Data Extraction from Electric Vehicles through OBD and Application of Carbon Footprint Evaluation

Chien-Ming Tseng[†], Wenshen Zhou[‡], Mohammad Al Hashmi[†],
Chi-Kin Chau[†], Soh Gim Song[‡] and Erik Wilhelm[‡]

[†]Masdar Institute of Science and Technology, UAE
[‡]Singapore University of Technology and Design, Singapore
{ctseng,malhashmi,ckchau}@masdar.ac.ae,{wenshen_zhou,sohgimsong,erikwilhelm}@sutd.edu.sg

ABSTRACT

While the data extraction through the On-Board Diagnostics (OBD) port for internal combustion engine vehicles follows an industry standard, electric vehicles are not constrained by such a standard. In this paper, we demonstrate how to extract data from the OBD port of electric vehicles. We discuss the general techniques, and apply specifically to Nissan Leaf and Chevrolet Volt. We continue to present an application of carbon footprint evaluation by analyzing the data extracted from the OBD port to obtain parameters of interest. Furthermore, the methodology that can be used in various ways to evaluate the carbon footprint of electric vehicles by building synthetic driving cycles is presented.

Keywords

Electric Vehicles, Data Extraction, OBD, Carbon Footprint

1. INTRODUCTION

Over one million electric vehicles (EVs) and Plug-in Hybrid Electric Vehicles (PHEVs) have been sold worldwide, and the study of the impact vehicles with electrified power trains are making is of increasing interest to researchers. To drivers, an accurate understanding of the vehicle status (e.g tire pressure, oil temperature and braking etc...) is a necessary indicator for safe and comfortable driving. In addition to these basic signals, EV drivers are often provided with additional information relating to battery level, battery temperature, distance-to-empty, and expected charging time. For researchers requiring insight into the details of the power train operation, a wealth of information is available from the On-Board Diagnostics (OBD) port [13], a mandatory communication interface for vehicle inspection and maintenance. OBD systems can provide status of the various vehicle subsystems for the vehicle owner or repair technician, but are also a rich source of data for researchers as we will proceed to describe in this paper.

The OBD protocol in modern vehicles is based on the Controller Area Network (CAN bus) architecture [3], a vehicle communication bus standard designed to allow controllers, sensors, and actuators to communicate with each other without a host computer. For example, the engine parameters like engine speed, mass-air flow and engine load are exchanged in CAN and used by the engine ignition timing system. Such information is available on CAN but of course is not shown on the vehicle dashboard. On an EV's CAN bus, the battery and motor information can be obtained via the OBD port, as well as many other valuable parameters for research applications. Unlike the mature combustion vehicle market, however, which has standard OBD protocol, getting data from EVs can be more difficult since car manufacturers have not yet reached agreement on the standard messages to be transmitted. For example, the message protocol for getting battery information may be very different from one manufacture to another. Against this backdrop, the contributions which we present in this paper are:

1. An encyclopedic reference for accessing useful information on the CAN bus via OBD port of electric vehicles.
2. A method for the synthesis of artificial driving cycles from CAN data that approximate real-world driving along principle dimensions for the study of carbon footprint.

This paper is organized as follows. In Section 2 we discuss previous work in this area, and in Section 3 we describe the CAN bus via OBD port in general terms. The best practice in extracting meaningful information from vehicles is described in Section 4, with particular emphasis on electrical vehicles, and the use of an OBD interpreter for the same purpose is described in detail in Section 5. An application of these data extraction technique to populated models used to generate synthetic drive cycles which help characterize energy and emissions from EV's are described in Section 6. We conclude in Section 7.

2. RELATED WORK

OBD II system has been a mandatory equipment for monitoring emission performance and vehicle maintenance, the standard procedures have been defined to obtain the OBD data [9] for combustion vehicles. There is a body of proprietary about vehicle diagnostic applications of smartphones [16], drivers can use the applications with OBD dongles to check the status of their vehicles. A hands-on project that uses a CAN bus equipment to obtain the battery and motor information for a Nissan Leaf is presented in [1]. A detail look at extracting CAN bus data for the Toyota Prius and Ford Escape has been presented in [15], where the authors

EV-Sys, June 21-24 2016, Waterloo, ON, Canada
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4420-3/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2939953.2939954

demonstrate analysis and injections of CAN packets to control vehicle via a laptop and CAN bus tools.

