energies

MDPI

Article

A Canopen-Based Gateway and Energy Monitoring System for Electric Bicycles

Van-Tung Bui [1], Chyi-Ren Dow [1,*], Yu-Chi Huang [1], Pei Liu [2] and Vu Duc Thai [3]

1 Department of Information Engineering and Computer Science, Feng Chia University, Taichung 40724, Taiwan; bvtung@mail.fcu.edu.tw (V.-T.B.); M0328107@mail.fcu.edu.tw (Y.-C.H.)
2 Department of Transportation and Logistics, Feng Chia University, Taichung 40724, Taiwan; peiliu@mail.fcu.edu.tw
3 Faculty of Information Technology, Thai Nguyen University of Information and Communication Technology, Thai Nguyen 24000, Vietnam; vdthai@ictu.edu.vn
* Correspondence: crdow@mail.fcu.edu.tw

Received: 21 June 2020; Accepted: 20 July 2020; Published: 22 July 2020

Abstract: The limitation of battery capacity is a cause of range anxiety that reduces the wide use of electric bicycles (e-bikes). Therefore, many works have developed systems that provide assistance to cyclists to deal with the range anxiety problem. However, these systems may have limited applications since they can only work with the e-bike manufacturers' hardware and communication protocols. This paper proposes an energy monitoring system (EMS) for e-bikes, which is based on EnergyBus, a standardized hardware and communication protocol for e-bikes. EnergyBus standard is based on controller area network (CAN) bus and CANopen protocols. EMS comprises a gateway connected to EnergyBus of e-bike and an EMS application installed on a smart device that connects to the gateway via Bluetooth. The gateway provides CAN bus monitoring and CANopen device data access services to the smart device. These services are modeled to determine gateway parameters to ensure the efficient performance of the gateway and to keep the working status of the monitored e-bike safe. The EMS application provides the cyclist information about battery status, rider efforts, and other related information such as distance and speed. Experimental results show that the proposed gateway can monitor data in real-time and ensure monitored system safety.

Keywords: CAN bus; CANopen; EnergyBus; e-bike; gateway; energy monitoring

1. Introduction

Transportation has a high impact on issues of air pollution, energy security, and global warming. According to [1], combustion engine vehicles produce 25% of greenhouse gas emissions. Especially, this impact is even higher in the city where there is a high density of vehicles. This has spiked interest in research to develop electric vehicles (EVs) to alter the traditional means of transportation. In particular, electric bicycles (e-bikes) have many advantages compared to others. An e-bike is a traditional bicycle integrated with a battery unit and an electric motor to assist the cyclist while pedaling. With lightweight and small footprint characteristics, e-bikes are the greenest vehicles among EVs. According to [2], e-bikes are less energy consuming and carbon-dioxide producing than other types of vehicles. With the power-assisted pedal, riding an e-bike can reduce physical effort yet still improve the health of the rider, thus attracting people of every age to the physical activity of e-bike cycling.

Despite the advantages, it generally takes more than two hours to fully charge a battery. Such a long waiting time may be unbearable for impatient users. In addition, the current limited battery capacity limits the travel range of an e-bike with one single charge. Therefore, range anxiety is a big issue reducing the wide use of e-bikes. To alleviate the range anxiety, a human–machine interface

(HMI), which states the battery status (such as remaining capacity, state of health) and the estimated range autonomy, is an indispensable component of e-bikes. However, due to low computing capacity and poor connectivity characteristics of the electronic system of e-bikes, the information shown on the HMI is generally inaccurate and not useful for inexperienced cyclists. Consequently, existing works have developed systems using smartphones to connect with e-bikes to provide assistance to cyclists, such as speed advisor [3], route guidance, range prediction [4], etc. In addition, the data collected from monitoring the e-bike battery in the real-world operating conditions can be used to improve the accuracy of battery status prediction [5]. Therefore, the interface device, which can be called the gateway connecting the smartphone with the e-bike, plays an important role. However, previous gateways were seldom designed for e-bikes equipped with electronic devices using a standardized communication.

Each e-bike manufacturer has his own hardware design, resulting in a high maintenance cost. It is also hard to build public charger stations suitable for various types of e-bike batteries. To address this issue, EnergyBus and CAN in Automation (CiA) organizations have proposed a standard for the electronic components of e-bikes, called EnergyBus standard [6–8], which is based on CANopen protocol [9] and specified in CiA454 document named 'CANopen application profile for energy management systems' [10]. CiA454 defines communication protocols, device profiles, and a unified connector for e-bike electronic devices connected by a CAN network. However, the standard gateway profile of the EnergyBus application is still under development. Although CiA has specified a standard for the gateway [11], designing an efficient gateway with low cost and ensuring system safety is still a big challenge.

In this paper, an energy monitoring system (EMS) for e-bikes based on EnergyBus is proposed. The system allows cyclists to use a smart device to obtain information about battery status, rider efforts, and other related information such as distance and speed from their e-bike. In addition, the rider's trip is also tracked using the built-in GPS of the smart device. Moreover, all the information is saved in the smart device's storage for the riders to review their physical activities. The EMS consists of an EMS App installed on a smart device and a CANopen-based gateway connected to the e-bike's EnergyBus. The EMS App communicates with the gateway through Bluetooth. The gateway works on both data link layer and application layer of the data transmission model, providing CAN bus monitoring and CANopen device data access services to the smart device, respectively. The design of the gateway is proposed to ensure the performance of data monitoring and the transparency of data access for the smart device while keeping the e-bike electronic system safe. A system prototype is implemented to prove the feasibility and performance of the system. Experimental results show that the proposed gateway based on a low-cost microcontroller can monitor data in real-time without losing data and ensure safety of the monitored system. The rest of this paper is organized as follows: Section 2 discusses existing works. Section 3 defines the problem of EMS and proposes our design to solve the problem. System implementation and prototype are demonstrated in Section 4. Experimental results are discussed in Section 5. Finally, Section 6 concludes the paper.

2. Related Work

Several smartphone-based systems have been developed for riders to improve the cycling experience. In these systems, an interface device between e-bike and smartphone is required. In [3], the authors proposed a wind-aware speed advisory system for e-bikes. The system continuously monitors e-bike speed. When e-bikes are approaching a signaled intersection, the system communicates with the traffic light controller via the infrastructure-to-vehicle communications to receive the current traffic light state and wind information. Based on the information received, the system recommends the optimal speed for cyclists to avoid high-power consumption scenarios (i.e., having to stop and start at the intersections due to red traffic light signals), thus reducing the energy consumption of the e-bike. In [4], a mobile cockpit system is presented to provide useful information to cyclists such as the battery state of charge (SoC), battery state of health (SoH), range prediction, and route guidance. The system collects information about the battery state, cyclist's efforts (cadence and torque), route, and weather to

build cyclist's profile. The range autonomy prediction is calculated based on the current battery state, route taken, weather, and cyclist's profile. In order to allow smartphones to be able to monitor the e-bike's devices, interface devices are needed. However, the aforementioned works did not take into account the importance of designing interface devices.

