

University of Wales Trinity Saint David

Institute of Science & Art



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Alternative Propulsion

ACHA 5010

Michael Charity

2100477

Jan 2024

Contents

1	Introduction	1
2	Specifications	2
2.1	Frontal area	2
2.2	Rolling resistance.....	2
2.3	Drag Coefficient	2
2.4	Test mass	3
2.5	Variables and WLTP Class	4
3	Cumulative Tracking and Braking Energy.....	5
3.1	Force Calculations:.....	5
4	Maximum moments of energy recovery	7

1 Introduction

The chosen vehicle is the Alfa Romeo 159 3.2 V6. However, it has a problem in the modern world. It is really very uneconomical. It is the experience of the author that it is impossible to get more than 25mpg at motorway speeds, and when driving spiritedly values as low as 4.5 mpg. It's no wonder the tax is so expensive. Therefore a boost in fuel economy would be beneficial, then these slightly more exciting things could be kept.

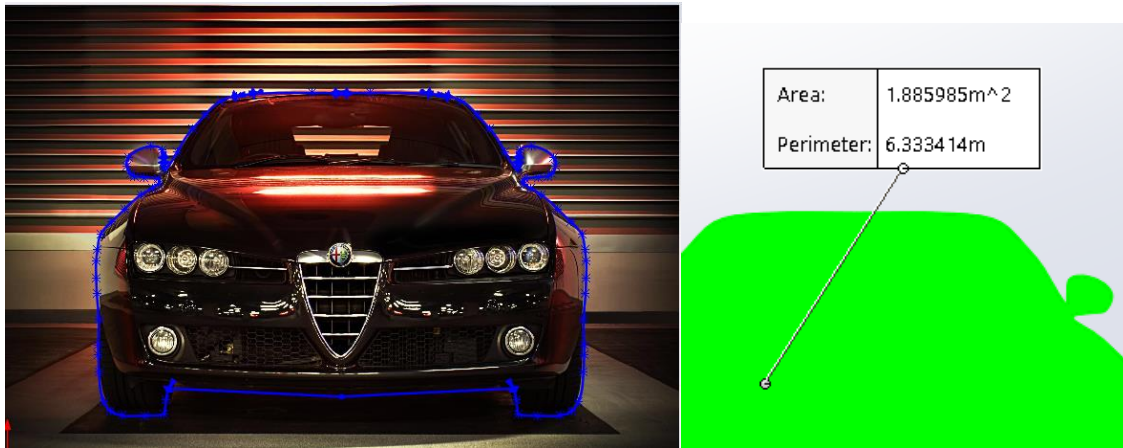
But the modern world is making these things harder and harder to have, not just in the case of the Alfa but across the board. As you are making us aware through this module, we are facing ever-tightening emissions regulations and growing pressure to move away entirely from combustion engines. The Euro 4 regulations already forced the famous 'Busso' V6 to stop being produced, leading to Alfa Romeo using The GM engine found in the 159, now that really is a shame. So, what can be done...

2 Specifications

To Calculate the WLTP cycle for the 159 from the velocity and acceleration data test mass, frontal area, drag coefficient and rolling resistance coefficient must be found.

2.1 Frontal area

Frontal area was estimated using know dimensions of the car and an image of the front view.



2.2 Rolling resistance

The rolling resistance coefficient (C_{rr}) for a specific tyre model, namely the Pirelli P7. This tyre is characterized by its EU rolling resistance rating of 'C'. According to the regulatory framework set forth by Regulation (EU) 2020/740 of the European Parliament and of the Council, issued in 2020, tyre rolling resistance classifications are standardized to facilitate comparisons and enhance transparency for consumers (EU, 2020).

The EU rating system categorizes tyres based on their rolling resistance coefficient (RRC), measured in Newtons per kilonewton (N/kN). The classification ranges from 'A' (lowest rolling resistance) to 'G' (highest rolling resistance), with each class corresponding to specific RRC value ranges. For a tyre rated as class 'C', the lower boundary of the RRC is specified as 7.8 N/kN, and the upper boundary is 9 N/kN.

Given the absence of more specific data regarding the Pirelli P7's exact RRC within the 'C' classification, the upper value within this range was utilized for subsequent analyses. This approach ensures a balanced representation of the tyre's performance characteristics under the constraints of the available data.

2.3 Drag Coefficient

Drag Coefficient has been referenced as 0.33 (Carinf, 2023)