Several interesting applications have been built by research teams based on OBD data. Using OBD data from combustion vehicles to build a eco-routing system has been shown [7], as has using OBD data to create personalized energy consumption prediction models for different types of vehicles [2, 11, 12].

3. OVERVIEW OF OBD AND CAN BUS

In this section, we provide an overview of CAN bus, and introduce the systems which may be used to obtain data for electrified vehicle power trains.

3.1 Introduction to Vehicle CAN Bus

Controller Area Network (CAN) protocol was introduced in the early 1980s by Robert Bosch GmbH. It was originally developed for automotive systems and applications, but has found widespread use in other industrial and automation domains. Modern automobiles typically have more than 50 electronic control units (ECU) serving various roles in various subsystems. CAN is a serial bus that allows fast (up to 1MB/s) and robust (built-in error detection, retransmission, node isolation etc.) exchange of data between the different electronic modules in vehicles. Although the original OBD standard was protocol agnostic, CAN has become commonly applied to serve the data required by OBD in most vehicles.

At the data link layer (layer 2), CAN packets are composed of an identifier and data payload. The identifier usually comprises 11 bits or 29 bits, though 11 bit identifiers are most common. The identifier (CAN ID) also denotes the priority of the message, the lower the value, the higher the priority. Each controller on the bus broadcasts messages to all others, and the CAN 'arbitration ID' concept is therefore important for relevant control units on the bus to decide whether the packet should be processed and rebroadcast or not. After the identifier, there are from 0 to 8 bytes of data. Sometimes the data contains check-sum or other mechanisms to insure data integrity. Readers interested in more details concerning the CAN protocol's bit timing, synchronization, and data frame definitions, are encouraged to refer to [4, 6].

3.2 Overview of CAN Bus Equipment

There are two popular methods to extract the data from a vehicle's CAN bus which we will discuss. The first is to use a CAN bus 'sniffing tool' to observe the CAN message traffic at Layer 2 on the vehicles' CAN bus. This approach requires some background knowledge of CAN protocols. The second, and easier way is to use an 'OBD dongle', which interprets the Layer 2 message traffic and parses them to return human-readable parameters based on OBD conversion rules.

1. **Data link layer monitoring tools**

   CAN bus 'sniffing tools' can read the CAN packet and inject the CAN packet to the network. A commonly-used simple variant is a CAN-BUS shield designed to interface with an Arduino board. Such shields typically shield uses Microchip MCP2515 CAN controller with an MCP2551 CAN transceiver, and require users to program their own interpretation firmware. A more versatile and simple to use CAN sniffing tool, the neoVI Fire, allows users to record and extract CAN packets easily via commercial software. Data link layer monitoring tools typically serve data to portable computers for visualization and analysis due to the volume of serial data which must be read from the bus.

2. **OBD interpreters**

   A more direct and user-friendly approach to obtaining CAN data is to use a chip set which is designed in an application specific way to interpret CAN traffic into parameters which are then re-broadcast as serial messages. The most common OBD dongles use ELM327 hardware, but more recent products also employ STN1110 chip sets [10]. Both products are OBD to RS323 interpreters which function as a bridge that connect the OBD port to a standard RS232 serial port. The ELM327 hardware has the ability to work with nine different CAN protocols in addition to automatically detecting the used protocol. ELM327 is fully configurable using `AT` commands which is defined by the manufacturer [5]. It operates in two modes which are high speed mode and sleep mode. This chip is widely used in diagnostic trouble code readers and automotive scan tools. The dongles are usually equipped with Bluetooth or Wifi wireless function, which allows users to remotely connect to the dongles. Popular OBD interpreters are able to utilize Bluetooth or TCP/IP interfaces to smart phones or computers to communicate because of the relatively parsimonious way of extracting data.

We summarize the complete framework of utilizing the vehicle CAN bus in Fig.1 which shows the two common pathways.

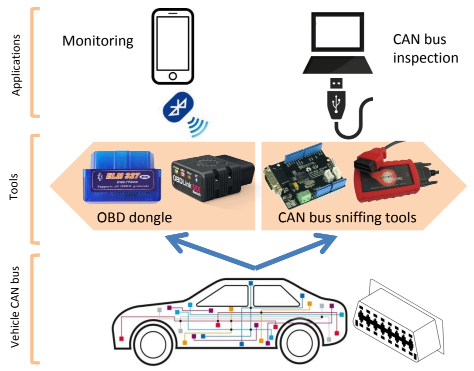


Figure 1: Illustration of using vehicle CAN bus.

