A PYTHON TOOL FOR SUMO TRAFFIC SIMULATION TO MODEL HYBRID ELECTRIC VEHICLES (HEV) FOR COMPREHENSIVE EMISSION ANALYSIS

Authors:
Tamás Tettamanti, Chinedu Amabel Okolie, Tamás Ormándi and Balázs Varga

Technical report for the open-source code of https://github.com/bmetrafficlab/SUMO_start_stop_system

BME Traffic Lab
Department of Control for Transportation and Vehicle Systems
Faculty of Transportation Engineering and Vehicle Engineering
Budapest University of Technology and Economics

MŰEGYETEM 1782

Budapest, 2024

Contents

1 Purpose of the Program

Hybrid Electric Vehicles (HEVs) play a crucial role in sustainable transportation, enhancing fuel efficiency and lowering emissions relative to traditional combustion engine vehicles. As the transportation sector greatly impacts global energy use and emissions, advancing HEV technology and its integration into traffic systems is key. This project seeks to develop an adaptable HEV engine/fuel consumption model based on the 2010 Toyota Prius, using variables like speed and acceleration to facilitate real-time applications and aid in environmental policy and traffic management decisions [Ahn and Rakha, 2019]. The goal is to support the shift towards greener mobility by accurately modeling and analyzing HEV performance. This initiative also addresses the complexity of existing HEV models by providing a simpler, yet robust approach that can be adapted for various HEV types, promoting wider application and facilitating advancements in HEV technology for a sustainable future.

2 Preliminaries

This section introduces the foundational elements utilized in HEV modeling, including an overview of Hybrid Electric Vehicles (HEVs), with a specific focus on the 2010 Toyota Prius. It also details the use of SUMO and TraCI for traffic simulations incorporating HEV dynamics. SUMO provides the simulation environment while TraCI facilitates real-time interactions, allowing for detailed analysis of HEV performance under various traffic conditions. Together, these tools form a comprehensive framework for examining the impact of HEV technology on urban mobility and environmental sustainability.

2.1 Overview of Hybrid Electric Vehicles (HEV)

HEVs are at the forefront of automotive innovation, combining the traditional internal combustion engine with electric propulsion systems to create vehicles that are both fuel-efficient and environmentally friendly. One of the most iconic examples of this technology is the Toyota Prius, which has led the market in hybrid technology since its initial launch in 1997 [Qawasmeh et al., 2017]. The Prius was one of the first vehicles to successfully integrate hybrid technology on a large scale, offering consumers the benefits of advanced fuel economy and reduced emissions without compromising performance.

The 2010 Toyota Prius, in particular, highlights significant advancements over its predecessors, most notably in its Hybrid Synergy Drive system. This system integrates the car's combustion engine and electric motors, which work together to optimize fuel efficiency. The 2010 Prius features improved battery technology and energy management systems that enhance the vehicle's ability to store and use electric power, thereby reducing reliance on the gasoline engine and increasing overall efficiency [Burress et al., 2011].

In practical terms, the 2010 Prius has proven its capabilities under various driving conditions, especially in urban settings where stop-and-go traffic typically consumes a significant amount of fuel. [Orecchini et al., 2020] focused on real-world driving conditions have shown that the Prius can maintain high levels of energy efficiency, effectively minimizing fuel consumption and emissions in environments that are traditionally challenging for non-hybrid vehicles.

Performance analysis of the Prius has also been a subject of academic research, particularly regarding its powertrain and transmission systems. The power split device within the Hybrid Synergy Drive is a critical component that allows the vehicle to seamlessly switch between or combine power from the electric motor and the gasoline engine. This not only ensures smooth driving performance but also maximizes energy efficiency across different driving scenarios [Cheng et al., 2011].

Comparative studies involving other hybrid and plug-in hybrid vehicles have demonstrated that the Prius's energy management strategies are integral to its success. These strategies utilize advanced algorithms to decide the most efficient way to use electric power, which is key to enhancing the vehicle's performance and range. The strategic use of electric power helps to reduce the frequency and duration of gasoline engine use, further improving the vehicle's environmental footprint. [Ma et al., 2012]

2.2 Simulation of Urban Mobility(SUMO) and Traffic Control Interface (TraCI)

Simulation of Urban Mobility (SUMO) is an open-source, highly portable, microscopic, and continuous road traffic simulation package designed to handle large networks. It allows for intermodal simulation, including pedestrians, and comes with a suite of tools for scenario creation. Traffic Control Interface (TraCI), on the other hand, is used to interact with the simulation, enabling real-time control and alteration of the simulation for dynamic experiments.

In the context of Hybrid Electric Vehicle(HEV) modeling, SUMO and TraCI play crucial roles:

1. Network Modeling: SUMO allows for the creation of detailed urban mobility scenarios tailored to study HEVs. The network file (net.xml), as used in your code, defines the roads, lanes, junctions, and other network attributes.

2. Simulation Control: TraCI provides a programmatic way to manipulate and retrieve data from the simulation during runtime. This includes changing vehicle routes, retrieving vehicle states, or modifying simulation parameters dynamically, which is essential for testing the impacts of HEVs under various traffic conditions and operational strategies.

3. Mode Switching Logic: The Python script utilizes TraCI to implement mode-switching logic in HEVs, switching between electric and petrol modes based on speed and power thresholds defined in the code. This aspect is critical in studying how HEVs perform in real-world driving conditions and how they contribute to energy efficiency and emissions reductions.

4. Data Collection: Through TraCI, data on fuel consumption, emissions, speed, and other vehicle metrics are collected and analyzed. This information is pivotal for evaluating the environmental impact of HEVs and optimizing their performance.

By integrating SUMO and TraCI into HEV research, the model leverages these tools to develop a deeper understanding of HEV behavior in traffic simulations, contributing to more sustainable transportation solutions. This approach not only helps in refining HEV models but also aids in forecasting the implications of broader HEV deployment in urban areas.

2.3 HBEFA4 Emission Model

The HBEFA (Handbook Emission Factors for Road Transport) version 4 is a comprehensive database that provides precise emission factors for road transport in Europe. It encapsulates detailed research and field studies to estimate emissions from a variety of vehicle types across different road conditions and driving patterns. The (HBEFA4) model is instrumental in environmental planning and policy-making, enabling simulations that reflect real-world vehicle emissions.

For the project at hand, the HBEFA 4 model serves as the backbone for simulating the emission outputs of the hybrid electric vehicles (HEVs) under study. Within the simulation environment, two distinct vehicle types are defined:

1. Petrol_mode: Represents vehicles running on petrol with the emission class <HBEFA4/PC_PHEV_petrol_Euro-4_(P)>. This class is aligned with Euro 4 standards for passenger cars with plug-in hybrid electric vehicle characteristics when operating in petrol mode.

2. Electric_mode: Signifies the electric operational mode of the HEVs, utilizing the emission class <HBEFA4/PC_PHEV_petrol_Euro-4_(El)>. While this category reflects the same Euro 4 standards, it is specifically tailored for the electric driving phase of plug-in hybrids, hence emitting significantly lower pollutants.

Incorporating these emission classes into the simulation allows for a realistic portrayal of the environmental impact of HEVs as they switch between petrol and electric modes. The HBEFA 4 model, thus, not only enhances the accuracy of the simulation but also provides a robust framework for analyzing the potential benefits of HEVs in urban traffic.

3 Overview of the Program

In this part, the main features of the developed program are introduced.