2.4 Test mass

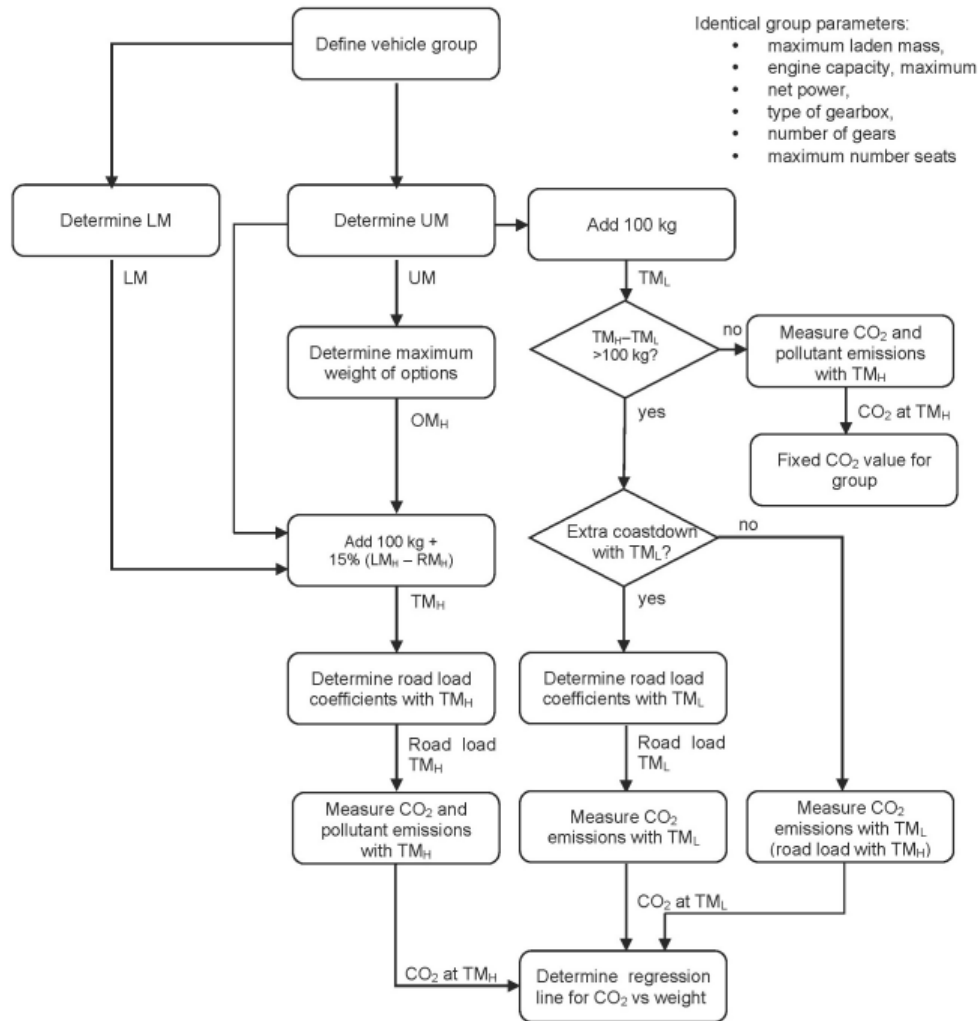


Fig 3. Test mass procedure, (Mock, 2011)

Determining the test mass for the heaviest vehicle (TMh) and the lightest vehicle (TMl) as per the document involves a process designed to account for variations in vehicle weight due to optional equipment and other variables. The process is as follows as well as shown above in Fig.3

For TMh:

- Start with the unladen mass (UM) of a vehicle from a group with similar specs.
- Add the maximum mass of all optional equipment (OMh) to UM.
- Include an additional 100 kg for a standard driver and some luggage to get the reference mass (RMh).
- For M1 vehicles, add 15% of the difference between the maximum laden mass (LM) and RMh to UM + OMh + 100 kg for TMh.

For TMl:

- Use the same UM as for TMh.

- Add 100 kg for the driver and luggage, plus the variable weight calculated for TMh.

The sum gives TMI, representing the vehicle's base weight with standard load, excluding optional equipment.

The process for determining TMh and TMI is designed to accurately reflect the range of weights a vehicle might have in real-world conditions, taking into account variations from optional equipment and other variable loads. TMh is used to simulate the scenario where a vehicle is equipped with all possible optional features, along with additional loads, while TMI represents the base vehicle weight plus a standard load without any optional equipment. This approach allows for a more tailored and accurate assessment of vehicle performance, particularly in terms of emissions testing.

For the purposes of the calculations, the curb weight of 1655kg along with the additional 100kg is used to give TMI. It is not known as to the weight of the optional extras.

2.5 Variables and WLTP Class

The Alfa Romeo 159 has a rated power of 191 Kw. Based on the power to weight and velocity class 3 was selected.

Below in Fig. 4 shows the variables set and file loading.

```
[20]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Constants and Vehicle Specifications
vehicle_mass = 1755 # kg
CD = 0.33 # Drag Coefficient
air_density = 1.225 # kg/m³
frontal_area = 1.88 # m²
C_rr = 0.009 # Coefficient of Rolling Resistance
g = 9.81 # m/s²
# Motor-Generator and Battery Efficiency
motor_generator_efficiency = 0.85 # Efficiency of the motor-generator (0.85 = 85%)
battery_charge_efficiency = 0.90 # Efficiency of charging the battery (0.90 = 90%)

# Load the data from the uploaded file
file_path = "C:\\Users\\mikec\\OneDrive\\Desktop\\wltp class 3.xlsx"
data = pd.read_excel(file_path, skiprows=6) # Read the data, skipping non-data rows

# Select relevant columns for Time, Velocity, and Acceleration
data = data[["s", "km/h", "m/s²"]].copy()
data.columns = ["Time", "Velocity_kmh", "Acceleration"]

# Calculate Velocity in m/s
data['Velocity_ms'] = data['Velocity_kmh'] * 1000 / 3600
```

Fig.4 variables and file loading

3 Cumulative Tracking and Braking Energy.

3.1 Force Calculations:

Rolling Resistance Force is calculated using the coefficient of rolling resistance (C_{rr}), the vehicle mass, and gravitational acceleration (g). This force opposes the motion of the vehicle and is constant under uniform conditions.

Aerodynamic Drag Force depends on the vehicle's velocity squared, air density, drag coefficient (CD), and frontal area. It increases with the square of the vehicle's speed.

Acceleration Force is calculated from the vehicle's mass and acceleration ($F = ma$). It's positive when the vehicle accelerates and negative during deceleration (braking).

Total Force combines rolling resistance, drag, and acceleration forces to represent the overall force acting on the vehicle at each moment. This is shown by Fig. 5 and the graph Fig. 6.

Fig. 7 then shows the plot of power over time,

```
# Calculate Rolling Resistance Force and Aerodynamic Drag Force
data['RollingResistanceForce'] = C_rr * vehicle_mass * g
data['DragForce'] = 0.5 * air_density * CD * frontal_area * data['Velocity_ms']**2

# Calculate Total Resistance
data['TotalResistance'] = data['RollingResistanceForce'] + data['DragForce']

# Calculate Acceleration Force (positive for acceleration, negative for deceleration)
# F=ma
data['AccelerationForce'] = vehicle_mass * data['Acceleration']

# Calculate Total Force (including acceleration/deceleration)
data['TotalForce'] = data['TotalResistance'] + data['AccelerationForce']

# Calculate Total Power (positive for acceleration, negative for deceleration)
data['TotalPower'] = data['TotalForce'] * data['Velocity_ms']
```

Fig 5. Total force calculation.