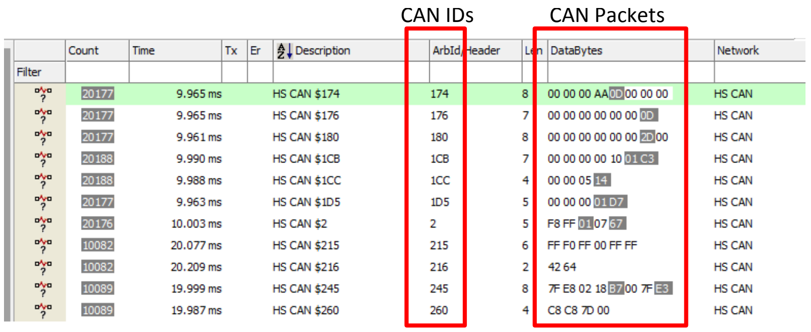
4. EXTRACTING INFORMATION FROM THE DATA LINK LAYER

In this section, we demonstrate the use of CAN bus sniffing tool, neoVI Fire, to get the data from the CAN network of Nissan Leaf and reverse-engineer the message of CAN packet.

4.1 Equipment and Setup

Fig.2a illustrates the practical setup of the demonstration. The neoVI Fire is connected to the laptop and its supported software, Vehicle Spy, is used. Fig.2b displays the interface of Vehicle Spy, we label two import information of the interface, the first one is CAN IDs and its related CAN packets.

(a) Practical setup and OBD port position in Leaf.



	Count	Time	Tx	Er	Description	ArbId/Header	Len	DataBytes	Network
Filter									
	20177	9.965 ms			HS CAN $174	174	8	00 00 00 AA 0D 00 00 00	HS CAN
	20177	9.965 ms			HS CAN $176	176	7	00 00 00 00 00 00 0D	HS CAN
	20177	9.961 ms			HS CAN $180	180	8	00 00 00 00 00 00 2D 00	HS CAN
	20188	9.990 ms			HS CAN $1CB	1CB	7	00 00 00 00 10 01 C3	HS CAN
	20188	9.988 ms			HS CAN $1CC	1CC	4	00 00 05 14	HS CAN
	20177	9.963 ms			HS CAN $1D5	1D5	5	00 00 00 01 D7	HS CAN
	20176	10.003 ms			HS CAN $2	2	5	F8 FF 01 07 67	HS CAN
	10082	20.077 ms			HS CAN $215	215	6	FF F0 FF 00 FF FF	HS CAN
	10082	20.209 ms			HS CAN $216	216	2	42 64	HS CAN
	10089	19.999 ms			HS CAN $245	245	8	7F E8 02 18 B7 00 7F E3	HS CAN
	10089	19.987 ms			HS CAN $260	260	4	C8 C8 7D 00	HS CAN

(b) Vehicle Spy user interface.

Figure 2: Equipment and software interface.

4.2 Reverse Engineering CAN Packet of Leaf

Although we can read CAN packets from the data link layer of the vehicle network, the meanings of the packet are not published by the vehicle manufacturers. To extract the meaningful information and manipulate the CAN bus to control the vehicles, we have to reverse-engineer the CAN packets. There are two popular methods for determining specific CAN components:

1. **Fuzzing**

   We can iteratively test random or partially random CAN packets and see how the vehicle reacts. This can be performed without much effort due to limited number of CAN IDs in the network. We strongly suggest not to use this dangerous approach, since the vehicle may act irregularly and risk safety. In addition, the vehicles' warranty will be voided.

2. **Visually correlate physical system interactions with identifiable patterns**

   Most of sensors and actuators are connected to the vehicle CAN bus and broadcast to each other, hence, we can observe the CAN packets change when we physically interact with the vehicle. For example, seat belt light is turned off when the belt is fastened, which means there is a CAN packet sent to the buzzer controller to stop the warning sounds. In this paper, we use this method to determine the CAN ID and CAN packet of gear position sent to dashboard in Nissan Leaf.