Several studies have considered the design and implementation of interface devices. In [12], the authors introduced an analysis of e-bike data usage collected through the WeBike project to identify riders' behavior. The data include e-bike trips and battery status, which are collected by smartphone in real-time and are uploaded to the server whenever the smartphone is connected to the Internet. For the smartphone to be able to obtain the battery status, an interface device, a phidget board [13], is used for communication between the smartphone and the battery monitoring sensors. Song et al. [14] designed a novel system to monitor e-bike battery using smartphones. The system consists of a battery control unit and an Android client. The battery control unit monitors the battery status (temperature, voltage, current, SoC, SoH) and controls battery charging and discharging processes. The Android client can receive the battery status and send control commands to the battery control unit via Wi-Fi. In [15], the authors proposed a smart e-bike monitoring system to monitor the data about power assistance level and location. The system includes an interface board, a smartphone, and a server. The interface board directly monitors the assistance level and sends it to the smartphone via the universal serial bus. The smartphone uploads the data to the server for data analysis. However, these studies may have only limited applications since they can only work with the manufacturers' hardware and communication protocols.

In [16], an Internet of Thing (IoT)-based cycling training solution is proposed. A trainer can monitor information about cyclist's activities (such as speed, cadence, and power generated by the cyclist) online through a Web application. This information is published by hardware installed on the traditional bicycle. The hardware including sensors (speed, cadence, and force) and an interface device between sensors and server was designed. The interface device is responsible for collecting and processing data from sensors, which it then sends to the server via the internet. In this system, the interface device can be applied to regular bicycles, however, it is not suitable for e-bikes which already have those appropriate sensors installed.

In [17], an architecture for e-bikes control is proposed. The system includes e-bike electronic devices, a gateway, and a smartphone. The e-bike devices are connected by CAN bus. The gateway acts as a central device and has CAN bus and Bluetooth interfaces for communication between the e-bike devices and the smartphone or joystick. The gateway periodically transmits messages which carry the data about battery SoC, motor temperature, e-bike speed, etc. to the application installed on the smartphone. The smartphone application presents the information received from the gateway to the rider.

Table 1 shows the feature comparison of prior works described above for e-bike monitoring with our system. For monitored data, the prior works only collect limited kinds of data, such as speed, battery state, assistance level, cyclist effort, and temperature but our system can access any data from the e-bike electronic system, such as the status of the energy sources to the users. For communication protocols, the above systems are based on IEEE 802.11p, Bluetooth, USB, WiFi, UART, MQTT, and CAN [3,14–17], but our system is based on both Enerygybus/CANopen/CAN and Bluetooth. Our design ensures that the gateway can be used for a smart device to monitor any CANopen system in real-time without affecting the functionality of the monitored system.

On the other hand, CANopen communication protocol complements the application layer for CAN bus. CANopen specifications are provided by CiA301 [9] and CiA302 [18]. A CANopen device is modeled into three components, including communication units, object dictionary (OD), and application program. Communication units provide communication service objects, such as network management (NMT), process data object (PDO), service data object (SDO), synchronization (SYNC), emergency (EMCY), and heartbeat (HB). Each communication object is identified by a CAN-ID. The OD is the interface between the application and the communication services. The OD stores

information about the configuration of services for communication units and variables of the node. Each object in the OD is addressed by a 16-bit index and an 8-bit sub-index. Based on the CANopen core defined in CiA301 and CiA302, some CiA draft standards define additional services for CANopen, such as CiA305 [19] for node-ID assigning services and CiA320 [20] for sleep/wake-up services. CiA also defines standards for CANopen application profiles such as CiA454 [10] for energy management system of EnergyBus. The CANopen gateway can be developed based on specifications CiA309 [11], CiA302-7 [18], CiA315 [21], and CiA457 [22].

Table 1. Comparison of prior works for e-bike monitoring applications and our system.

Features	Monitored Data	Communication Protocol
Reference [3]	Speed	IEEE 802.11p/Bluetooth
Reference [4]	Battery state Cyclist effort	Bluetooth
Reference [12]	Battery state	USB
Reference [14]	Battery state	Wi-Fi
Reference [15]	Assistance level	UART
Reference [16]	Speed Cyclist effort	USB/MQTT
Reference [17]	Battery state Temperature Speed	CAN Bluetooth
Our system	Full electronic system data	EnergyBus/CANopen/CAN Bluetooth

3. Energy Monitoring System

The proposed EMS for EnergyBus-based e-bikes is discussed in this section. Section 3.1 presents overview of the EMS. Section 3.2 shows the EMS architecture in which the system is divided by layers, such as data link, CANopen services, OD, and EMS App. The details of layers are then specified in Sections 3.3–3.6.

3.1. System Overview

This section presents an overview of the proposed EMS for e-bikes is in Figure 1. The EMS consists of an e-bike with its currently installed EnergyBus-based electronic system, a gateway connected to the EnergyBus, and an EMS application installed on a smart device, which connects to the gateway via Bluetooth. The aim of EMS is to obtain energy data from the EnergyBus devices of the e-bike as well as the current location from the GPS sensor in the smart device and provide useful energy-related information to the user. The energy data, location, and timestamp are then combined into tracking logs that may be useful for further trip analysis.

The electronic system of an EnergyBus-based e-bike [23] includes a motor control unit (MCU), a battery management system (BMS), a braking sensor unit (BSU), a pedal sensor unit (PSU), a human-machine interface unit (HMI), and an EnergyBus controller (EBC). These devices use the EnergyBus protocol to communicate and can be called nodes of the network. When the system is running, nodes use the PDO services to transmit/receive process data to/from other nodes. The PDOs are synchronously transmitted after each SYNC message that is broadcasted by the EBC. In addition, the EBC uses the NMT services to manage the network and uses the SDO services to access data in the OD of nodes. The EBC also is an SDO manager that provides the SDO channels to other nodes, allowing them to access each other's OD.

In our EMS, the energy-related information, including the battery status and the cyclist's effort, is provided by the BMS and PSU, transferred through the CANopen network by means of the PDOs, and forwarded to the EMS application by the gateway. Due to incomplete energy information in the process data, the EMS application needs to access the OD of the nodes (i.e., BMS) for further energy information. The process is completed with the help of the gateway.

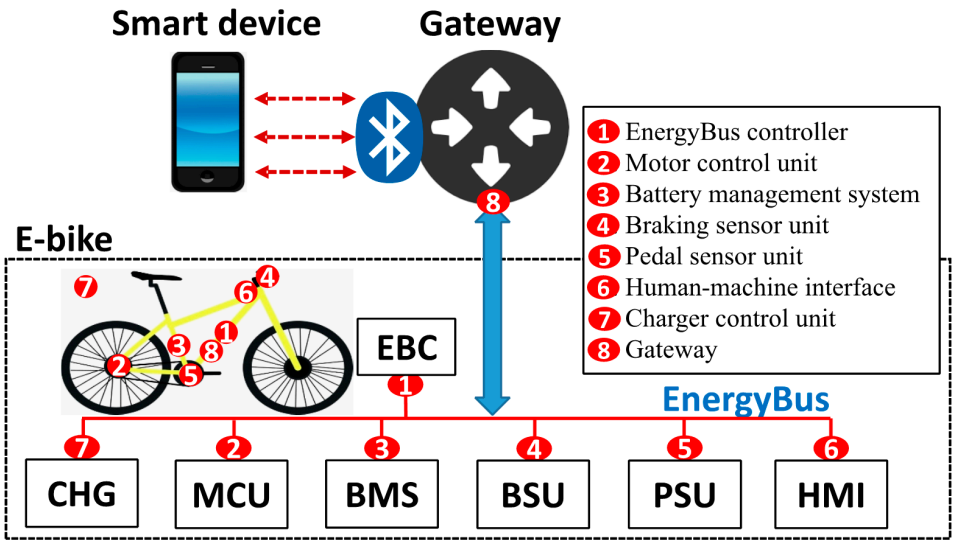


Figure 1. System overview.