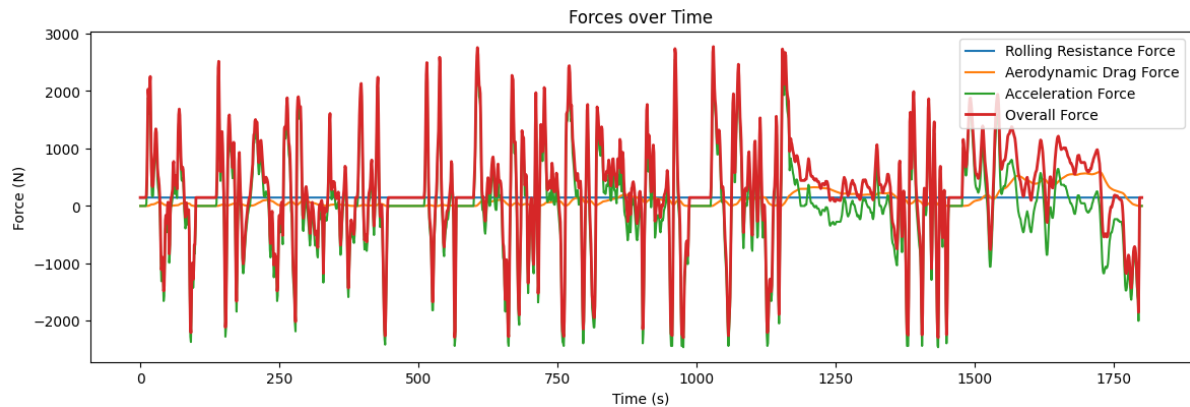
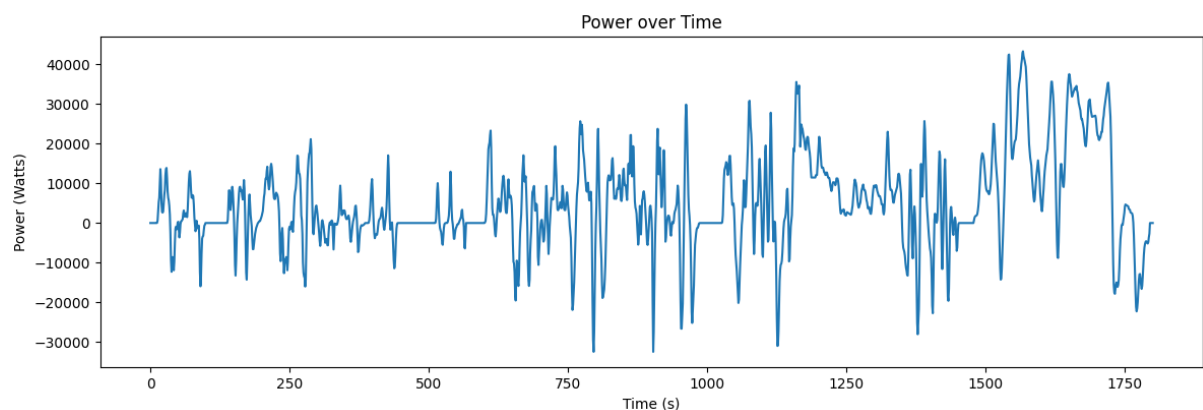


Fig. 6 Force over time (above)

Fig. 7 Power over time (below)



Energy at each time step is essentially the power at that moment since power is energy per unit time, This gives the energy in Joules.

Cumulative Energy is calculated by cumulatively summing the energy over time. This provides a running total of the energy consumed or regenerated as the vehicle moves.

From this the positive and negative values were separated giving cumulative braking and tractive energy. Fig. 8 and Fig. 9

```
# Calculate Energy per Time Step (Joules) - with a 1-second interval, it's equal to the power
data['Energy'] = data['TotalPower'] # Because Power is in Watts and Time step is 1 second

# Calculate Cumulative Energy over Time (Joules)
data['CumulativeEnergy'] = data['Energy'].cumsum()

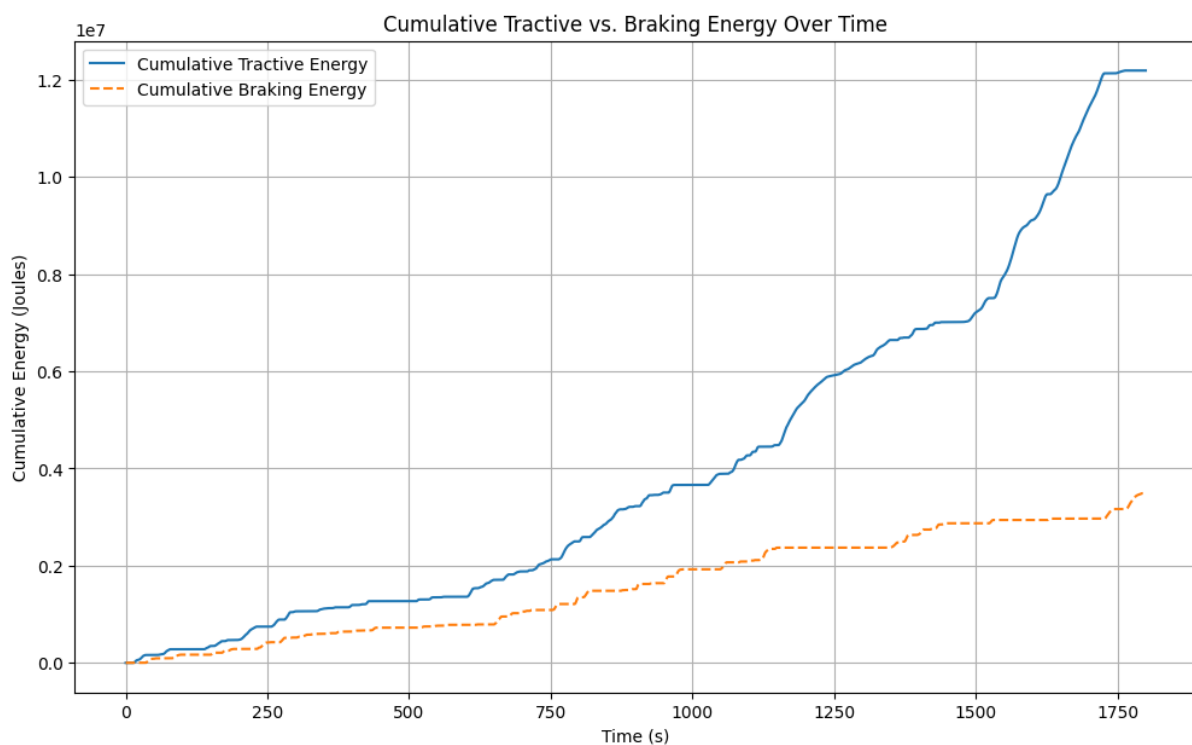
# Initialize 'TractiveEnergy' and 'BrakingEnergy' columns with default float values
data['TractiveEnergy'] = 0.0
data['BrakingEnergy'] = 0.0

# Assign values to 'TractiveEnergy' and 'BrakingEnergy'
# You do not need to cast these columns to float64 after initialization since they already contain float values
data.loc[data['TotalPower'] > 0, 'TractiveEnergy'] = data['TotalPower']
data.loc[data['TotalPower'] < 0, 'BrakingEnergy'] = -data['TotalPower']

# Calculate Cumulative Tractive and Braking Energy
data['CumulativeTractiveEnergy'] = data['TractiveEnergy'].cumsum()
data['CumulativeBrakingEnergy'] = data['BrakingEnergy'].cumsum()
```