4.2.1 *Demonstration*

To perform the second approach to CAN reverse engineering, we stop the Leaf and change the gear position from Parking to Neutral repeatedly. At the same time we observe the changes of CAN IDs and their packets in the Vehicle Spy. Luckily, two CAN ID actions correlate our gear changing actions, we list these two CAN data in Tab.1. Recording the

CAN ID	Length	Parking	Neutral
216	2	42 64	42 60
421	3	08 00 00	18 00 00

Table 1: Part of CAN packets of Leaf

received CAN packets, we can use the software to inject the mock CAN packets (e.g., packet of neutral position) in the Leaf CAN bus. We first try to send the packet to CAN ID 216 but the dashboard of Leaf does not changes, and we receive I-Key System Fault in the dashboard when the packet is sent for a while. Therefore, we can consider that CAN ID 216 possibly relates to the key detection as well as changes of gear position. Next, we try to send the packet to CAN ID 421, and successfully change gear position shown in dashboard. Therefore, we now know the CAN ID 421 controls the display of gear position on the dashboard. Fig.3 shows the result of this reverse-engineer demonstration. We would like to re-emphasize that experiments of this nature should be performed under controlled conditions, and we recommend lifting vehicle wheels off of the ground for any injection of message data to the CAN bus.

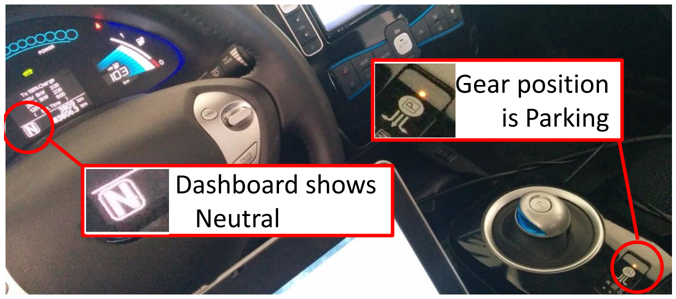


Figure 3: Changing the dashboard display of gear position.

5. EXTRACTING DATA FROM OBD INTERPRETERS

In this section, we demonstrate using ELM327 Bluetooth OBD dongle and smartphone to query the information from two vehicles, Nissan Leaf and Chevrolet Volt.

An application exists for smart phone platforms called 'Leaf Spy Pro', provides measurement of Leaf information using this kind of dongle, however, its sample rate is relatively low (0.25 Hz).

5.1 Notations

The format of received information is very different from one vehicle to another, therefore, parsing the information to meaningful values is necessary. To explain the parsing rule for received information, we define several notations for string extraction and conversion:

1. Hex substring (S_i^j): We define S_i^j as the substring of index i to j from the received hex string. Where i begins from 0 and starts at the leftmost of the string.
2. Unsigned integer conversion (uInt): We use $\mathsf{uInt}(S)$ to convert the hex string to unsigned integer value.

Command	Response	Parsing rule	Unit	Description and Result
022101	7BB10296101FFFFFBA4	—	—	—
	7BB2102870000271BFF	$\mathsf{Int}(S_9^{16})/1000$	Amp.	Battery current: 10A
	7BB22FFFFFF03C62AF8	—	—	—
	7BB2399C432A7038400	$\mathsf{uInt}(S_5^8)/100$	Vol.	Battery voltage: 393.64V
	7BB245C2098000B2ECC	$\mathsf{uInt}(S_{11}^{18})/10000$	SOC %	SOC: 73.29%
	7BB2500088D9F800005	$\mathsf{uInt}(S_5^{12})/10000$	Ah	Capacity: 56.05 Ah

Table 2: Commands and responses of Leaf

Command	Response	Parsing rule	Unit	Description and Result
222883	4183008D	$\mathsf{Int}(S_4^7)/20$	Amp.	Motor A current: 7.05A
222884	41840665	$\mathsf{Int}(S_4^7)/20$	Amp.	Motor B current: 81.85A
222885	418586C4	$\mathsf{uInt}(S_4^7)/100$	Vol.	Motor A voltage: 345V
222886	418686C5	$\mathsf{uInt}(S_4^7)/100$	Vol.	Motor B voltage: 346V
222414	4141072C	$\mathsf{Int}(S_4^7)/20$	Amp.	Battery current: 91.8A
222429	41295101	$\mathsf{Int}(S_4^7)/64$	Vol.	Battery voltage: 324.02V

Table 3: Commands and responses of Volt

3. Signed integer conversion (Int): We use $\mathsf{Int}(S)$ to convert the hex string to signed integer value.

EXAMPLE 1. *Assume we receive the following hex string S comprising 19 characters:*

`7BB10296101FFFFFBA4`

The value of $\mathsf{uInt}(S_{15}^{18}) = \mathsf{uInt}(\texttt{FBA4}) = 64420$.