The heart of the proposed EMS is the communication gateway, which enables message exchange between the e-bike and the smart device. The gateway is installed on the e-bike and has two interfaces: the CANopen interface, connected to the EnergyBus of e-bike and the Bluetooth interface, enabling wireless connection with the smart device. The gateway forwards the PDO messages which transfer energy-related data from the CAN bus to EMS App. In addition, the gateway provides the SDO services for the EMS App to access data from the OD of any CANopen node. This data access process increases bus load of the CAN bus. If bus load is increased too high, it will increase the message latency causing the loss of messages which affects the correct functionality of the e-bike electronic system. Therefore, the design of gateway must not only ensure data exchange performance but also ensure that the CAN bus load does not increase excessively.

The gateway is an embedded device, including a controller which can efficiently communicate with the CAN bus and RS232, and a Bluetooth module interfacing with the controller through RS232. Figure 2 shows the hardware block diagram of the gateway.

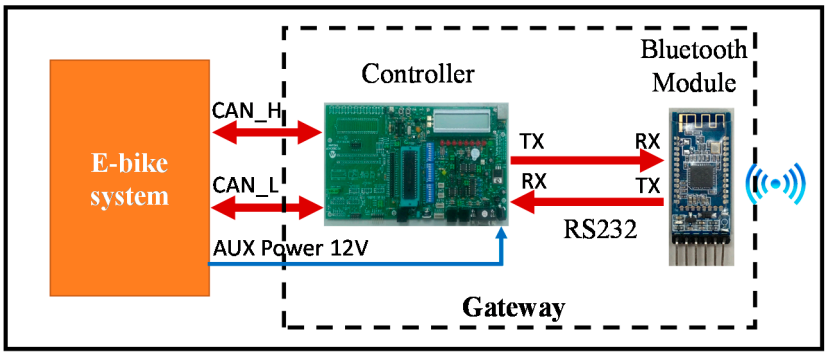


Figure 2. Hardware block diagram.

3.2. EMS Architecture

Figure 3 shows the system architecture of the EMS. At the Bluetooth side, the CANopen is also employed for communication between the smart device and gateway at application layer. Due to the Bluetooth/RS232 working at physical layer, a data link layer (DLL) is added between these two layers. Thus, the model of all devices in the EMS is the model of CANopen devices that includes the CAN message transmitter, CANopen stack, object dictionary, and application program.

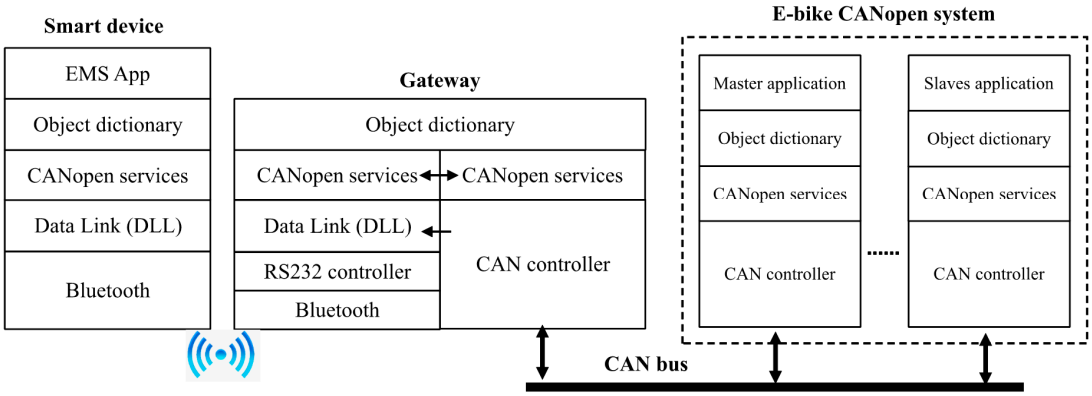


Figure 3. Energy monitoring system (EMS) architecture.

The gateway enables data exchange between a CANopen system and a smart device via Bluetooth. As introduced in Section 3.1, the gateway provides for the PDO forwarding and OD accessing services. Because the gateway must not understand the process data objects of the monitored system, the first service is done at DLL where the performance is higher than at the application layer. In this service, one-way messages coming from the CAN bus are monitored, filtered, and forwarded to the RS232 for smart devices. The filter ignores the unwanted CAN messages which affect the performance of smart device. Therefore, a smart device can receive any CAN message it needs. Then, the messages are processed at its CANopen layer. For example, smart device can use the PDO services to receive real-time process data from nodes. Thus, the PDO forwarding service is changed into a CAN messages forwarding service. The second service is done at the CANopen layer. The smart device uses the SDO services to access the OD of any node (including gateway). With this design, the gateway can limit the influence of smart device to an e-bike electronic system, and the gateway can be used for the smart device to monitor any CANopen system.

3.3. Data Link Layer

The main function of the DLL is the framing function for transmitting the CAN messages between smartphone and gateway. To transmit CAN messages through the Bluetooth/RS232, CAN-UART frames are used. Each CAN-UART frame carries a CAN message. The DLL encapsulates the CAN message into a CAN-UART frame and vice versa. Each CAN message contains an 11-bit message identifier (ID) (8 low bits IDL and 3 high bits IDH), a 4-bit message length (DLC) and up to 8 data bytes [24]. The format of the CAN-UART frame, as shown in Figure 4, consists of a header, a CAN message, and 1-byte checksum. The length of the header will be discussed in Section 3.5.

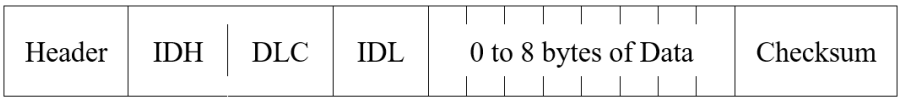


Figure 4. Frame format for the data link layer (DLL).

Figure 5 shows the data flows in the DLL of the gateway. There is one flow for CAN message forwarding service and one flow between the DLL and CANopen layer for the SDO services. For the first flow, the DLL receives every CAN RX message from the CAN bus. Each message then goes to the filter; only messages with an allowed ID (i.e., PDO messages) can pass to queue up in the receive (RX) queue. Next, each message is de-queued from the RX queue and packed into the CAN-UART frame. Each byte of the CAN-UART frame is then sent to UART transmit (TX) buffer to wait to be transmitted. Finally, each CAN message is transmitted to the EMS App's DLL and then is passed to the CANopen layer. In short, data of the CANopen devices on the CAN bus side have been transmitted to the EMS App.