Fig. 8 Energy calculations (above)

Fig.9 Cumulative energy (below)



4 Maximum moments of energy recovery

To Identify the maximum moments of energy recovery, firstly this was separated into hay many watts were being produced, equal to the steepest gradient of the braking energy plot.

Secondly for each period of braking energy how much total energy was available.

This was calculated as shown in Fig. 10 and resulted in Fig. 11 and 12. This resulted in a maximum instantaneous power availability of 12.85Kw and the largest segment of recoverable energy of 352978J

```
# Assuming 'data' is your DataFrame and 'CumulativeBrakingEnergy' is calculated
data['BrakingEnergyGradient'] = np.gradient(data['CumulativeBrakingEnergy'])

# Initialize segment identification
data['RecoverySegment'] = 0
segment_id = 0
for i in range(1, len(data)):
    if data.loc[i, 'BrakingEnergyGradient'] > 0:
        if data.loc[i-1, 'BrakingEnergyGradient'] <= 0:
            segment_id += 1
            data.loc[i, 'RecoverySegment'] = segment_id
        else:
            data.loc[i, 'RecoverySegment'] = 0

# Sum energy by segment
recovery_segments = data[data['RecoverySegment'] > 0]
total_energy_by_segment = recovery_segments.groupby('RecoverySegment')['BrakingEnergy'].sum()

# Identify top segments
top_recovery_segments = total_energy_by_segment.nlargest(3)
print(top_recovery_segments)

# Assuming 'data' and 'top_instant_power' are already defined

plt.figure(figsize=(14, 7))

# Cumulative Braking Energy Graph
plt.plot(data['Time'], data['CumulativeBrakingEnergy'], label='Cumulative Braking Energy', color='blue', alpha=0.5)

# Highlighting Top Instantaneous Power Recovery Points
for index, row in top_instant_power.iterrows():
    plt.scatter(row['Time'], data.at[index, 'CumulativeBrakingEnergy'], color='red', zorder=5)
    plt.text(row['Time'], data.at[index, 'CumulativeBrakingEnergy'],
            f'{row["InstantaneousPower_KW"]:.2f} kW', color='red', verticalalignment='bottom')

plt.xlabel('Time (s)')
plt.ylabel('Cumulative Braking Energy (Joules)')
plt.title('Cumulative Braking Energy & Instantaneous Power Recovery Points')
plt.legend()
plt.grid(True)
plt.show()

# Assuming 'data' is your DataFrame with 'Time' and 'CumulativeBrakingEnergy' among the columns,
# and 'RecoverySegment' indicating the recovery segments
plt.figure(figsize=(14, 7))

# Plot the cumulative braking energy over time
plt.plot(data['Time'], data['CumulativeBrakingEnergy'], label='Cumulative Braking Energy', color='blue', alpha=0.5)

# Highlight the top recovery segments
for segment_id in top_recovery_segments.index:
    segment_data = data[data['RecoverySegment'] == segment_id]
    plt.fill_between(segment_data['Time'], segment_data['CumulativeBrakingEnergy'], color='green', alpha=0.3)

# Additional plot formatting
```

Fig. 10

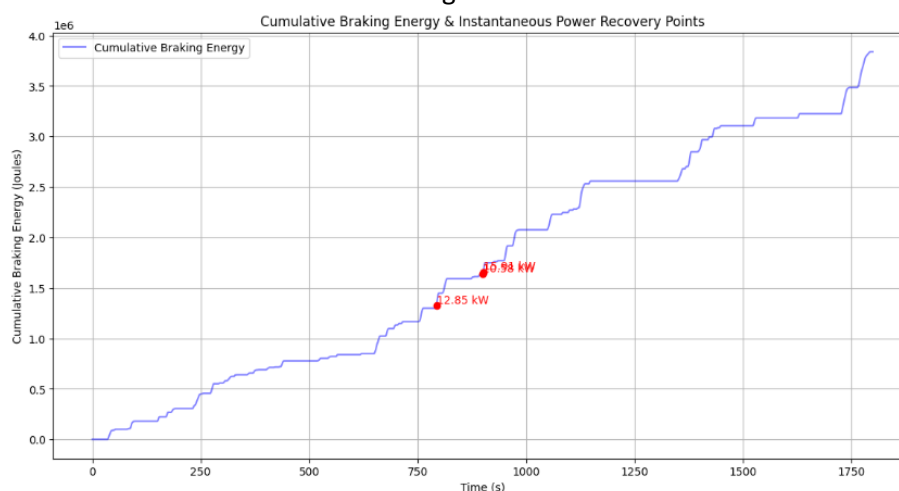


Fig. 11 Maximum instantaneous power recovery.