5.2 Dongle Setup for the Vehicles

We use OBDLink Bluetooth dongle with Android app on a smart phone to obtain the data from the vehicles. Before starting sending the query commands to the dongles, we have to initialize the dongle to match the CAN protocol of the vehicles. This section provides the list of commands of OBD dongle setup for Leaf and Volt. The commands are verified for ELM327 and STN1110 dongles. Tab.4 shows the `AT` setup commands for Nissan Leaf and Chevrolet Volt.

Leaf Setup	Description
`ATZ`	reset all
`ATL0`	line feed off
`ATSP6`	set protocol to mode 6 (ISO-15765-4 CAN)
`ATH1`	set headers on
`ATS0`	set space off
`ATCAF0`	set auto formatting off
`ATSH797`	set header to 797
`ATFCSH797`	set flow control and header to 797
`ATFCSD300014`	set flow control and data to 300014
`ATFCSM1`	set flow control to mode 1
`ATSH79B`	set header to 79B
`ATFCSH79B`	set flow control and header to 79B
Volt Setup	
`ATZ`	reset all
`ATL0`	line feed off
`ATE0`	set echo off
`ATS0`	set space off
`ATSP0`	set auto protocol

Table 4: `AT` setup commands for ELM327 dongles

5.3 Nissan Leaf (Model 2013)

The queried information from Leaf includes battery current, battery voltage, state-of-charge (SOC) and battery capacity. The information is useful for the energy consumption modeling and driving distance estimation. We notice that the current of battery is a signed value. Positive value means the battery is being charged by the plug-in cable or regenerative braking. On the other hand, negative value means the battery is used for powertrain and auxiliary loading.

5.4 Chevrolet Volt (Model 2013)

Chevrolet Volt is an plugin-hybrid electric vehicle (PHEV), which support both standard OBD Parameter IDs (PIDs) for engine information and specific PIDs for battery and electric motor information. Volt comprises two motors, the motor with higher power is used as main propelling motor (Motor B), on the other hand, the smaller one is used as generator or auxiliary propelling (Motor A).

Tab.3 shows the query commands and responses for battery and motor information of Volt.

On the contrary, the negative value of battery current means the battery is being charged and positive value means the battery is discharging.

6. CARBON FOOTPRINT EVALUATION

Although EVs do not produce tailpipe emissions, the electricity grid to charge their battery may produce emissions. Carbon dioxide (CO_2) is the primary greenhouse gas emitted through human activities. In this section, we calculate the CO_2 emission of EVs and compare it to that of combustion vehicle in the same trip, which can be seen as net emission saving of EVs. We first collect vehicle data using the approach described in Section 5, then the data is utilized to create energy consumption models which can be translated into CO_2 emissions using the emission factors in Table 6. Furthermore, to determine the speed profiles for energy consumption models, we use high-level attributes information to generate the closer real-world driving profiles.

6.1 Construct Energy Consumption Model

According to the vehicle powertrain model, the vehicle power can be expressed as following equation:

$$P_t^B = \eta\cdot(\frac{\rho_a k_d A_f v_t^3}{2} + \mathsf{mg}\sin(\alpha_t)v_t + \mathsf{mg}k_r v_t + \mathsf{m}v_t a_t + \mathsf{c}_0) \quad (1)$$

where P_t^B is the power from battery of EVs, m is the vehicle weight, g is the gravitational constant, α_t is the road grade (e.g., inclination of route), ρ_a is the density of air, A_f is the frontal area of the vehicle, k_d if aerodynamic drag coefficient of the vehicle, k_r is the rolling friction coefficient, and c_0 is the default load (e.g., due to air-conditioning). These parameters can be obtained from standard vehicle information or OBD measurement.

Where η is the total efficiency includes battery efficiency, transmission efficiency and motor efficiency, which is nonlinear and difficult to get. For simplicity, we use quadratic regression model to map the efficiency according to vehicle speed.

$$\eta = \alpha_1 v^2 + \alpha_2 v + \alpha_3 \tag{2}$$

where the coefficients $\alpha_{1,2,3}$ can be determined using least squares method.

6.2 Construct Custom Driving Cycle

A driving cycle is a time-series of data points representing the speed of a vehicle. In this section we use high-level attributes (e.g. road types, traffic conditions, and driver's aggressiveness) to build custom driving cycles reflecting real-world driving patterns, which is the required input to the energy consumption model.