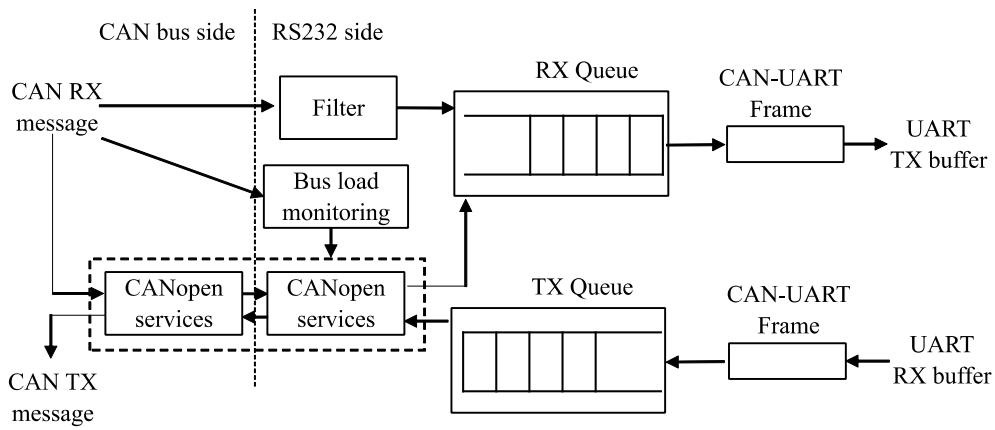


Figure 5. Data flows in DLL of the gateway.

For the second flow, first, the request message is sent from the smart device and enters the gateway via Bluetooth. Request messages are queued in the TX queue before going to the CANopen services layer. The CANopen layer then completes the SDO services on the CAN bus side with the target device. Finally, the response message returned from the CANopen layer is queued up in the RX queue to wait to be transmitted to the smart device.

The gateway must control the throughput of the second flow to keep the CAN bus load in a safe value. Therefore, the CAN bus load is a feedback value of the CANopen layer. A bus-load-monitoring module is added to collate statistics of all messages coming from the CAN bus to calculate the immediate CAN bus load.

Since many devices on the e-bike can concurrently send messages to the CAN bus, we designed a message-acceptance filter based on the message ID for letting only messages with specific allowed IDs to go through the gateway. Based on our monitoring purpose, in the configuration phase, we defined an array containing IDs of messages that we wanted to monitor (e.g., battery status messages, cyclist effort messages). This filter mechanism restricts the flood of data from multiple unwanted devices which cause an overflow in gateway.

An object, the RXFilter, is defined in the gateway's OD to store filter configuration. As shown in Figure 6, the RXFilter is an array with 64 elements. Since an expedited SDO can transfer up to 4 bytes of data, we defined each element of the RXFilter as 32 bits. Hence, the filter can handle a system with 2048 specific CAN-IDs (all possible cases of 11-bit message IDs). For configuration, if the *i*th bit in the RXFilter is set to 1, the CAN messages with ID *i* can go through the filter, as defined in Table 2. When the gateway receives a CAN message that has ID *i*, the indexes of *i*th bit in the RXFilter are located at [*row*][*column*], where $row = i \; div \; 32$ and $column = i \; mod \; 32$. For example, for message ID $i = 35$, the filter bit is located at the RXFilter[1],[3]. This conversion allows a fast response of the filter.

RXFilter[0]	Bit 31	…	Bit 1	Bit 0
RXFilter[1]	Bit 63	…	Bit 33	Bit 32
…	…	…	…	…
RXFilter[63]	Bit 2047		Bit 2027	Bit 2026

Figure 6. Structure of CAN RX message filter object.

Table 2. Value definition of every RXFilter bit.

Name	Default Value	Value	Description
Bit i	0	0	CAN RX message ID i is not acceptable
		1	CAN RX message ID i is acceptable

3.4. CAN Message Forwarding Service Model

As discussed in Section 3.3, there are parameters for the gateway, namely, CAN bus load, header length of CAN-UART frame, and RX queue length. To determine these parameters and to evaluate the performance of the message forwarding service, the MM1 model [25] is used to model the CAN message forwarding service in case the filter accepts all CAN messages. The arrival rate λ is the number of CAN messages arriving at the CAN port of the gateway per second. The service rate μ is the number of CAN messages departing to the Bluetooth port per second.

It is noted that we distinguish the concept of CAN message and CAN frame. Each CAN message contains 11-bit message ID, 4-bit message length, and up to 8 data bytes, whereas each CAN frame is a packet of a CAN message used to transmit the message through physical layer.

Assume that the monitored CANopen system uses n different CAN message IDs to transmit data between nodes. Let f_i be the rate of message m_i ($1 \le i \le n$). The arrival rate λ can be calculated by

$$\lambda = \sum\nolimits_1^n f_i. \tag{1}$$

Let L_i (bits), L_{Di} ($0 \le L_{Di} \le 8$ bytes), and R_I be the length of CAN frame m_i, the data length of CAN message m_i, and the baud rate of CAN bus, respectively. The bus load α can be calculated by

$$\alpha = \frac{\sum_1^n L_i f_i}{R_I}, \tag{2}$$

where L_i can be calculated by the worst case length of CAN frame m_i [26].

$$L_i = 47 + 8L_{Di} + \left\lfloor \frac{33 + 8L_{Di}}{4} \right\rfloor = 55 + 10L_{Di}. \tag{3}$$

Let $\overline{L}$ (bits) and $\overline{L_D}$ ($0 \le L_{Di} \le 8$ bytes) be the average length of a CAN frame and the average data length of a CAN message, respectively.

$$\overline{L} = \frac{\sum_1^n L_i f_i}{\sum_1^n f_i} = 55 + \frac{10\sum_1^n L_{Di} f_i}{\sum_1^n f_i} = 55 + 10\overline{L_D}. \tag{4}$$

By combining calculations in Equation (4) for Equation (1), we have:

$$\lambda = \frac{\sum_1^n L_i f_i}{55 + 10\overline{L_D}} \tag{5}$$

$$\lambda = \frac{\alpha R_I}{55 + 10\overline{L_D}} \tag{6}$$

The service process includes dequeuing, packaging, and transmitting. The dequeuing and packaging are processed for the next CAN-UART frame while each byte of the current CAN-UART frame is transmitted at the UART controller, assuming that the time for dequeuing and packaging processes is less than that for the CAN-UART frame transmitting process. Therefore, the service rate μ can be calculated by

$$\mu = \frac{1}{t_{tr}}, \tag{7}$$

where t_{tr} is the average time required to transmit a CAN-UART frame.