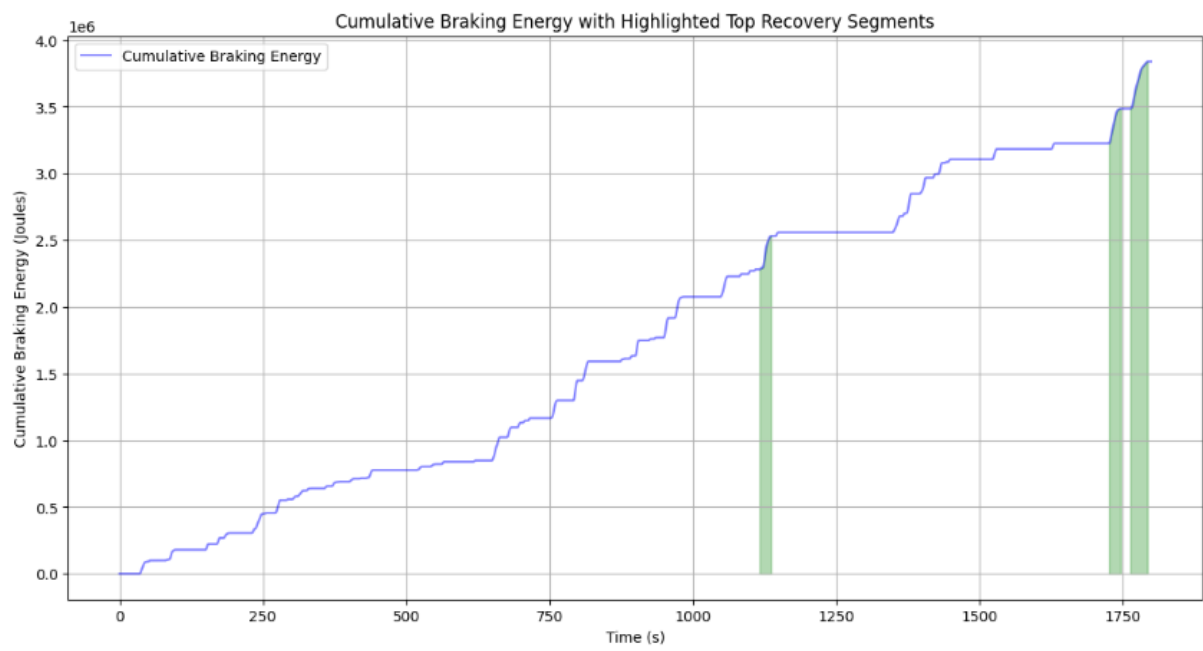


Fig. 12 Segments of greatest recoverable energy

5 Kers Specification

The Kinetic Energy Recovery System (KERS) is a mechanism for recovering the kinetic energy that would otherwise be lost to heat during braking. This energy is stored in a form that can be later used to propel the vehicle, improving fuel efficiency and reducing emissions. A suitable KERS for a passenger vehicle typically includes components such as an energy storage device (e.g., a flywheel, batteries, or supercapacitors), a motor-generator unit, power electronics, and a control system.

Energy Storage Device:

- **Flywheel:** A flywheel system stores kinetic energy mechanically. It is ideal for applications requiring high power density and short energy storage duration. Its efficiency can vary but 90% is typical (Choudhury, 2021), depending on the materials and bearing design.
- **Batteries:** Battery storage offers high energy density and is suitable for longer energy storage durations. The overall efficiency (charge/discharge cycle) can range from 80% to 95% for modern lithium-ion batteries. (Choudhury, 2021)
- **Supercapacitors:** Supercapacitors provide very high power density and rapid charge/discharge cycles, with efficiencies around 95% to 99%. However, they have lower energy density compared to batteries (Pattnaik, 2022)

For this system a conventional hybrid system will be implemented owing to its current relevancy to the automotive industry and the availability of off the shelf part.

The requirements this system must meet for the WLTP test cycle would be.

- Peak power required is around 45KW
- Maximum power available during regenerative braking is 12.85Kw
- Maximum battery size of 3.6MJ
- Duty cycle of motor/generator and componentry of 250 seconds or more.

- **Motor-Generator Unit Specification**

Given the calculated requirements, selecting a Compro REB 60 motor-generator unit capable of producing a peak power of 60 kW far exceeds the identified maximum regenerative power input of 12.85 kW during braking and aligns well with the 45 kW maximum output derived from our analysis, 45kW being the maximum sustained power the motor will provide (compro, 2023). This unit's efficiency is in the higher spectrum, around 90-95%, suitable for both propulsion and energy recovery modes. (compro, 2023)