6.2.1 Methodology

The method for generating custom driving cycle mainly includes two parts: Multivariate regression model construction and driving cycle generation using dynamic programming. The relation between high-level attributes and driving pattern can be mapped by regression models.

6.2.2 Driving Pattern Regression Model

Considering the large number of factors affecting driving and the complexity of involving all factors in a physical-based model, we use a multivariate regression method to describe the relation between high-level attributes and driving pattern. This model outputs six target parameters to represent driving patterns, which includes: Average velocity v_{ave}, average absolute value of acceleration $|a|_{ave}$, percentage of time in acceleration P_a, percentage of time in deceleration P_d, percentage of time in cruise P_c and percentage of time in idling P_i. We take the parameter v_{ave} as an example:

$$\hat{v}_{ave}(i) = \beta_0 + \beta_1 \chi_{i1} + \beta_2 \chi_{i2} + \cdots + \beta_m \chi_{nm} \tag{3}$$

Where β_i is a set of unknown coefficients determined from the historical data. χ is a set of variables that influence driving pattern, and must be measurable and predictable. Solving $\boldsymbol{\beta}$ in Equ.(3) from the historical data:

$$\boldsymbol{\beta} = (\boldsymbol{\chi}^T \boldsymbol{\chi})^{-1} \boldsymbol{\chi}^T \boldsymbol{v}_{ave} \tag{4}$$

Similarly, the other five target parameters can be determined and used to describe a driving cycle.

Considering the limit of vehicle types and minuscule precipitation in test sites and the constraints that few of high-level attributes can be directly measured, we choose the following three parameters to form $\boldsymbol{\chi}$.

(1) *Traffic conditions*: Traffic conditions influence driving pattern significantly. The percentage of time spent at $v < 10$ km/h will be used to represent traffic conditions:

$$\chi_{TC} = Pt_{v<10} \tag{5}$$

(2) *Driving aggressiveness*: Driving aggressiveness is affected by many factors. Acceleration could reflect how aggressive a period of driving is. Here we use the mean of absolute acceleration to represent driving aggressiveness.

$$\chi_{DA} = |a|_{ave} \tag{6}$$

(3) *Road types*: Four categories of roads, highway, main street, city road and campus road are represented by number 1 to 4, where type 1 denotes the highest speed level (e.g., highway) and type 4 denotes the lowest speed level (e.g., campus road). A trip may consist of different portions of road types, therefore, we define the weighted average for a trip by χ_{RT}.

$$\chi_{RT} = \gamma_1 + 2\gamma_2 + 3\gamma_3 + 4\gamma_4 \tag{7}$$

where γ_i is the portion of road type i in the trip.

Provide high-level attribute tuples $\boldsymbol{\chi} = [\ 1 \quad \chi_{TC} \quad \chi_{DA} \quad \chi_{RT}\ ]$ as well as the collected driving data, the coefficiencts $\boldsymbol{\beta}$ can be determined and thus regression model is completed.

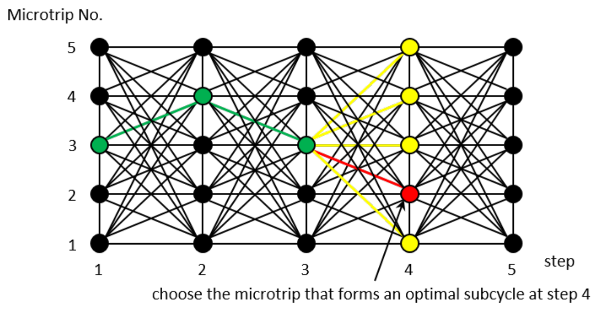
6.2.3 Driving Cycle Generation with DP

Construction of driving cycles is based on microtrips. Microtrip is an excursion between two successive time points at which the vehicle starts to move, consisting of a period of motion starting from zero speed, followed by a period of stopping. Existing standard driving cycles are used to generate microtrips by being divided at every starting point.