$$t_{tr} = \frac{l}{R_o}\overline{L_{mo}}, \tag{8}$$

where R_o is the baud rate of UART, l is the length of a UART frame (e.g., l = 10 bits for 1 start bit, 8 data bits, and 1 stop bit), and $\overline{L_{mo}}$ is the average length of a CAN-UART frame. As the CAN-UART frame structure described in Section 3.3,

$$\overline{L_{mo}} = L_{header} + \overline{L_D} + 3. \tag{9}$$

The stability condition of the MM1 queue is $\lambda \leq \mu$. Thus,

$$\frac{\alpha R_I}{55 + 10\overline{L_D}} < \frac{1}{\frac{l}{R_0}\overline{L_{mo}}}. \tag{10}$$

Therefore,

$$\overline{L_{mo}} < \frac{\left(55 + 10\overline{L_D}\right)R_o}{\alpha R_I l}. \tag{11}$$

The average length of the RX queue is $\overline{Q}$

$$\overline{Q} = \frac{\rho^2}{1-\rho} = \frac{\left(\frac{\lambda}{\mu}\right)^2}{1-\lambda/\mu} = \frac{\lambda^2}{\mu(\mu-\lambda)}. \tag{12}$$

In short, the bus-load-monitoring module can use Equations (2) and (3) to calculate the CAN bus load α. In the design phase, header length of CAN-UART frame and length of RX queue are required. For the header length, $\overline{L_{mo}}$ must be determined first by satisfying Equation (11), then, L_{header} is found by Equation (9). The RX queue length is estimated by Equation (12).

3.5. CANopen Layer

In this section, we discuss the CANopen services for the smart device. Figure 7 highlights two main CANopen services of EMS: SDO and EMCY. The EMCY producer service is used for the gateway to report its errors, if any, such as TX queue/RX queue overflow.

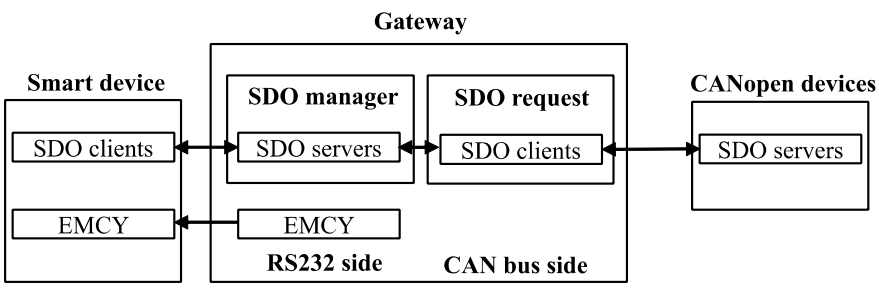


Figure 7. CANopen services of EMS.

The SDO services of the gateway include couples of the SDO server and SDO client to adapt requests from the SDO clients of the smart device. Each couple works as a relay to serve an SDO client. It may have a maximum of 127 nodes in a CANopen network. Therefore, the gateway needs to provide 127 couples. The default channels are used for the SDO communication between the smart device and gateway. In particular, the smart device can use the default SDO channel to communicate with the desired node. The channels used for SDO communication between the gateway and nodes on the CAN bus side are provided by dynamic SDO request service [18,27]. In the configuration phase, the 127 couples are initialized as follows: the SDO server channels are assigned to default channels of

node IDs from 1 to 127 and the SDO client channels are assigned to invalid values. Each SDO client channel of the gateway will be updated when the gateway serves an SDO client of the smart device for the first time. There are not enough valid channels for all SDO clients of the gateway at the same time, therefore, the gateway can use SDO release processes to release an SDO channel if it cannot request a new channel.

These SDO services increase the CAN bus load α introduced in Section 3.4. The worst case is that every CANopen node on the CAN bus side has only one default SDO channel, meaning that the SDO clients of gateway can only access nodes through the CANopen master (EBC). Then, if F is the throughput of SDO commands sent from SDO clients of the smart device, the CAN bus load changes from α to α' as follows:

$$\alpha' = \alpha + \frac{4F * L_{SDO}}{R_I} = \alpha + \frac{4F * (55 + 10 * 8)}{R_I} = \alpha + \frac{540F}{R_I}, \tag{13}$$

where L_{SDO} is the maximum length of an *SDO* message.

In fact, we have the constraint that new CAN bus load should be less than M (maximum value for bus load), that is,

$$\alpha + \frac{540F}{R_I} \leq M \tag{14}$$

$$F \leq (M - \alpha)\frac{R_I}{540}. \tag{15}$$

During the operation, the gateway periodically checks for and executes the SDO services if an SDO request from the smart device is available. F is used to determine the SDO cycle $1/F$ at which the gateway can send an SDO command to the system.

3.6. System Operation Flow

Figure 8 shows the operation flow of our EMS. First, the Bluetooth connection between the smart device and gateway needs to be established. Then, at the first run time, the EMS App will configure the filter of the gateway DLL. To configure the filter, the EMS App uses SDO services to update the RXFilter object in the gateway's OD. Finally, the EMS App can monitor, present, and record energy data taken from the e-bike. The real-time energy information, including the battery status (such as the remaining capacity, voltage, current, and temperature) and rider effort (such as the torque and pedal speed), is located in the OD of the EMS App and updated automatically by the PDO service. Further energy information about the detailed battery status (such as the state of health and cell voltage) is not transmitted by the PDOs in the EnergyBus; therefore, the EMS App uses the SDO services to obtain this information.

In the task of data presentation, the battery remaining time, instant range, and rider energy consumption are also considered. Let E_{rm}, P_{cs} be the battery remaining energy and the power consumption, respectively. E_{rm} and P_{cs} can be calculated as follows:

$$E_{rm} = SoC * SoH * E_r, \tag{16}$$

$$P_{cs} = V * I, \tag{17}$$

where SoC (state of charge) is the battery remaining capacity in percentages, SoH is the state of health of the battery in percentages, E_r is the full energy capacity of the battery when it is manufactured, V is the battery voltage, and I is the battery current. The remaining time $T_{r,}$ of the battery can be calculated as follows:

$$T_r = \frac{E_{rm}}{P_{cs}}. \tag{18}$$

The instant range d, based on the current e-bike speed v and battery remaining time T_r, can be calculated as follows:

$$d = v * T_r. \tag{19}$$

Let P_{rd} be the rider power and Δt be the sampling period. The rider energy consumption E_{rd} can be given as follows:

$$E_{rd} = \sum P_{rd} * \Delta t. \tag{20}$$

Let T_{rd}, ω_{pd} be the torque and pedal speed, respectively. The rider power P_{rd} can be given as follows:

$$P_{rd} = \frac{T_{rd} * 2\pi * \omega_{pd}}{60}. \tag{21}$$

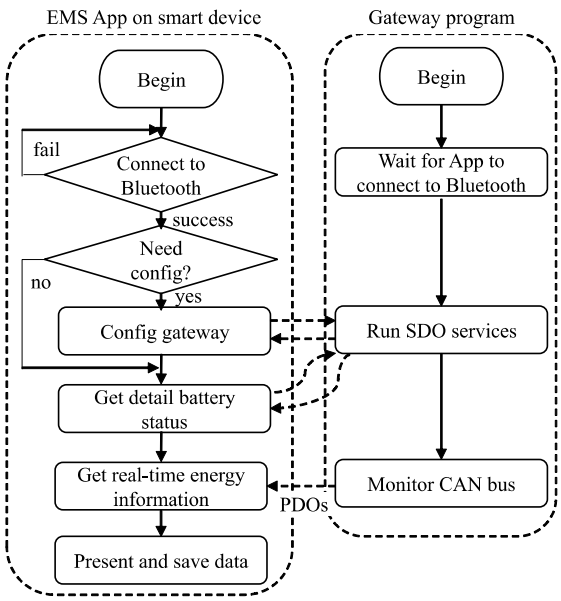
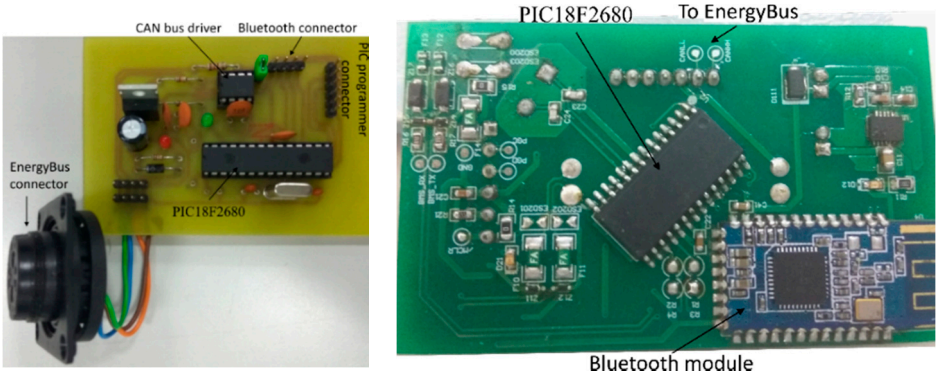


Figure 8. Operation flow of EMS.

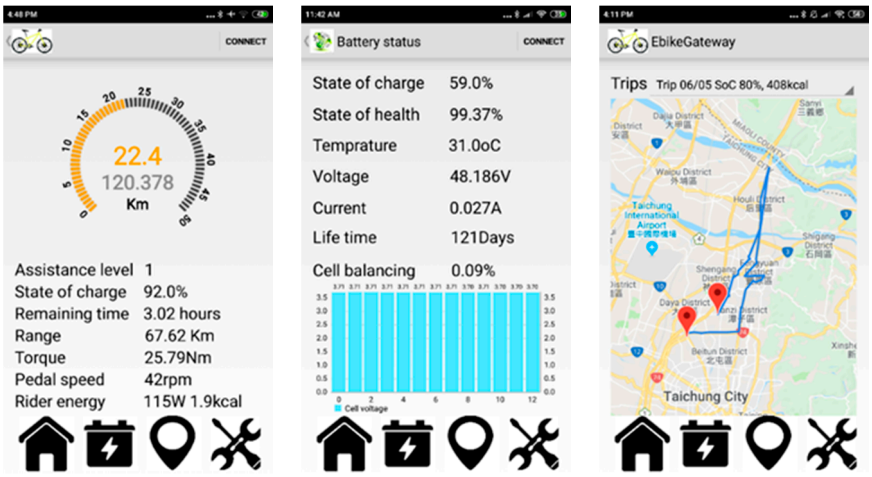
4. System Implementation and Prototype

We implemented a gateway and an EMS App to realize the proposed EMS. The gateway prototype includes a peripheral interface controller (PIC) 18F2680 40-MHz low-cost microcontroller and HM10 Bluetooth module, as shown in Figure 9. The PIC microcontroller is programed by a MPLAB X IDE [28] with the C18 compiler [29]. The CANopenNode v1.10 [30], which is a CANopen open source program for microcontrollers, is adopted to program the PIC18F2680. The HM10 Bluetooth module [31] is a Bluetooth low energy module interfacing with the microcontroller and smartphone through the UART serial communication and Bluetooth, respectively. The EMS App is implemented by an Android Studio IDE. The EMS App prototype, as shown in Figure 10, includes aggregate monitor, battery monitor, and history data report interfaces. The aggregate monitor interface is the main interface. The battery monitor interface shows battery status in detail, and the history data report interface presents the rider's past physical activities. Riders can use the EMS for EnergyBus devices such as the battery (Figure 11a), the battery and charger (in charging process) (Figure 11b), or the e-bike (Figure 11c).



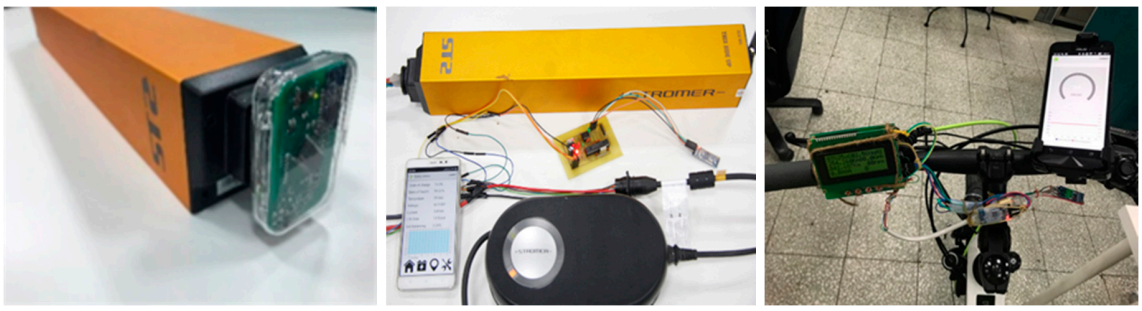
(**a**) (**b**)

Figure 9. Gateway prototypes. (**a**) Version 1.0. (**b**) Version 2.0.



(**a**) (**b**) (**c**)

Figure 10. EMS App prototypes. (**a**) Aggregate monitor, (**b**) battery monitor, (**c**) history data report.



(**a**) (**b**) (**c**)

Figure 11. System prototypes. (**a**) Battery monitoring, (**b**) battery charging monitoring, (**c**) e-bike monitoring.

To demonstrate the monitoring functions of the EMS, we conducted two scenarios for battery charging and battery discharging. In the charging monitoring scenario, we used an ST2 battery and a charger manufactured by TD HiTech Energy Inc. The ST2 battery is a 48 V, 17 Ah, 814 Wh lithium-ion battery. In the discharging monitoring scenario, we used a real e-bike equipped with the ST2 battery. To evaluate the gateway, we simulated an EnergyBus e-bike system [23] that consists of nodes, such as EBC, BMS, MCU, HMI, BSU, PSU, a light control unit, and a security unit, as shown in Figure 12. We used Micorchip EVM boards APP001 and CANopenNode v1.10 to implement the nodes of the

CANopen system. The configuration of transmit-PDOs (TPDOs) in an EnergyBus system for each node is described in Table 3. Each node has a heartbeat producer that produces a heartbeat message per second. The EBC is the master node that manages the network and produces the SYNC for synchronous TPDOs. The TPDOs are transmitted with every SYNC. Every byte in the data field of every TPDO is 0. The CAN baud rate is 250 k baud.

