w_eng, T_eng) control combinations
[w_set,T_set] = ndgrid(w_speed,T_eng);

% Evaluate the HEV model for each control combination:
% returns:
% - SOC_next: predicted future value of the State of Charge
% - stageCost: fuel consumption rate (g/s)
% - unfeas: mask indicating infeasible control combinations
[SOC_next, stageCost, unfeas, ~] = hev_cell_model({SOC}, {w_set, T_set}, {vehSpd,
vehAcc}, veh) ;

% Compute the constant used to convert battery energy change to equivalent fuel
const = ((veh.batt.nomEnergy)/veh.eng.fuelLHV) * 1000 * 3600; % (g)

% Replicate current SOC to match the dimensions of the cost array
SOC = SOC * ones(size(stageCost)); % (-)
```

```
% Compute the equivalent fuel consumption:
% actual fuel flow minus electric contribution scaled by equivalence factor
fuelFlwRate_eq = stageCost - eqFactor_n * const * (SOC_next{1} -SOC); % (g/s)

% Penalize controls that cause SOC to exceed safe operational boundaries
fuelFlwRate_eq(SOC_next{1} < 0.4 | SOC_next {1} > 0.8) = fuelFlwRate_eq ( SOC_next
{1} < 0.4 | SOC_next {1} > 0.8) + 10^6;

 % Set cost to infinity for infeasible control combinations
fuelFlwRate_eq (unfeas == 1) = inf;

% Find the control combination that minimizes equivalent fuel consumption
[min_val, idx] = min(fuelFlwRate_eq, [], 'all');

min_fuelFlwRate_eq = min_val;

[a,b] = ind2sub(size(fuelFlwRate_eq), idx);

% Return the optimal engine speed and torque
engSpd = w_speed(a);
engTrq = T_eng(b);
end
```