3.1 Starting the Program

This program is written in python and initiates with the importation of several Python libraries, each serving a critical function in the setup and execution of the SUMO simulation. The *'traci'* library, or Traffic Control Interface, enables real-time interaction and control over the SUMO simulation directly from Python scripts, making it indispensable for dynamic traffic management simulations. Mathematical calculations, essential for physics-based elements of the simulation, are handled by the *'math'* library. The simulation's network configurations, defined in XML format, are parsed using *'xml.etree.ElementTree'*, commonly referenced as *ET*. For handling large data sets and complex mathematical operations, the *'numpy'* library is employed, while *'matplotlib.pyplot'* is used to visually represent simulation results through various plots and charts, enhancing the interpretability of data such as emissions and vehicle metrics.

```
import traci
import math
import xml.etree.ElementTree as ET
import numpy as np
import matplotlib.pyplot as plt
```

The configuration for the simulation is specified in two key files. The *sumo_cfg*, a configuration file located at:

```
sumo_cfg = "C:/Users/ochin/Downloads/ttt/m1m7_2023Oct_Mondays_8.sumocfg"
```

which contains critical settings such as the network file paths, vehicle routes, and detailed simulation parameters like time steps and duration. Additionally, the physical layout of the traffic network is detailed in the netFile_path, which points to:

```
netFile_path = "C:/Users/ochin/Downloads/ttt/M1-M7.net.xml"
```

and includes descriptions of roads, lanes, and junctions that are crucial for accurate traffic simulations.

To launch the simulation, the *traci.start* function is employed, which accepts a series of command-line arguments dictating the operation mode of SUMO. The command:

```
traci.start(["sumo-gui", "-c", sumo_cfg, "--start"])
```

is typically used to initiate the *SUMO* simulation with its graphical user interface. This mode not only facilitates visual monitoring of the simulation process but also allows for interactive adjustments, enhancing the user's ability to conduct detailed traffic analyses. The -c flag is used to specify the configuration file, ensuring that all predefined settings are correctly loaded to set the stage for the simulation. The –start argument ensures that the simulation begins automatically, providing a seamless start-up experience.

This setup initiates the SUMO environment, readying it for the simulation scenarios defined by the user. The interface provided by *traci* enables dynamic interaction with the running simulation, such as retrieving vehicle data or modifying vehicle states based on the simulation conditions.

3.2 Customization Overview

This section underscores the program's capacity to morph according to user specifications, ensuring that each simulation is fine-tuned to address distinct research questions or operational scenarios. The user-centric approach underscores the simulation's versatility, making it an indispensable tool for traffic management and environmental impact assessment of hybrid electric vehicles.

As the backbone of user-driven customization, the simulation script initiates with the users setting their own simulation environment by providing paths to the SUMO configuration file (*sumo_cfg*) and the network file (*netFile_path*) as seen in section 3.1. These paths are crucial as they direct the simulation to the user-defined setup files, detailing the network layout and the simulation's dynamics. By allowing users to input their file paths, the program extends the capability to model a vast array of traffic scenarios and network configurations, reflecting the diverse conditions under which hybrid electric vehicles operate.

Once the environment is set, users are afforded the autonomy to dictate the vehicle dynamics and simulation parameters which are pivotal in shaping the behavior of the vehicles within the simulation. Constants employed in the script, such as vehicle mass, rolling resistance coefficient, air density, drag coefficient, and frontal area, are carefully chosen to emulate the real-world dynamics of a 2010 Toyota Prius. This decision to model after a specific vehicle type is grounded in empirical data, ensuring that the simulated vehicle's interactions with the environment are representative of actual performance.

```
vehicle_mass = 1460  # in kg (avg human weight + weight of prius)
gravitational_constant = 9.81  # in m/s^2
rolling_resistance_coefficient = 0.2  # for dry asphalt
air_density = 1.293  # in kg/m^3
drag_coefficient = 0.23  # for a 2010 Toyota Prius
frontal_area = 2.22  # in m^2 (converted from 23.9 square feet)
regeneration_efficiency = 0.462  # 46.2% regen efficiency
```

The simulation further accentuates precision by incorporating efficiency values, which serve as a testament to the simulation's adherence to real-world accuracy. Variables such as *eta_batt*, *eta_pe*, *eta_mot*, and *eta_pt* reflect the battery, power electronics, electric motor, and powertrain efficiencies respectively.

```
eta_batt = 0.95   # Average battery efficiency (90% to 95%)
eta_pe = 0.98     # Average power electronics efficiency (96% to 99%)
eta_mot = 0.96    # Average electric motor efficiency (90% to 97%)
eta_pt = 0.98     # Mechanical efficiency of the powertrain
```

These efficiencies are not arbitrary; they are founded on rigorous research, enabling the simulation to deliver realistic insights into the fuel consumption patterns of hybrid electric vehicles [Varga et al., 2019].

Customization extends into the operational logic of the simulation. Users have the discretion to establish thresholds for the vehicle mode switching logic through:

```
SPEED_THRESHOLD_FOR_ELECTRIC = 32  # km/h
SPEED_THRESHOLD_FOR_PETROL = 100   # km/h
POWER_THRESHOLD_FOR_ELECTRIC = 10  # kW
POWER_THRESHOLD_FOR_PETROL = 120   # kW
```

These thresholds are pivotal in determining when a vehicle should operate on electric power versus when to switch to petrol, hence influencing the simulation's outcome on energy consumption and emissions.

A notable aspect of customization is the flexibility to specify the proportion of vehicles equipped with mode-switching capabilities. By adjusting the parameter in the variable *vehicle_ratio*, users can experiment with different ratios of HEVs within the traffic mix, offering a versatile tool for studying the impacts of HEV penetration in traffic flows.

```
# Ratio of vehicles with mode switching
vehicle_ratio = 20
```

The program's adaptability does not end with the setting of physical and efficiency constants. It further extends to the customization of vehicle types and route definitions in the additional configuration file. This adaptability is exemplified by the introduction of custom vehicle types within the simulation, allowing for detailed modeling of different vehicular technologies and their distinct behavior patterns. Specifically, the script leverages emission classes such as "HBEFA4/PC_PHEV_petrol_Euro-4_(P)" for petrol mode and "HBEFA4/PC_PHEV_petrol_Euro-4_(El)" for electric mode. These classes facilitate the simulation's ability to switch between electric and petrol modes, accurately capturing the emissions profile for each vehicle state as defined in the following XML:

```
<additional>
    <vType id="petrol_mode" carFollowModel="EIDM" laneChangeModel="LC2013" minGap="
   2.0" emissionClass="HBEFA4/PC_PHEV_petrol_Euro-4_(P)"/>
    <vType id="electric_mode" carFollowModel="EIDM" laneChangeModel="LC2013" minGap="
   2.0" emissionClass="HBEFA4/PC_PHEV_petrol_Euro-4_(El)"/>
</additional>
```

By incorporating these specific emission classes into vehicle definitions, the simulation not only reflects the nuanced performance characteristics of hybrid electric vehicles but also enhances the accuracy of environmental impact assessments. This level of detail ensures that each vehicle's emissions are tracked and analyzed based on its current mode of operation, offering a sophisticated approach to studying the real-world implications of HEV technology adoption.

3.3 Parameter Validation

Parameter validation is key to ensuring the appropriate setup before the simulation runs commence. The parameters set by the user, which include the path to the SUMO configuration file (*sumo_cfg*), the network file (*netFile_path*), and the ratio of vehicles with mode-switching capability (*vehicle_ratio*), are essential inputs that drive the simulation process.