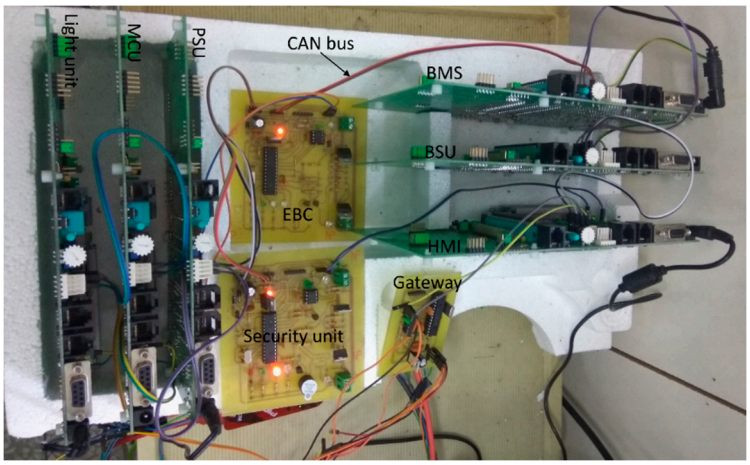


Figure 12. EnergyBus system simulator.

Table 3. Transmit process data objects (T-PDOs) of the EnergyBus system simulator.

Node's Name	TPDOs' CAN-ID (hex)	Length (bytes)
EBC	181	8
	281	8
MCU	18A	8
	28A	8
	38A	8
HMI	1C0	3
PSU	1CD	2
	2CD	6
BSU	1CB	2
	2CB	6
BMS	192	8
	292	8
	392	8
Light	1A6	8
	2A6	8
Security	1C8	4
Gateway	None	-

5. Experimental Results

In this section, two main results are presented: (1) the evaluation of the gateway real-time performance and safety features, and (2) the experimental results of the EMS application.

5.1. Real-Time Monitoring Performance of the Gateway

To determine the parameters of the gateway, including the RX queue length and the UART baud rate, we simulated the relationship between the average data length $\overline{L_D}$ and average RX queue length $\overline{Q}$ with the given CAN baud rate R_I, UART baud rate R_o, and CAN bus load α, as shown in Figure 13. The value of R_I is 250 k. We set up three simulation experiments with three values of R_o, which were 115.2 k, 230.4 k, and 460.8 k, as illustrated in Figure 13a–c, respectively. It is noted that $\overline{Q}$ must be limited with respect to memory available of the microcontroller. In the experiments, we let L_{header} be 3 bytes, let $\overline{L_D}$ vary from 0 to 8 bytes, and tried to slightly increase the α from 10% until it reaches 100% or until $\overline{Q}$ approaches the upper bound, which can cause packet loss (in these experiments, the upper bound was set to 300). $\overline{Q}$ is computed by Equation (12).

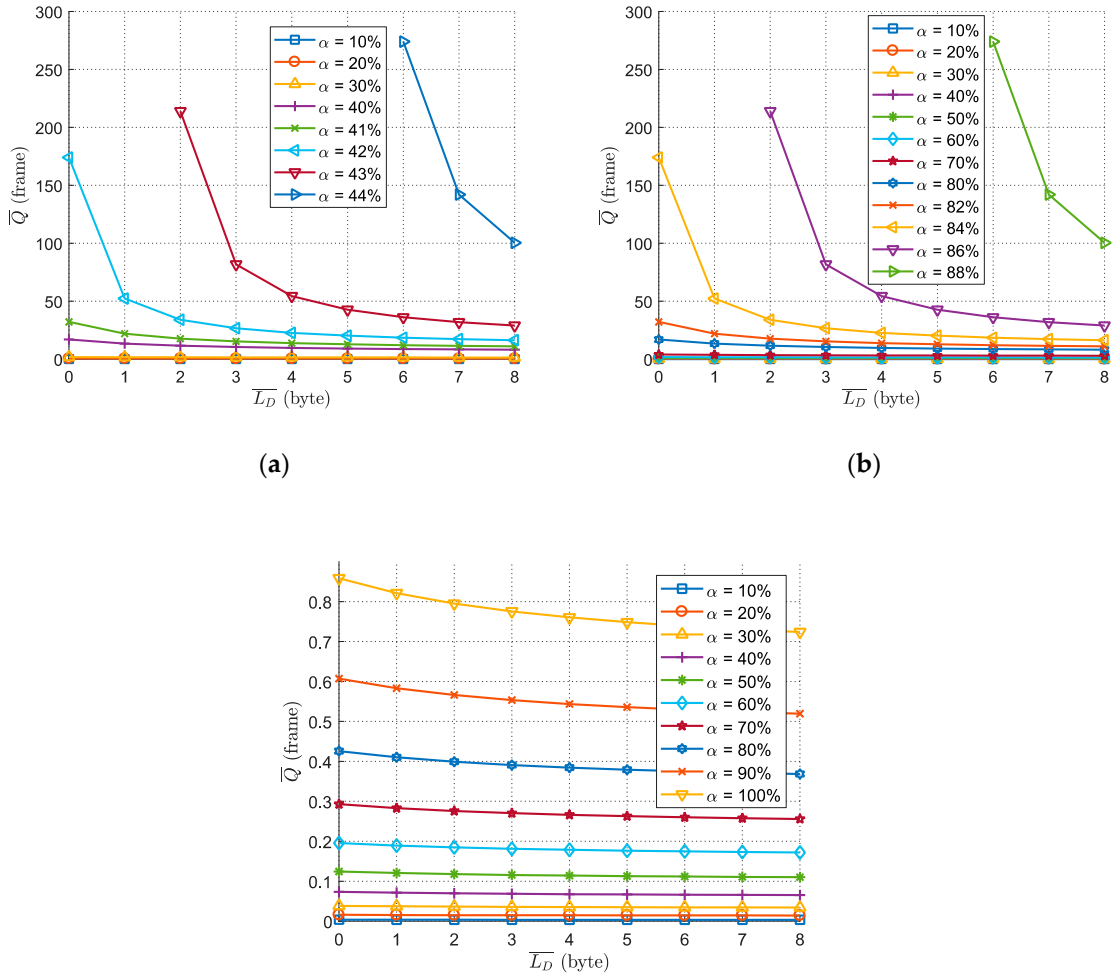


Figure 13. Relationship between average RX queue length $\overline{Q}$ and average length of data $\overline{L_D}$ with a given CAN bus load α and UART baud rates R_o. (**a**) 115,200 baud, (**b**) 230,400 baud, (**c**) 460,800 baud.

The experimental results in Figure 13 show that when $\overline{L_D}$ decreases, $\overline{Q}$ increases, and vice versa. With the UART baud rate of 460.8 k, which is larger than the CAN baud rate of 250 k, the gateway always ensures monitoring a CAN bus load of 100%, as the value of $\overline{Q}$ is kept less than 1 (Figure 13c). However, with the UART baud rate at 115.2 k and 230.4 k, which are smaller than the CAN baud rate of 250 k, the gateway can monitor a CAN bus load of 42% and 84%, as shown in Figure 13a,b, respectively. The average queue length increases dramatically in comparison with bus load and average data length. Therefore, the UART baud rate should be 460.8 k if that speed is available in the Bluetooth module,

otherwise, it should be 230.4 k since at this baud rate, the gateway can monitor a CAN bus load of 84%, which is larger than the usual recommended CAN bus load that does not exceed 80%.