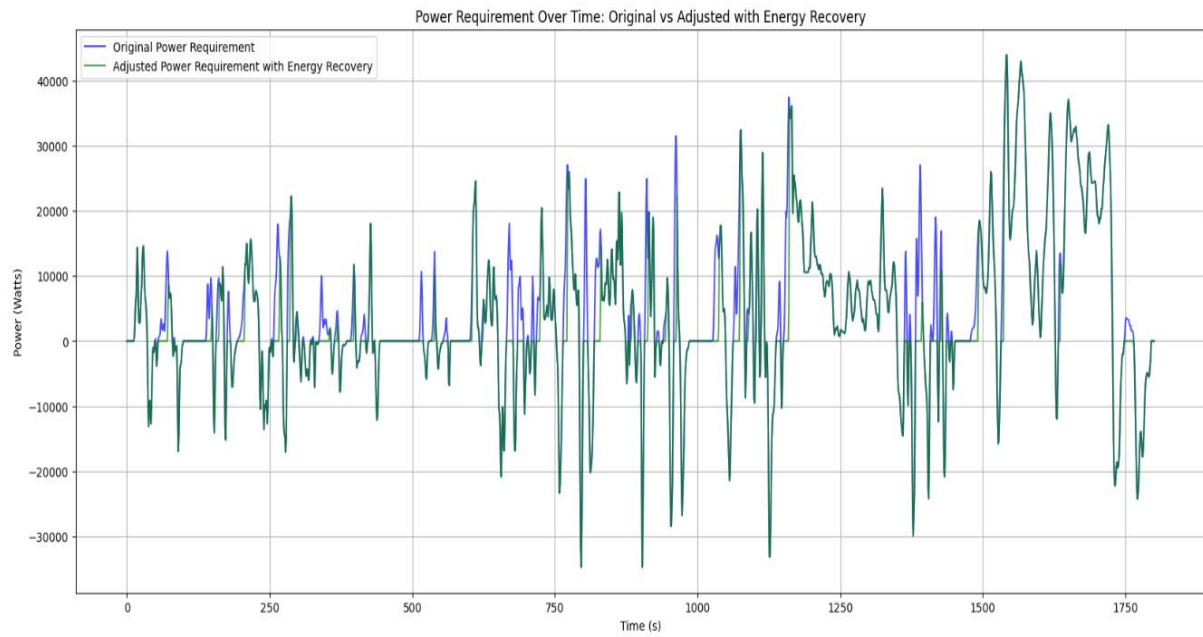
- **Battery System and Electrical Architecture**

The battery system, pivotal in storing the energy recaptured during regenerative braking, is determined to require a capacity of approximately 1.75 MJ. This calculation tailors the system for effective energy management within the vehicle's duty cycle while considering the dynamic power requirements during the WLTP test cycle.

- **Battery Type:** Utilizing LG 18650HE2 lithium-ion batteries forms the basis of the energy storage solution. These batteries are known for their robust discharge characteristics and thermal stability.

- **Capacity and Working Voltage:** With a total energy capacity of 1.75 MJ, the system needs to be designed with a safe operating voltage in mind. Considering the characteristics of the chosen batteries, the system's working voltage should be designed to optimize the balance between energy density and power delivery, ensuring that the battery pack operates within efficient voltage ranges throughout the discharge curve.
- **Efficiency Considerations:** The charge and discharge efficiencies of these batteries hover around 90-95%. To maintain these efficiencies, a battery management system (BMS) is essential for monitoring cell health, temperature, and voltage levels, ensuring the system remains within optimal operating conditions.
- **Power Electronics and Control System**
- **Power Electronics:** Key components like inverters and converters must be chosen for their ability to efficiently manage the flow of high currents at the system's operating voltage. For this application, components capable of handling the sudden surges in power associated with regenerative braking and rapid acceleration are selected, aiming for efficiencies of 95% or higher to minimize energy losses.
- **Control System:** An advanced control module, capable of real-time decision-making based on vehicle dynamics, battery state, and driver inputs, orchestrates the operation of the KERS. The system prioritizes energy recovery, efficient power use, and seamless transition between propulsion modes.
- **Operational Insights and Driver Experience**
- **Plug-in Hybrid Advantage:** Emphasizing a plug-in hybrid configuration allows the vehicle to start with a full charge, significantly extending the electric-only driving range and enhancing fuel savings for typical short trips. This system flexibility caters to a wide range of driving habits and reduces reliance on the ICE, especially for urban driving.
- **Power Distribution:** Transitioning to a configuration where the ICE powers the rear wheels and the electric motor drives the front introduces a novel power distribution method. This setup not only increases the total power output to 210 kW but also offers an enhanced driving experience with improved traction and handling, attributed to the balanced power delivery.

Fig. 12 WLTP cycle for minimum battery requirements



Above in Fig. 12 shows WLTP shows ICE output adjusted for the hybrid system in green. This saves a total of 250KW throughout the WLTP.

6 Conclusion

Specifying a KERS for a traditional hybrid vehicle, especially with a focus on a plug-in hybrid system, involves intricate planning around the motor-generator unit, battery system, power electronics, and control strategies. By selecting specific components like the Compro REB 60 motor and LG 18650HE2 lithium-ion batteries, the system is tailored to meet the dynamic requirements of the WLTP test cycle while offering an enhanced driving experience. Such a system not only improves fuel efficiency and reduces emissions but also provides a smoother and more responsive ride, aligning with modern expectations for hybrid vehicle performance and sustainability.

Should a plug-in hybrid option be selected this would come at the cost of a larger battery and adaptations to the battery monitoring system. From market research of plug in hybrids battery capacity can vary from 10KWH (36MJ) to around 50KWH (180MJ).

The calculations in the code a battery capacity of 12MJ was returned which would be enough to conduct the WLTP which covers a distance of 23.25KM and would seem to meet personal requirements in order to use the vehicle on full electric propulsion.

Based on UK prices for fuel and electricity (dec 2024, £1.45/L and £0.29/KWH) this would result the total cost of the WLTP journey costing 94p, opposed to £3.06 originally (25mpg assumed).

This would offer a vastly more practical and economical solution in the long run.