The *vehicle_ratio* parameter is especially crucial as it determines the percentage of the traffic fleet capable of mode switching, which should be a positive integer, ideally between 1 and 100, to indicate a percentage. The thresholds for mode switching as mentioned in section 3.2 are set to align with the specific dynamics of the modeled vehicles and traffic conditions. These values influence the mode-switching behavior of the vehicles during the simulation, impacting metrics such as fuel consumption and emissions.

At the heart of the simulation logic is the run_simulation function. It initiates the SUMO simulation with the given user settings and starts the process of assigning vehicles and collecting data on their performance. Within the simulation's setup phase, the run_simulation function calls upon parse_net_file, passing it the net_file_path parameter.

```
lane_shapes = parse_net_file(netFile_path)
```

This function meticulously parses the network XML file specified by net_file_path and constructs a dictionary that maps each lane's ID to its geometric shape.

```
def parse_net_file(net_file_path):

    tree = ET.parse(net_file_path)
    root = tree.getroot()

    lane_shapes = {}

    for lane in root.findall('.//lane'):
        lane_id = lane.get('id')
        shape = lane.get('shape')

        # Convert the shape string into a list of (x,y) tuples
        points = [tuple(map(float, point.split(','))) for point in shape.split()]

        # Store the start and end points
        if points:
            lane_shapes[lane_id] = {'start': points[0], 'end': points[-1]}

    return lane_shapes
```

This mapping is foundational for subsequent calculations of road slope which, when coupled with the speed and acceleration of each vehicle, are integral to the determination of instantaneous power requirements. These calculations are pivotal as they underpin the dynamic mode-switching logic, affecting both the energy consumption and the emissions output during the simulation.

Once the simulation steps begin, the slope for each vehicle's current lane is calculated.

```
if lane_id in lane_shapes:
    z_start = lane_shapes[lane_id]['start'][2]
    z_end = lane_shapes[lane_id]['end'][2]

    change_in_z = z_end - z_start

    lane_length = traci.lane.getLength(lane_id)

    alpha_radians = math.atan(change_in_z / lane_length)

    theta = alpha_radians
```

This is crucial for determining the road's gradient, which, alongside speed and acceleration, feeds into the calculation of instantaneous power. The calculation begins with the slope for each vehicle's current lane, which is derived from the geometric shapes obtained through the *parse_net_file* function using the *net_file_path*. The slope is a direct measure of the road's gradient and is essential for accurate physics-based modeling of vehicle behavior.

With the slope determined, the simulation then integrates the vehicle's current speed and its acceleration to calculate the instantaneous power.

```
total_instantaneous_power = calculate_instantaneous_power(speed_in_km, acceleration,
    theta)
instantaneous_power_in_kw = total_instantaneous_power / 1000  # Converts power to kW
```

This power calculation is a complex formula that encapsulates several forces acting on the vehicle:

1. *Rolling Resistance Power*: It considers the resistance encountered by the tires rolling on the road surface, which is dependent on the coefficient of rolling resistance, vehicle mass, gravitational constant, and the vehicle's speed.

2. *Gradient Resistance Power*: This component accounts for the power needed to overcome the gravitational pull when ascending or descending a slope.

3. *Aerodynamic Drag Power*: The resistance from air drag is factored in, especially at higher speeds, and is influenced by the air density, vehicle's frontal area, drag coefficient, and the cube of the vehicle's speed.

4. *Acceleration Power*: When a vehicle accelerates, additional power is required, which is calculated using the vehicle's mass and the rate of acceleration.

5. *Regenerative Braking Power*: In the case of deceleration, the simulation accounts for the potential regeneration of power, which is a unique aspect of electric and hybrid vehicles. This regeneration is modeled to recapture energy during braking, adding to the efficiency of the vehicle mode.

These forces are combined to ascertain the total tractive power required at any given moment or the power required to move the vehicle.

$$P_w(k) = P_{roll}(k) + P_g(k) + P_{drag}(k) + P_{acc}(k) + P_{regen}(k) \tag{1}$$

To obtain the instantaneous power, this total tractive power is divided by the product of the efficiencies of the vehicle's battery (*eta_batt*), power electronics (*eta_pe*), electric motor (*eta_mot*), and the powertrain mechanical efficiency (*eta_pt*). These efficiencies are critical as they represent the energy losses during the conversion of electrical energy to mechanical energy and vice versa. For instance, battery efficiency accounts for the energy lost when storing and retrieving energy from the battery, while motor efficiency accounts for the conversion of electrical energy into mechanical work [Varga et al., 2019].

$$E_{cons}(k) = \sum_{k=1}^{N} \frac{P_{roll}(k) + P_g(k) + P_{drag}(k) + P_{acc}(k)}{\eta_{batt} \cdot \eta_{pe} \cdot \eta_{mot} \cdot \eta_{pt}} + P_{regen}(k) \cdot \eta_{batt} \cdot \eta_{pe} \cdot \eta_{mot} \cdot \eta_{pt} \tag{2}$$

In scenarios where the vehicle is decelerating and potentially regenerating energy, the simulation incorporates the regeneration efficiency. This reflects the vehicle's capability to recover a portion of the kinetic energy typically lost during braking and convert it back to electrical energy, which is then stored in the battery. The recovered power is a product of the negative acceleration (deceleration) and the regeneration efficiency, which is then added to the instantaneous power after being appropriately adjusted by the aforementioned efficiencies.

```
def calculate_instantaneous_power(speed, acceleration, theta):

    # Rolling Resistance Power
    rolling_resistance_power = rolling_resistance_coefficient * vehicle_mass *
   gravitational_constant * math.cos(theta) * speed

    # Gradient Resistance Power
    gradient_resistance_power = vehicle_mass * gravitational_constant * math.sin(
   theta) * speed

    # Aerodynamic Drag Power
    aerodynamic_drag_power = 0.5 * air_density * drag_coefficient * frontal_area *
   speed ** 3
```

```
    # Acceleration Power
    if acceleration > 0:
        positive_acceleration = acceleration
    else:
        positive_acceleration = 0
    acceleration_power = vehicle_mass * positive_acceleration * speed

    # Regeneration Power
    if acceleration < 0:
        negative_acceleration = acceleration
        regeneration_power = vehicle_mass * negative_acceleration * speed *
    regeneration_efficiency
    else:
        regeneration_power = 0

    # Summing all the power components.
    total_tractive_power = rolling_resistance_power + gradient_resistance_power +
    aerodynamic_drag_power + acceleration_power

    total_instantaneous_power = (total_tractive_power / (eta_batt * eta_pe * eta_mot
    * eta_pt)) + (regeneration_power * eta_batt * eta_pe * eta_mot * eta_pt)

    return total_instantaneous_power
```

The calculation of instantaneous power not only considers the immediate demands placed on the vehicle due to road conditions and driving actions but also the intrinsic efficiencies of the vehicle's components. The simulation leverages this detailed calculation to ascertain whether the vehicle should be operating in electric mode, favoring efficiency and reduced emissions, or switch to petrol mode to meet higher power demands, reflecting a key decision-making process in the operation of hybrid electric vehicles.