A driving cycle is constructed by connecting different microtrips sequentially. The goal of the driving cycle generation procedure is to build a cycle that best satisfies the criteria estimated from regression model. The error E is introduced to describe the difference between generated cycle and criteria outputted from regression model:

$$\begin{aligned} E = {} & \delta_1 (v^d_{ave} - v^c_{ave})^2 + \delta_2 (|a|^d_{ave} - |a|^c_{ave})^2 \\ & + \delta_3 (P^d_a - P^c_a)^2 + \delta_4 (P^d_d - P^c_d)^2 \\ & + \delta_5 (P^d_c - P^c_c)^2 + \delta_6 (P^d_i - P^c_i)^2 \end{aligned} \tag{8}$$

where $\delta_1, \cdots, \delta_6$ are weighted coefficients, $v^d_{ave}, |a|^d_{ave}, P^d_a$, P^d_d, P^d_c, P^d_i are the target parameters of driving cycle, and $v^c_{ave}, |a|^c_{ave}, P^c_a, P^c_d, P^c_c, P^c_i$ are corresponding parameters in the criteria.



Microtrip No.
5
4
3
2
1
1
2
3
4
5
step
choose the microtrip that forms an optimal subcycle at step 4

Figure 4: The network for the driving cycle generation problem, exemplified for the case of five steps.

The optimal driving cycle will be that of minimal error. Here a "subproblem" of step k is defined as finding the optimal cycle from the first step to the kth step, and a "subcycle" is the solution to "subproblem". The whole problem can be solved by solving "subproblem" sequentially: solve $(k+1)$th subproblem with kth subcycle adding a new microtrip from feasible set. The new cycle with a minimal E will be chosen as $(k+1)$th subcycle. Figure 4 illustrates the procedure of finding solution of step 4 subproblem with step 3 solution. Here only microtrips whose speed and acceleration does not exceed the bounds in real driving profiles can be used to form cycles. Therefore, there would be a feasible set S with N_S microtrips for the cycle generation. This leads to the following algorithm to solve the whole problem, where $ERR(A, B)$

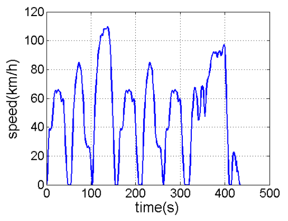
means calculating E between profiles A and B according to Eq.(8).

noend 1 `Cycle.DP`

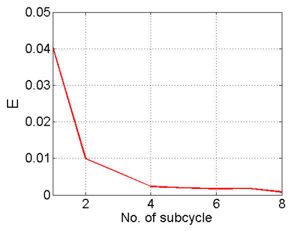
```
1: for each microtrip $M_j \in S$ do
2:     $EM_j \leftarrow ERR(M_j, Criteria)$
3: $E(1) \leftarrow \min_j EM_j, Subsycle \leftarrow [M_j], Index(1) \leftarrow j$
4: for $k$ 2 to $N_S$ do
5:     for $j$ 1 to $N_S$ do
6:         $OptCycle_j \leftarrow [Subcycle, M_j]$
7:     $E(k) \leftarrow \min_j ERR(OptCycle_j, Criteria)$
8:     $Subcycle \leftarrow OptCycle_j, Index(k) \leftarrow j$
9:     if $E(k) < \varepsilon$ then
10:        break
11: return $Cycle(M_{Index(1)}, M_{Index(2)}, \cdots, M_{Index(k)})$
```

6.2.4 *Driving Cycle Generation Results*

Parameters of driving pattern regression model are trained and validated with EV OBD data obtained. Then model outputs criteria corresponding to driving conditions inputted, for dynamic programming algorithm to approach. Errors of all six target parameters between generated cycle and criteria are computed to evaluate this algorithm's performance. Figure 5 shows the result with a specific set of driving conditions of $\boldsymbol{\chi_1} = [\ 1\quad \chi_{TC}\quad \chi_{DA}\quad \chi_{RT}\ ] = [\ 1\quad 0.1\quad 0.8\quad 2\ ]$. Six target parameters of criteria and the generated cycle are shown in Table 5: All errors of six parameters are less than 10%, meaning the generated cycle can be a good representation of driving patterns under this specific driving condition. Tests with other driving conditions show similar results.



(a) Generated driving cycle.



(b) Change of error E with adding new microtrips.

Figure 5: Generated Cycle and Change of E with a Specific Driving Condition χ_1.

Parameters	v_{ave} (km/h)	$\lvert a\rvert_{ave}$ (m/s^2)	P_a(%)	P_d(%)	P_c(%)	P_i(%)
Criteria	48.859	0.8000	40.11	41.69	8.20	10.00
Cycle	48.655	0.8188	40.69	41.84	7.59	9.89
Error(%)	0.42	2.35	1.44	0.36	7.47	1.15

Table 5: Comparison between target parameters of criteria and generated cycle with χ_1.