7 References

REGULATION (EU) 2020/740 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL, 2020, available at <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32020R0740>

Carinf, DOA 28/12/23, available at <https://www.carinf.com/en/68f1015616.html>

P Mock, 2011, Development of a Worldwide Harmonized Light Vehicles Test Procedure (WLTP)

S Choudhury, 2021, Flywheel energy storage systems: A critical review on technologies, applications, and future prospects

S Pattnaik, 2022, A review on characterization of supercapacitors and its efficiency analysis for different charging methods and applications

Compro, DOA 28/12/23. Available at <https://www.mgm-compro.com/electric-motor/60-kw-electric-motor/>

8 Appendix

Full script below, largely the same file that had been submitted previously

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


# Constants and Vehicle Specifications

vehicle_mass = 1755 # kg

CD = 0.33 # Drag Coefficient

air_density = 1.225 # kg/m³

frontal_area = 1.88 # m²

C_rr = 0.009 # Coefficient of Rolling Resistance

g = 9.81 # m/s²

# Motor-Generator and Battery Efficiency

motor_generator_efficiency = 0.85 # Efficiency of the motor-generator (0.85 = 85%)

battery_charge_efficiency = 0.90 # Efficiency of charging the battery (0.90 = 90%)


# Load the data from the uploaded file

file_path = "C:\\Users\\mikec\\OneDrive\\Desktop\\wlrc class 3.xlsx"

data = pd.read_excel(file_path, skiprows=6) # Read the data, skipping non-data rows


# Select relevant columns for Time, Velocity, and Acceleration

data = data[["s", "km/h", "m/s²"]].copy()

data.columns = ["Time", "Velocity_kmh", "Acceleration"]


# Calculate Velocity in m/s

data["Velocity_ms"] = data["Velocity_kmh"] * 1000 / 3600


# Calculate change in distance for each 1-second time step

data["Change_in_Distance"] = 0.5 * data["Acceleration"] * (1**2)


# Giving me velocity, acceleration and change in distance.


# Calculate Rolling Resistance Force and Aerodynamic Drag Force

data["RollingResistanceForce"] = C_rr * vehicle_mass * g

data["DragForce"] = 0.5 * air_density * CD * frontal_area * data["Velocity_ms"]**2


# Calculate Total Resistance

data["TotalResistance"] = data["RollingResistanceForce"] + data["DragForce"]


# Calculate Acceleration Force (positive for acceleration, negative for deceleration)

# F=MA

data["AccelerationForce"] = vehicle_mass * data["Acceleration"]


# Calculate Total Force (including acceleration/deceleration)

data["TotalForce"] = data["TotalResistance"] + data["AccelerationForce"]


# Calculate Total Power (positive for acceleration, negative for deceleration)

data["TotalPower"] = data["TotalForce"] * data["Velocity_ms"]
```

```

# Plotting

plt.figure(figsize=(12, 20))

# Plot Velocity

plt.subplot(5, 1, 1)

plt.plot(data['Time'], data['Velocity_ms'])

plt.xlabel('Time (s)')

plt.ylabel('Velocity (m/s)')

plt.title('Velocity over Time')

# Plot Acceleration

plt.subplot(5, 1, 2)

plt.plot(data['Time'], data['Acceleration'])

plt.xlabel('Time (s)')

plt.ylabel('Acceleration (m/s²)')

plt.title('Acceleration over Time')

# Plot Forces

plt.subplot(5, 1, 3)

plt.plot(data['Time'], data['RollingResistanceForce'], label='Rolling Resistance Force')

plt.plot(data['Time'], data['DragForce'], label='Aerodynamic Drag Force')

plt.plot(data['Time'], data['AccelerationForce'], label='Acceleration Force')

plt.plot(data['Time'], data['TotalForce'], label='Overall Force', linewidth=2)

plt.xlabel('Time (s)')

plt.ylabel('Force (N)')

plt.title('Forces over Time')

plt.legend()

# Plot Power

plt.subplot(5, 1, 4)

plt.plot(data['Time'], data['TotalPower'])

plt.xlabel('Time (s)')

plt.ylabel('Power (Watts)')

plt.title('Power over Time')

plt.tight_layout()

plt.show()

# Calculate Energy per Time Step (Joules) - with a 1-second interval, it's equal to the power

data['Energy'] = data['TotalPower'] # Because Power is in Watts and Time step is 1 second

# Calculate Cumulative Energy over Time (Joules)

data['CumulativeEnergy'] = data['Energy'].cumsum()

# Initialize 'TractiveEnergy' and 'BrakingEnergy' columns with default float values

data['TractiveEnergy'] = 0.0

data['BrakingEnergy'] = 0.0

# Assign values to 'TractiveEnergy' and 'BrakingEnergy'

# You do not need to cast these columns to float64 after initialization since they already contain float values

data.loc[data['TotalPower'] > 0, 'TractiveEnergy'] = data['TotalPower']

```

```

data.loc[data['TotalPower'] < 0, 'BrakingEnergy'] = -data['TotalPower']

# Calculate Cumulative Tractive and Braking Energy

data['CumulativeTractiveEnergy'] = data['TractiveEnergy'].cumsum()

data['CumulativeBrakingEnergy'] = data['BrakingEnergy'].cumsum()

# Plotting Cumulative Energy over Time

plt.figure(figsize=(10, 6))

plt.plot(data['Time'], data['CumulativeEnergy'], label='Cumulative Energy')

plt.xlabel('Time (s)')

plt.ylabel('Cumulative Energy (Joules)')

plt.title('Cumulative Energy Consumption/Regeneration Over Time')

plt.legend()

plt.grid(True)

plt.show()

# Now, you can safely plot the results without encountering the KeyError

# Plotting

plt.figure(figsize=(12, 7))

plt.plot(data['Time'], data['CumulativeTractiveEnergy'], label='Cumulative Tractive Energy')

plt.plot(data['Time'], data['CumulativeBrakingEnergy'], label='Cumulative Braking Energy', linestyle='--')

plt.xlabel('Time (s)')

plt.ylabel('Cumulative Energy (Joules)')

plt.title('Cumulative Tractive vs. Braking Energy Over Time')

plt.legend()

plt.grid(True)

plt.show()

# Calculate the gradient of BrakingEnergy to find instantaneous power recovery in kW

# This assumes BrakingEnergy is already in your DataFrame and represents the energy in Joules

data['InstantaneousPower_kW'] = np.gradient(data['BrakingEnergy'], edge_order=2) / 1000 # Converting to kW from Watts

# Calculate the difference in cumulative braking energy to identify increases

data['CumulativeBrakingEnergyDiff'] = data['CumulativeBrakingEnergy'].diff()

# Top 3 Instantaneous Power Recovery in kW and their times

top_instant_power = data.nlargest(3, 'InstantaneousPower_kW')[['Time', 'InstantaneousPower_kW']]