The mode-switching logic is a dynamic process within the run_simulation function. It constantly evaluates whether a vehicle should be in electric or petrol mode based on its current speed and power demand. If a vehicle exceeds the *SPEED_THRESHOLD_FOR_ELECTRIC* but is below the *SPEED_THRESHOLD_FOR_PETROL*, and its power demand goes over the *POWER_THRESHOLD_FOR_ELECTRIC*, it switches to electric mode. Conversely, if it exceeds both the speed and power thresholds for petrol, it reverts to petrol mode.

```
# Mode switching logic
if vehicle_id in mode_switch_vehicles:
    # Check if we need to switch to electric mode
    if type_id == "petrol_mode" and SPEED_THRESHOLD_FOR_ELECTRIC < speed_in_km <= \
            SPEED_THRESHOLD_FOR_PETROL and instantaneous_power_in_kw >
    POWER_THRESHOLD_FOR_ELECTRIC:
        traci.vehicle.setType(vehicle_id, "electric_mode")
        vehicle_modes[vehicle_id] = "electric_mode"
        print(f"Vehicle {vehicle_id} at {speed_in_km}km/h has switched to electric
    mode.")

    # Check if we need to switch back to petrol mode
    elif type_id == "electric_mode" and speed_in_km > SPEED_THRESHOLD_FOR_PETROL and
    \
            instantaneous_power_in_kw > POWER_THRESHOLD_FOR_PETROL:
        traci.vehicle.setType(vehicle_id, "petrol_mode")
        vehicle_modes[vehicle_id] = "petrol_mode"
        print(f"Vehicle {vehicle_id} at {speed_in_km}km/h has switched back to petrol
     mode.")
```

This logic is repeated at each simulation step, allowing for a responsive and dynamic simulation that can model real-world driving conditions and vehicle responses. The mode-switching mechanism plays a crucial role in assessing the efficacy and environmental impact of hybrid vehicles in urban settings.

3.3.1 Mode-Switch logic Equations

In order to express the mode-switching logic s a formal equation, we can summarize the conditions under which a vehicle switches between modes using mathematical notation. This can be implemented as:

- v represent the vehicle speed in km/h.
- p represent the instantaneous power in kW.
- S_e represent the speed threshold for switching to electric mode.
- S_p represent the speed threshold for switching back to petrol mode.
- P_e represent the power threshold for switching to electric mode.
- P_p represent the power threshold for switching back to petrol mode.

Equation for Switching to Electric Mode

- If a vehicle is currently in petrol mode:

$$\text{Switch to Electric if } (S_e < v \leq S_p) \wedge (p > P_e) \tag{3}$$

Equation for Switching to Petrol Mode

- If a vehicle is currently in electric mode:

$$\text{Switch to Petrol if } (v > S_p) \wedge (p > P_p) \tag{4}$$

Description:

1. **Electric Mode Switching Condition:**
   - The vehicle must be traveling at a speed greater than the electric mode speed threshold (S_e) and less than or equal to the petrol mode speed threshold (S_p).
   - Additionally, the vehicle's power output must exceed the electric mode power threshold (P_e).
2. **Petrol Mode Switching Condition:**
   - The vehicle must be traveling at a speed greater than the petrol mode speed threshold (S_p).
   - The vehicle's power output must also exceed the petrol mode power threshold (P_p).

These equations provide a clear, formal description of when a vehicle should switch between electric and petrol modes based on its speed and power output, aligning with the logic implemented in the python script through TraCI for controlling HEVs in a simulated environment.

3.4 Emission Data Handling

The handling of emission data within the traffic simulation is a critical step that directly influences the accuracy and reliability of the study's environmental impact assessment. The simulation effectively distinguishes between two distinct types of vehicles: those equipped with mode-switching capabilities and regular vehicles. This bifurcation is essential, as it allows the assessment of how switching between electric and petrol modes can influence overall emissions.

The *run_simulation* function encapsulates the core logic of emission data collection. Within this function, vehicles are dynamically assigned to either the mode-switch group or the regular group, depending on whether their ratio falls within the user-defined threshold for mode-switching capabilities. This classification leverages the *vehicle_ratio* parameter, determining the proportion of vehicles that will have mode-switching capabilities during the simulation.

As the simulation progresses, each vehicle's emissions are meticulously recorded at every step. For mode-switch vehicles, the emissions data are captured along with their respective modes, reflecting the immediate environmental output based on whether they are operating in electric or petrol mode. This data is stored in two primary dictionaries, *mode_switch_metrics* and *regular_vehicle_metrics*, which maintain a record of the accumulated emissions(CO_2, NO_x, PM_x) and fuel consumption for each group of vehicles throughout the simulation run.

```
mode_switch_metrics = {
    'fuel_consumption': [],
    'pmx': [],
    'co2': [],
    'nox': [],
    'speed': []
}

regular_vehicle_metrics = {
    'fuel_consumption': [],
    'pmx': [],
    'co2': [],
    'nox': [],
    'speed': []
}
```

The emission data are then processed to extract meaningful insights. For instance, the simulation analyzes the cumulative emissions, revealing the potential reduction in pollutants due to the use of electric mode in mode-switch vehicles. The simulation also records the speed data, adding another layer to the emission profiles, as vehicle speeds can significantly affect emission rates.

the emission data handling within this simulation is carefully orchestrated to provide a detailed temporal view of emissions for further analysis. The structured approach to capturing and processing data emphasizes the sophisticated methods employed to simulate and analyze the impact of mode-switching on vehicle emissions. The simulation's timeline is meticulously managed, ensuring that data collection and adjustments are precisely timed and recorded, underscoring the program's refined approach to simulating and assessing the ecological footprint of HEVs.

3.5 Cumulative Emissions and Output

The post-simulation phase is dedicated to the critical task of compiling and outputting cumulative emissions data. This phase synthesizes the wealth of information captured during the simulation to provide a comprehensive view of the environmental impact of vehicles with mode-switching capabilities compared to regular vehicles.

The simulation aggregates the collected emission data, summing up the total amounts of CO2, NOx, PMx, and fuel consumption for both mode-switch and regular vehicles separately. These sums are vital in quantifying the overall environmental footprint of the simulated traffic scenario.

```
print("------------------------------")
print("Cumulative emissions for vehicles without mode-switch logic:")
print("Sum of CO2:", sum_CO2, "mg")
print("Sum of PMx:", sum_PMx, "mg")
print("Sum of NOx:", sum_NOx, "mg")
print("Sum of Fuel Consumption:", sum_fuel, "mg")

print("------------------------------")
print("Cumulative emissions for vehicles with mode-switch logic:")
print("Sum of CO2:", sum_CO2_mode_switch, "mg")
print("Sum of PMx:", sum_PMx_mode_switch, "mg")
print("Sum of NOx:", sum_NOx_mode_switch, "mg")
print("Sum of Fuel Consumption:", sum_fuel_mode_switch, "mg")
```

Using the mode_switch_metrics and regular_vehicle_metrics dictionaries, the script calculates the cumulative emissions for each pollutant type. This is achieved by summing the individual emission metrics over the entire simulation period for both sets of vehicles.The total emissions are then printed to the console, providing immediate feedback on the environmental performance of the vehicles. This output includes the sum of each pollutant type for mode-switch vehicles and regular vehicles.