6.3 CO_2 Emission Calculation

Given the constructed driving cycles and energy consumption model, we can calculate the energy consumption for EVs with different driving conditions. According to global carbon intensity in electricity generation data [8], the corresponding CO_2 emissions of the power grid are variant in different countries. Table 6 compares the average energy consumption of Chevrolet Volt with different driving conditions (high-level attributes) and the equivalent CO_2 emission in different countries, in which Column 2 shows the energy consumed per kilometer (in kWh/km), and Column 3 to 5 are equivalent CO_2 emission in three countries respectively (in g CO_2/km). Compared to the average CO_2 emission of ICE light duty vehicles in US (255 g CO_2/km) [14], replacing ICE vehicles with EVs may help to reduce CO_2 emission. Besides, grid carbon intensity shows a much bigger impact on CO_2 emission than high-level attributes in real driving.

High-level attributes $\boldsymbol{\chi}$	Energy consumption	US	China	France
$[\ 1\quad 0.15\quad 0.4\quad 1\ ]$	0.1497	82.63	114.67	11.83
$[\ 1\quad 0.1\quad 0.6\quad 1.5\ ]$	0.1665	91.91	127.54	13.15
$[\ 1\quad 0.15\quad 0.8\quad 3\ ]$	0.1801	99.41	137.96	14.23

Table 6: Average energy consumption (in kWh/km) and equivalent CO_2 emission (in g CO_2/km) in three countries.

7. CONCLUSIONS

We have described two methods for obtaining data from a vehicle's OBD port which are useful for building models of velocity inputs which can then further be extended to characterize energy and emissions of electric vehicles. Various useful techniques which researchers may employ during their efforts in CAN bus information extraction were detailed, for example the approach to reverse-engineering CAN bus at the data link layer to determine important parameters. We have explored one application of the data gathered using the techniques described, in this case the synthesis of driving cycles from real-world driving, and their application in order to determine the influence of various parameters on electric vehicle energy consumption and therefore emissions.

8. REFERENCES

[1] J. Allande, P. Tyler, and E. Woodruff. Nissan Leaf On-BoardDiagnostic Bluetooth Utility. Technical report, Cal Polytechnic State University, 2012.

[2] C.-K. Chau, K. Elbassioni, and C.-M. Tseng. Fuel minimization of plug-in hybrid electric vehicles by optimizing drive mode selection. In *ACM eEnergy*, 2016.

[3] S. Corrigan. Introduction to the Controller Area Network (CAN). Technical report, Texas Instruments, 2008.

[4] M. Di Natale. Understanding and using the controller area network. 2008.

[5] E. Electronics. ELM327 OBD to RS232 Interpreter, 2015.

[6] L.-B. Fredriksson. Controller area networks and the protocol can for machine control systems. *Mechatronics*, 4(2):159–172, 1994.

[7] R. K. Ganti, N. Pham, H. Ahmadi, S. Nangia, and T. F. Abdelzaher. GreenGPS: a participatory sensing fuel-efficient maps application. In *ACM MobiSys*, 2010.

[8] IEA. CO2 emissions from fuel combustion highlights 2012, 2012.

[9] S. International. E/e diagnostic test modes, 2012.

[10] OBDSolution. STN1110 vs ELM327 Comparison, 2010.

[11] C.-M. Tseng, C.-K. Chau, S. Dsouza, and E. Wilhelm. A participatory sensing approach for personalized distance-to-empty prediction and green telematics. In *ACM eEnergy*, 2015.

[12] C.-M. Tseng, S. Dsouza, and C.-K. Chau. A social approach for predicting distance-to-empty in vehicles. In *ACM eEnergy*, 2014.

[13] US EPA. Basic information | on-board diagnostics(OBD), 2015.

[14] US EPA. Greenhouse gas emissions from a typical passenger vehicle, 2015.

[15] C. Valasek and C. Miller. Adventures in Automotive Networks and Control Units. Technical report, IOActive, 2014.

[16] E. Wilhelm, J. Siegel, S. Mayer, L. Sadamori, S. Dsouza, C.-K. Chau, and S. Sarma. CloudThink: A scalable secure platform for mirroring transportation systems in the cloud. *Transport*, 30(3), 2015.