# Top 3 Largest Increases in Cumulative Braking Energy and their times

top_energy_recovery = data.nlargest(3, 'CumulativeBrakingEnergyDiff')[['Time', 'CumulativeBrakingEnergyDiff']]

# Assuming 'data' is your DataFrame and 'CumulativeBrakingEnergy' is calculated

data['BrakingEnergyGradient'] = np.gradient(data['CumulativeBrakingEnergy'])

# Initialize segment identification

data['RecoverySegment'] = 0

segment_id = 0

for i in range(1, len(data)):

    if data.loc[i, 'BrakingEnergyGradient'] > 0:

        if data.loc[i-1, 'BrakingEnergyGradient'] <= 0:

            segment_id += 1

        data.loc[i, 'RecoverySegment'] = segment_id

```

```

else:

    data.loc[:, 'RecoverySegment'] = 0

# Sum energy by segment

recovery_segments = data[data['RecoverySegment'] > 0]

total_energy_by_segment = recovery_segments.groupby('RecoverySegment')['BrakingEnergy'].sum()

# Identify top segments

top_recovery_segments = total_energy_by_segment.nlargest(3)

print(top_recovery_segments)

# Assuming 'data' and 'top_instant_power' are already defined

plt.figure(figsize=(14, 7))

# Cumulative Braking Energy Graph

plt.plot(data['Time'], data['CumulativeBrakingEnergy'], label='Cumulative Braking Energy', color='blue', alpha=0.5)

# Highlighting Top Instantaneous Power Recovery Points

for index, row in top_instant_power.iterrows():

    plt.scatter(row['Time'], data.at[index, 'CumulativeBrakingEnergy'], color='red', zorder=5)

    plt.text(row['Time'], data.at[index, 'CumulativeBrakingEnergy'],

             f'{row["InstantaneousPower_kW"]:.2f} kW', color='red', verticalalignment='bottom')

plt.xlabel('Time (s)')

plt.ylabel('Cumulative Braking Energy (Joules)')

plt.title('Cumulative Braking Energy & Instantaneous Power Recovery Points')

plt.legend()

plt.grid(True)

plt.show()

# Assuming 'data' is your DataFrame with 'Time' and 'CumulativeBrakingEnergy' among the columns,

# and 'RecoverySegment' indicating the recovery segments

plt.figure(figsize=(14, 7))

# Plot the cumulative braking energy over time

plt.plot(data['Time'], data['CumulativeBrakingEnergy'], label='Cumulative Braking Energy', color='blue', alpha=0.5)

# Highlight the top recovery segments

for segment_id in top_recovery_segments.index:

    segment_data = data[data['RecoverySegment'] == segment_id]

    plt.fill_between(segment_data['Time'], segment_data['CumulativeBrakingEnergy'], color='green', alpha=0.3)

# Additional plot for matting

plt.xlabel('Time (s)')

plt.ylabel('Cumulative Braking Energy (Joules)')

plt.title('Cumulative Braking Energy with Highlighted Top Recovery Segments')

plt.legend()

plt.grid(True)

plt.show()

# Assuming the initial part of your code has been executed successfully and 'data' DataFrame is ready

# Calculate Usable Braking Energy considering motor-generator and battery charge efficiency

data['UsableBrakingEnergy'] = data['BrakingEnergy'] * motor_generator_efficiency * battery_charge_efficiency

# Calculate cumulative usable braking energy over time

```

```

data['CumulativeUsableBrakingEnergy'] = data['UsableBrakingEnergy'].cumsum()

# Initialize a new column for the adjusted power requirement

data['AdjustedPower'] = data['TotalPower'].copy()

# Accumulated energy used for adjustment, to keep track of how much energy has been used

accumulated_energy_used = 0

for i in range(len(data)):

    if data.loc[i, 'TotalPower'] > 0: # Only adjust positive power values (acceleration phases)

        # Calculate how much of the accumulated energy can be used to offset this power requirement

        energy_available_to_use = min(data.loc[i, 'CumulativeUsableBrakingEnergy'] - accumulated_energy_used, data.loc[i, 'TotalPower'])

        # Adjust the power requirement by the available energy

        data.loc[i, 'AdjustedPower'] -= energy_available_to_use

        # Update the accumulated energy used

        accumulated_energy_used += energy_available_to_use

# Plotting Original vs Adjusted Power Requirement Over Time

plt.figure(figsize=(20, 8))

# Original Total Power Requirement

plt.plot(data['Time'], data['TotalPower'], label='Original Power Requirement', color='blue', alpha=0.7)

# Adjusted Power Requirement After Applying Recovered Energy

plt.plot(data['Time'], data['AdjustedPower'], label='Adjusted Power Requirement with Energy Recovery', color='green', alpha=0.7)

plt.xlabel('Time (s)')

plt.ylabel('Power (Watts)')

plt.title('Power Requirement Over Time: Original vs Adjusted with Energy Recovery')

plt.legend()

plt.grid(True)

plt.show()

# Calculate the difference between the original and adjusted power for each time step to find savings per second

data['EnergySavings'] = data['TotalPower'] - data['AdjustedPower']

# Calculate total energy savings over the entire period

total_energy_savings = data['EnergySavings'].sum() # This gives the total energy saved in Joules (since power is in Watts and time step is 1 second)

# Use a rolling window of 100 seconds to calculate the sum of energy savings over each 100-second period

data['RollingEnergySavings100s'] = data['EnergySavings'].rolling(window=100).sum()

# Identify the start of the 100-second period with the greatest energy savings

max_savings_period_start = data['RollingEnergySavings100s'].idxmax()

# Greatest energy savings over a 100-second period

greatest_savings_100s = data['RollingEnergySavings100s'].max()

print(f"Total energy savings over the entire period: {total_energy_savings} Joules")

print(f"Greatest energy savings over a 100-second period: {greatest_savings_100s} Joules, starting at second {max_savings_period_start}")

```