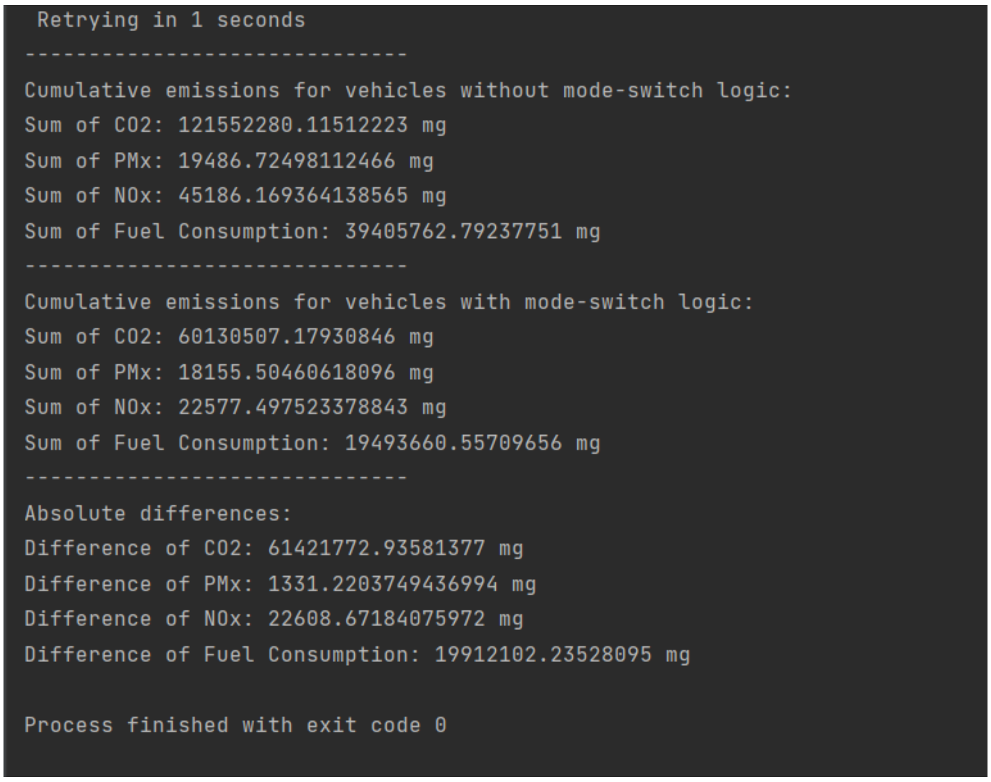
To underscore the impact of mode-switching capabilities, the script calculates the absolute differences in emissions between the mode-switch vehicles and the regular vehicles. This highlights the potential reductions in emissions achieved through mode-switching technology.

```
print("------------------------------")
print("Absolute differences:")
print("Difference of CO2:", diff_CO2, "mg")
print("Difference of PMx:", diff_PMx, "mg")
print("Difference of NOx:", diff_NOx, "mg")
print("Difference of Fuel Consumption:", diff_fuel, "mg")
```

The script visualizes the data through the *plot_grouped_bar_chart* function, which generates a bar chart to illustrate the absolute differences in emissions. This visual representation is essential for a clear and immediate understanding of the data. The cumulative emissions data, along with the visualization, serve as a robust basis for documentation and further analysis. Researchers and practitioners can utilize this information to evaluate the efficacy of mode-switching technology in reducing vehicular emissions.

3.6 Program Ending

As the simulation concludes, the program's closing sequence initiates a final review to ascertain the successful execution of all procedures and the integrity of the data collected. This moment marks a significant milestone, affirming the precision and stability of the simulation. While the code does not explicitly confirm the end of the process, it is implicit in the successful generation of output data and visualizations. A simple print statement could be introduced to signal the program's completion, reassuring the user of the simulation's thoroughness and the reliability of the findings it has yielded.



```
 Retrying in 1 seconds
------------------------------
Cumulative emissions for vehicles without mode-switch logic:
Sum of CO2: 121552280.11512223 mg
Sum of PMx: 19486.72498112466 mg
Sum of NOx: 45186.169364138565 mg
Sum of Fuel Consumption: 39405762.79237751 mg
------------------------------
Cumulative emissions for vehicles with mode-switch logic:
Sum of CO2: 60130507.17930846 mg
Sum of PMx: 18155.50460618096 mg
Sum of NOx: 22577.497523378843 mg
Sum of Fuel Consumption: 19493660.55709656 mg
------------------------------
Absolute differences:
Difference of CO2: 61421772.93581377 mg
Difference of PMx: 1331.2203749436994 mg
Difference of NOx: 22608.67184075972 mg
Difference of Fuel Consumption: 19912102.23528095 mg

Process finished with exit code 0
```

Figure 1: Output example

4 A Working Example

In this part, a working example is presented with example outputs generated by the tool.

4.1 Assessing the Emission Effect of a Ratio of Vehicles with HEV mode-switch logic

In our exploration of the environmental benefits of mode-switch technology in vehicular traffic, we configure a SUMO simulation environment with a 50% distribution of vehicles equipped with the ability to switch between electric and internal combustion engine modes. This setup reflects a balanced approach, providing a nuanced view of the technology's potential impact.

The user-defined settings within the script establish the parameters of the simulation, with *sumo_cfg* pointing to the SUMO configuration file and *netFile_path* indicating the network definition file. These initial settings are critical as they lay the groundwork for the simulation, dictating the traffic flow and network over which the vehicles will operate.

With constants carefully chosen to represent a 2010 PHEV Toyota Prius, the simulation incorporates realistic vehicle dynamics. These constants—vehicle mass, gravitational constant, rolling resistance coefficient, air density, drag coefficient, frontal area, regeneration efficiency, and various efficiencies (battery, power electronics, motor, and powertrain)—are all taken into account to accurately model the vehicle's behavior and energy consumption.

The simulation proceeds to parse the network file, obtaining crucial data about lane shapes, which feeds into the calculation of road gradients. These gradients, along with vehicle speed and acceleration, enable the computation of instantaneous power—a determining factor in the mode-switch logic.

As the simulation begins, vehicles are dynamically assigned to either mode-switch or regular categories. The *run_simulation* function, set with a *vehicle_ratio* of 50, ensures that half of the vehicles in the simulation can switch modes according to speed and power thresholds, which are also user-defined.

Throughout the simulation run, detailed metrics on fuel consumption, CO_2, NO_x, and PM_x emissions, and speed are collected for both vehicle types. These metrics are crucial for assessing the environmental performance of the mode-switch technology.

The results, as seen in the provided plots and output(Fig.1), display the cumulative emissions for both sets of vehicles. The first plot(Fig.2) illustrates the total emissions and fuel consumption side by side, clearly highlighting the reductions achieved by mode-switch vehicles. The second plot presents the absolute differences in emissions between the two sets, offering a stark visualization of the benefits offered by the mode-switch mechanism.

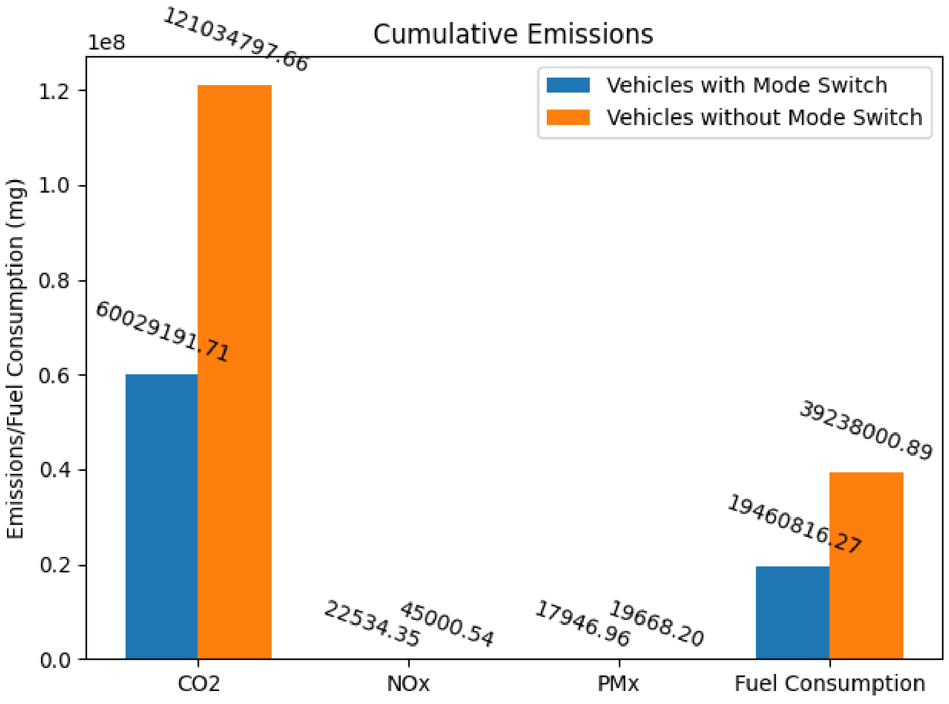


Figure 2: Plot output example showcasing the cumulative emissions and fuel consumption measured

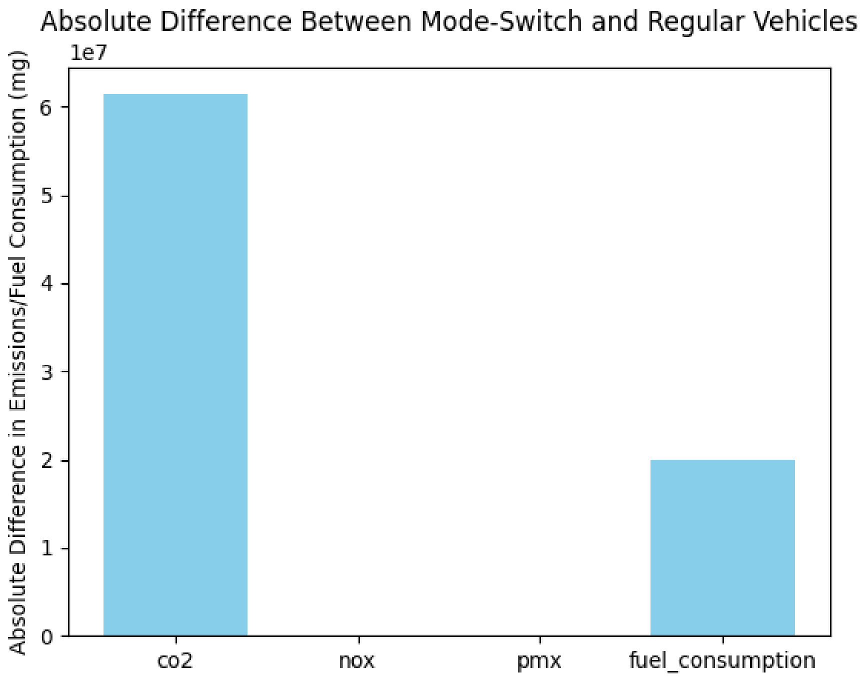


Figure 3: Plot output example showcasing the absolute difference of total emissions measured

Concluding with the console output, we can observe the significant reductions in emissions, particularly CO_2, achieved through the mode-switch system. The process concludes without error, indicating a successful run and affirming the reliability of the data collected.

This working example, supported by concrete data and visualizations, underscores the effectiveness of mode-switch vehicles in reducing emissions and serves as a persuasive argument for the broader application of such technologies in urban mobility planning.

5 Limitations

5.1 Model Specificity

The project's results are heavily dependent on the characteristics of a 2010 PHEV Toyota Prius, including its weight, aerodynamics, and powertrain efficiency. This specificity means the simulation's applicability to other models or newer technologies is limited. For instance, newer hybrid models may have different efficiencies, battery capacities, and regenerative braking capabilities that would affect real-world performance. Additionally, the simulation does not account for advancements in internal combustion engines or fully electric vehicles, which could offer different environmental benefits or drawbacks.

5.2 Simulation Environment

While SUMO is adept at replicating traffic flow and general vehicle dynamics, it does not fully replicate the idiosyncrasies of real-world driving. Real-world factors such as sudden stops, evasive maneuvers, variations in driver behavior (like aggressive acceleration or idling), and the impact of vehicle maintenance (like tire pressure and engine health) are challenging to model accurately in any simulation environment. These factors can significantly influence fuel consumption and emissions, potentially leading to discrepancies between the simulated and real-world environmental impact of start-stop systems.

5.3 Emission Factors

The project uses theoretical models to estimate emissions, which might not align with the actual emissions produced by vehicles under varied driving conditions. Emissions can be affected by a range of factors not accounted for in the simulation, such as changing load (passenger and cargo weight), road grade, ambient temperature, and the state of the vehicle's catalytic converter. Additionally, the simulation does not consider cold starts, which can substantially increase a vehicle's emissions until the catalytic converter reaches its operating temperature.

5.4 Energy Regeneration

The simulation assumes a constant regeneration efficiency for the vehicle's energy recovery system, which is an oversimplification. In reality, the efficiency of regenerative braking varies with factors like battery charge level, temperature, and age. For example, a nearly full or very cold battery accepts charge less efficiently, reducing the benefits of regenerative braking. Moreover, different driving scenarios (stop-and-go traffic vs. highway driving) can greatly impact the amount of energy that can be recuperated.

5.5 Traffic Complexity

The simulation environment may not capture the intricacies of traffic flow, such as the impact of traffic lights, stop signs, and the interactions between vehicles at different times of the day. Traffic congestion can lead to increased idling times and frequent start-stop cycles, which could affect the efficiency of mode-switching systems. While SUMO can simulate traffic jams and other congestion patterns, these do not fully encompass the complexity of urban traffic systems, nor do they reflect the diverse driving environments that exist in different cities or countries.

References

Kyoungho Ahn and Hesham Rakha. A simple hybrid electric vehicle fuel consumption model for transportation applications(doi: 10.5772/intechopen.89055). Technical report, U.S. Department of Energy, Office of Scientific and Technical Information, 10 2019. URL `https://www.osti.gov/biblio/1570836`.

Timothy A Burress, Steven L Campbell, Chester Coomer, Curtis William Ayers, Andrew A Wereszczak, Joseph Philip Cunningham, Laura D Marlino, Larry Eugene Seiber, and Hua-Tay Lin. Evaluation of the 2010 toyota prius hybrid synergy drive system. Technical report, U.S. Department of Energy, Office of Scientific and Technical Information, 3 2011. URL `https://www.osti.gov/biblio/1007833`.

Yuan Cheng, Rochdi Trigui, Christophe Espanet, Alain Bouscayrol, and Shumei Cui. Specifications and design of a pm electric variable transmission for toyota prius ii. *IEEE Transactions on Vehicular Technology*, 60(9): 4106–4114, 2011. doi: 10.1109/TVT.2011.2155106.

C Ma, J Kang, W Choi, M Song, J Ji, and H Kim. A comparative study on the power characteristics and control strategies for plug-in hybrid electric vehicles. *International Journal of Automotive Technology*, 13: 505–516, 2012.

Fabio Orecchini, Adriano Santiangeli, and Fabrizio Zuccari. Hybrid-electric system truth test: Energy analysis of toyota prius iv in real urban drive conditions. *Sustainable Energy Technologies and Assessments*, 37:100573, 2020. ISSN 2213-1388. doi: https://doi.org/10.1016/j.seta.2019.100573. URL `https://www.sciencedirect.com/science/article/pii/S2213138818306593`.

BR Qawasmeh, A Al-Salaymeh, A Swaity, A Mosleh, and S Boshmaf. Investigation of performance characteristics of hybrid cars. *Environmental Engineering*, 14:59–69, 2017.

Balázs Varga, Tamás Tettamanti, and Balázs Kulcsár. Energy-aware predictive control for electrified bus networks. *Applied Energy*, 252:113477, 2019. ISSN 0306-2619. doi: https://doi.org/10.1016/j.apenergy.2019.113477. URL `https://www.sciencedirect.com/science/article/pii/S0306261919311511`.

6 Appendix: the full code of the HEV-plugin Program (main.py)

```python
import traci
import math
import xml.etree.ElementTree as ET
import numpy as np
import matplotlib.pyplot as plt


# <--------------------- USER SETTINGS -------------------->
sumo_cfg = "C:/Users/ochin/Downloads/ttt/m1m7_2023Oct_Mondays_8.sumocfg"
netFile_path = "C:/Users/ochin/Downloads/ttt/M1-M7.net.xml"

# Ratio of vehicles with mode switching
vehicle_ratio = 50

# Define thresholds for mode switching
SPEED_THRESHOLD_FOR_ELECTRIC = 32  # km/h
SPEED_THRESHOLD_FOR_PETROL = 100   # km/h
POWER_THRESHOLD_FOR_ELECTRIC = 10  # kW
POWER_THRESHOLD_FOR_PETROL = 120   # kW
# <--------------------- END OF USER SETTINGS -------------------->


# <---------CONSTANTS FOR 2010 PHEV TOYOTA PRIUS----------------->
vehicle_mass = 1460  # in kg (avg human weight + weight of prius)
gravitational_constant = 9.81  # in m/s^2
rolling_resistance_coefficient = 0.2  # for dry asphalt
air_density = 1.293  # in kg/m^3
drag_coefficient = 0.23  # for a 2010 Toyota Prius
frontal_area = 2.22  # in m^2 (converted from 23.9 square feet)
regeneration_efficiency = 0.462  # 46.2% regen efficiency


eta_batt = 0.95  # Average battery efficiency (90% to 95%)
eta_pe = 0.98    # Average power electronics efficiency (96% to 99%)
eta_mot = 0.96   # Average electric motor efficiency (90% to 97%)
eta_pt = 0.98    # Mechanical efficiency of the powertrain


def parse_net_file(net_file_path):
    """
    Parse the network file to extract lane shapes.

    :param net_file_path: Path to the network XML file.
    :return: Dictionary mapping lane IDs to their start and end points.
    """
    tree = ET.parse(net_file_path)
    root = tree.getroot()

    lane_shapes = {}

    for lane in root.findall('.//lane'):
        lane_id = lane.get('id')
        shape = lane.get('shape')

        # Convert the shape string into a list of (x,y) tuples
        points = [tuple(map(float, point.split(','))) for point in shape.split()]

        # Store the start and end points
        if points:
            lane_shapes[lane_id] = {'start': points[0], 'end': points[-1]}

    return lane_shapes


def calculate_instantaneous_power(speed, acceleration, theta):

    """
    Calculate the instantaneous power needed or regenerated by a vehicle at a given
   speed, acceleration,
    and road elevation.

    :param speed: Speed of the vehicle in km/h.
    :param acceleration: Acceleration of the vehicle in m/s^2.
    :param theta: Road elevation in radians.
    :return: Total instantaneous power in watts.
    """

    # Rolling Resistance Power
    rolling_resistance_power = rolling_resistance_coefficient * vehicle_mass *
   gravitational_constant * math.cos(theta) * speed

    # Gradient Resistance Power
    gradient_resistance_power = vehicle_mass * gravitational_constant * math.sin(
   theta) * speed

    # Aerodynamic Drag Power
    aerodynamic_drag_power = 0.5 * air_density * drag_coefficient * frontal_area *
   speed ** 3

    # Acceleration Power
    if acceleration > 0:
        positive_acceleration = acceleration
    else:
        positive_acceleration = 0
    acceleration_power = vehicle_mass * positive_acceleration * speed

    # Regeneration Power
    if acceleration < 0:
        negative_acceleration = acceleration
        regeneration_power = vehicle_mass * negative_acceleration * speed *
   regeneration_efficiency
    else:
        regeneration_power = 0

    # Summing all the power components.
    total_tractive_power = rolling_resistance_power + gradient_resistance_power +
   aerodynamic_drag_power + acceleration_power

    total_instantaneous_power = (total_tractive_power / (eta_batt * eta_pe * eta_mot
   * eta_pt)) + (regeneration_power * eta_batt * eta_pe * eta_mot * eta_pt)

    return total_instantaneous_power


def run_simulation(mode_switch_ratio):
    """
    Runs the traffic simulation with mode-switching capabilities and collect metrics.

    The simulation assigns vehicles with mode-switch capabilities based on a
```

```
specified ratio and collects various
 performance metrics, such as fuel consumption, PMx emissions, electricity
consumption, CO2 emissions, and speed.

 Mode-switch logic determines whether a vehicle should operate in electric or
petrol mode based on its speed and
 power demand. If a vehicle's speed is within the electric mode threshold and the
power demand exceeds the threshold
 for electric power, the vehicle switches to electric mode. Conversely, if the
vehicle's speed exceeds the petrol
 mode threshold and the power demand is above the petrol power threshold, it
switches back to petrol mode.

 :param mode_switch_ratio: Percentage of vehicles that will have mode-switching
capabilities.
 :return: A tuple of two dictionaries containing mode-switch metrics and regular
vehicle metrics.
 """
 lane_shapes = parse_net_file(netFile_path)
 traci.start(["sumo-gui", "-c", sumo_cfg, "--start"])

 period_time = 300
 step_time = 0.5

 assigned_vehicles = set()
 mode_switch_vehicles = set()
 total_vehicles_processed = 0

 # Dictionary to track the current mode of the vehicles
 vehicle_modes = {}

 mode_switch_metrics = {
     'fuel_consumption': [],
     'pmx': [],
     'co2': [],
     'nox': [],
     'speed': []
 }

 regular_vehicle_metrics = {
     'fuel_consumption': [],
     'pmx': [],
     'co2': [],
     'nox': [],
     'speed': []
 }

 for i in range(int(period_time / step_time)):
     traci.simulationStep()
     current_vehicle_ids = set(traci.vehicle.getIDList())

     for vehicle_id in current_vehicle_ids:
         type_id = traci.vehicle.getTypeID(vehicle_id)

         if vehicle_id not in assigned_vehicles:
             total_vehicles_processed += 1
             # Assigns vehicles to mode switch group based on the ratio
             if (len(mode_switch_vehicles) / total_vehicles_processed) < (
mode_switch_ratio / 100):
                 mode_switch_vehicles.add(vehicle_id)
             assigned_vehicles.add(vehicle_id)
```

```
        speed_in_km = math.ceil(traci.vehicle.getSpeed(vehicle_id) * 3.6)  # 
Convert m/s to km/h
        acceleration = traci.vehicle.getAcceleration(vehicle_id)

        lane_id = traci.vehicle.getLaneID(vehicle_id)

        # Calculate Slope
        # Check if the lane ID is in lane_shapes dictionary
        if lane_id in lane_shapes:
            z_start = lane_shapes[lane_id]['start'][2]
            z_end = lane_shapes[lane_id]['end'][2]

            change_in_z = z_end - z_start

            lane_length = traci.lane.getLength(lane_id)

            alpha_radians = math.atan(change_in_z / lane_length)

            theta = alpha_radians

            total_instantaneous_power = calculate_instantaneous_power(speed_in_km
, acceleration, theta)

            instantaneous_power_in_kw = total_instantaneous_power / 1000  # 
Converts power to kW

            # Mode switching logic
            if vehicle_id in mode_switch_vehicles:
                # Check if we need to switch to electric mode
                if type_id == "petrol_mode" and SPEED_THRESHOLD_FOR_ELECTRIC < 
speed_in_km <= \
                        SPEED_THRESHOLD_FOR_PETROL and instantaneous_power_in_kw
 > POWER_THRESHOLD_FOR_ELECTRIC:
                    traci.vehicle.setType(vehicle_id, "electric_mode")
                    vehicle_modes[vehicle_id] = "electric_mode"
                    # print(f"Vehicle {vehicle_id} at {speed_in_km}km/h has 
switched to electric mode.")

                # Check if we need to switch back to petrol mode
                elif type_id == "electric_mode" and speed_in_km > 
SPEED_THRESHOLD_FOR_PETROL and \
                        instantaneous_power_in_kw > POWER_THRESHOLD_FOR_PETROL:
                    traci.vehicle.setType(vehicle_id, "petrol_mode")
                    vehicle_modes[vehicle_id] = "petrol_mode"
                    # print(f"Vehicle {vehicle_id} at {speed_in_km}km/h has 
switched back to petrol mode.")

                mode_switch_metrics['fuel_consumption'].append(traci.vehicle.
getFuelConsumption(vehicle_id))
                mode_switch_metrics['pmx'].append(traci.vehicle.getPMxEmission(
vehicle_id))
                mode_switch_metrics['nox'].append(traci.vehicle.getNOxEmission(
vehicle_id))
                mode_switch_metrics['co2'].append(traci.vehicle.getCO2Emission(
vehicle_id))
                mode_switch_metrics['speed'].append(speed_in_km)

            else:
                # No mode switching; just append metrics to 
regular_vehicle_metrics
```

```
                    regular_vehicle_metrics['fuel_consumption'].append(traci.vehicle.
    getFuelConsumption(vehicle_id))
                    regular_vehicle_metrics['pmx'].append(traci.vehicle.
    getPMxEmission(vehicle_id))
                    regular_vehicle_metrics['nox'].append(traci.vehicle.
    getNOxEmission(vehicle_id))
                    regular_vehicle_metrics['co2'].append(traci.vehicle.
    getCO2Emission(vehicle_id))
                    regular_vehicle_metrics['speed'].append(speed_in_km)

    traci.close()

    return mode_switch_metrics, regular_vehicle_metrics


def plot_grouped_bar_chart(mode_switch_metrics, regular_vehicle_metrics):
    """
    Plot grouped bar charts comparing emissions and fuel consumption for vehicles
    with and
    without mode-switch capabilities.

    :param mode_switch_metrics: Dictionary containing aggregated metrics for mode-
    switch vehicles.
    :param regular_vehicle_metrics: Dictionary containing aggregated metrics for
    regular vehicles.
    """
    # Sum of emissions without mode-switch logic
    sum_CO2 = sum(regular_vehicle_metrics['co2'])
    sum_PMx = sum(regular_vehicle_metrics['pmx'])
    sum_NOx = sum(regular_vehicle_metrics['nox'])
    sum_fuel = sum(regular_vehicle_metrics['fuel_consumption'])

    # Print cumulative sum
    print("------------------------------")
    print("Cumulative emissions for vehicles without mode-switch logic:")
    print("Sum of CO2:", sum_CO2, "mg")
    print("Sum of PMx:", sum_PMx, "mg")
    print("Sum of NOx:", sum_NOx, "mg")
    print("Sum of Fuel Consumption:", sum_fuel, "mg")

    # Sum of emissions with mode-switch logic
    sum_CO2_mode_switch = sum(mode_switch_metrics['co2'])
    sum_PMx_mode_switch = sum(mode_switch_metrics['pmx'])
    sum_NOx_mode_switch = sum(mode_switch_metrics['nox'])
    sum_fuel_mode_switch = sum(mode_switch_metrics['fuel_consumption'])

    # Print cumulative sum
    print("------------------------------")
    print("Cumulative emissions for vehicles with mode-switch logic:")
    print("Sum of CO2:", sum_CO2_mode_switch, "mg")
    print("Sum of PMx:", sum_PMx_mode_switch, "mg")
    print("Sum of NOx:", sum_NOx_mode_switch, "mg")
    print("Sum of Fuel Consumption:", sum_fuel_mode_switch, "mg")

    # Calculating the absolute differences
    diff_CO2 = abs(sum_CO2_mode_switch - sum_CO2)
    diff_PMx = abs(sum_PMx_mode_switch - sum_PMx)
    diff_NOx = abs(sum_NOx_mode_switch - sum_NOx)
    diff_fuel = abs(sum_fuel_mode_switch - sum_fuel)

    # Print Absolute differences
    print("------------------------------")
    print("Absolute differences:")
    print("Difference of CO2:", diff_CO2, "mg")
    print("Difference of PMx:", diff_PMx, "mg")
    print("Difference of NOx:", diff_NOx, "mg")
    print("Difference of Fuel Consumption:", diff_fuel, "mg")

    metrics = ['co2', 'nox', 'pmx', 'fuel_consumption']
    mode_switch_sums = [sum(mode_switch_metrics[metric]) for metric in metrics]
    regular_sums = [sum(regular_vehicle_metrics[metric]) for metric in metrics]

    # Define the label locations and the width of the bars
    x = np.arange(len(metrics))  # the label locations
    width = 0.35  # the width of the bars

    fig, ax = plt.subplots()
    rects1 = ax.bar(x - width/2, mode_switch_sums, width, label='Vehicles with Mode
   Switch')
    rects2 = ax.bar(x + width/2, regular_sums, width, label='Vehicles without Mode
   Switch')

    ax.set_ylabel('Emissions/Fuel Consumption (mg)')
    ax.set_title('Cumulative Emissions')
    ax.set_xticks(x)
    ax.set_xticklabels(['CO2', 'NOx', 'PMx', 'Fuel Consumption'])
    ax.legend()

    # Plotting the absolute difference bar chart
    fig_diff, ax_diff = plt.subplots()
    diffs = [diff_CO2, diff_NOx, diff_PMx, diff_fuel]
    ax_diff.bar(metrics, diffs, color='skyblue')
    ax_diff.set_ylabel('Absolute Difference in Emissions/Fuel Consumption (mg)')
    ax_diff.set_title('Absolute Difference Between Mode-Switch and Regular Vehicles')
    ax_diff.set_xticks(x)
    ax_diff.set_xticklabels(metrics)

    # Text label
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.annotate('{:.2f}'.format(height),
                        xy=(rect.get_x() + rect.get_width() / 2, height),
                        xytext=(0, 3),  # 3 points vertical offset
                        textcoords="offset points",
                        ha='center', va='bottom',
                        rotation=-20)

    autolabel(rects1)
    autolabel(rects2)

    fig.tight_layout()

    plt.show()


mode_switch_metrics, regular_vehicle_metrics = run_simulation(vehicle_ratio)
plot_grouped_bar_chart(mode_switch_metrics, regular_vehicle_metrics)
```