To estimate the RX queue length more accurately, we conducted experiments comparing the difference between average and maximum queue lengths at different bus loads. The average queue length is computed in theory by Equation (12), whereas the maximum queue length is measured in the real world. To obtain the real maximum queue length, we let the gateway continuously report the value of maximum queue length, connected the gateway with the EnergyBus system simulator, and monitored the maximum queue length value. We adjusted the bus load by changing the SYNC cycle in the EBC. In this experiment, the average data length $\overline{L_D}$ was fixed as 6.2, which is the average data length of all messages in Table 3; the UART baud rate was set as 230.4 k and 460.8 k; the bus load was increasing from 60%. As shown in Figure 14, when the baud rate was 460.8 k, the maximum RX queue lengths were stable at one frame. The same stability is found with the maximum queue length when the baud rate was 230.4 k. At each baud rate, the higher the set bus load, the smaller the gap measured between the maximum and average queue lengths. At the maximum bus load that the gateway can monitor, the average queue length is asymptotic to the maximum queue length. Therefore, the RX queue length can be estimated more accurately by Equation (12) with the given maximum bus load.

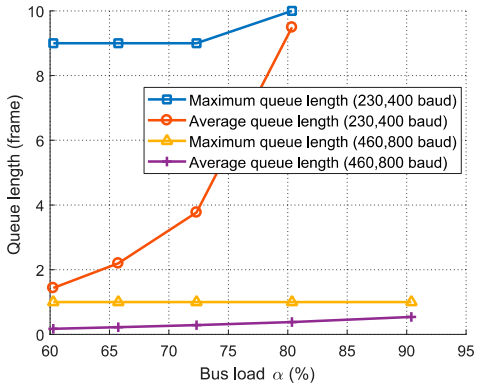


Figure 14. Difference between the average and maximum RX queue lengths.

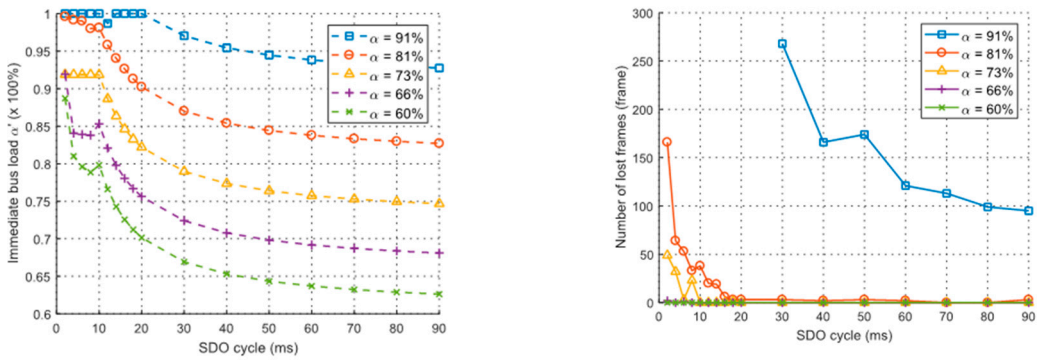
5.2. *Effect of the Gateway's SDO Services on the CAN Bus Side*

During the operation, the smart device can frequently or periodically send SDO commands to access the OD of specific devices of the e-bike. These commands are queued in the TX queue, as shown in Figure 5. The gateway periodically checks two things. The first thing it checks is the TX queue to see whether there is any new requests. The second thing it checks is the execution status of the current SDO command to see it is completed. If both are yes, the new SDO command is executed and puts a new loading into the CAN bus. The period of time for that checking is called the SDO cycle. Of course, the shorter the SDO cycle, the more loading the bus receives.

In this section, we use the EnergyBus system simulator to examine the effect of SDO services on a CANopen system under variations of SDO cycles and original CAN bus load values α, assuming that there are always new requests in the TX queue. The duration of SDO cycles varies from 90 ms down to 2 ms. We adjusted the original bus load by changing the SYNC cycle in the EBC. The experiment was performed in 1800 s. The immediate CAN bus load α' is calculated by Equation (13) based on the original CAN bus load and the addition bus load caused by the SDO services. The number of CANopen TX frames lost in every node is recorded. The results help us to determine which SDO cycles can be used for monitoring a CANopen system.

Figure 15 shows the experimental results. The effect of the gateway's SDO services on CAN bus load α of the e-bike simulator is shown in Figure 15a. In each case of bus load α, starting the SDO cycle from 90 ms, bus load α' is slightly increased at the beginning its gain improves over time. When the SDO cycle was less than 10 ms, the bus load changed unpredictably. This abnormality was caused by

the fact that 10 ms is the smallest time period required to complete an SDO command. When the SDO cycle is less than 10 ms, the gateway must wait for more SDO cycles until the previous SDO command is finished. The actual SDO cycle, therefore, is longer than current SDO cycles.



(**a**) (**b**)

Figure 15. The effect of the gateway's SDO services. (**a**) Effect on CAN bus load, (**b**) effect on frame loss.

The effect of the gateway's SDO services on frame loss on all of the nodes of the e-bike simulator is shown in Figure 15b. The higher the set bus load, the higher the number of lost frames that may occur. With bus load α of 81% and 91%, the frame loss occurs at every SDO cycle. With bus load α of 73% and 66%, the frame loss occurs when the SDO cycle is less than 10 ms and 2 ms, respectively. The number of lost frames is zero at every SDO cycle in the case of bus load α of 60%.

Table 4 summaries the number of lost frames at different values of CAN bus load α and α'. It can be observed that if the original CAN bus load α is not greater than 73% and the CAN bus load α' is less than 85%, then the SDO services will not affect the performance of the monitored system (the number of lost frames is zero). This result satisfies the rule that the CAN bus load of a system should not be greater than 80% ($\alpha \leq 0.8$). Therefore, the SDO cycle should be chosen such that the bus load does not exceed 80% ($M = 0.8$).

Table 4. Summary of the number of lost CAN frames at different values of α and α′.

CAN Bus Load α	CAN Bus Load α'	Number of Lost CAN Frames
91%	From 93% to 100%	From 99 to 2970
81%	From 82% to 100%	From 0 to 166
73%	From 89% to 92%	From 0 to 49
73%	From 75% to 89%	0
66%	From 85% to 91%	From 0 to 2
66%	From 68% to 85%	0
60%	From 63% to 89%	0

5.3. Visualization of Monitored Data

This section demonstrates the energy monitoring and data visualization features of the EMS in the two scenarios described in Section 4. For the first scenario, a battery charging process is monitored and the monitored data are illustrated in Figure 16. Charging starts at 4:11 AM with SoC 3% and ends at 10:42 AM with SoC 100%. The battery charging process occurs in two stages. In the first stage, the current is constant and the voltage is ascending. In the second stage, the current is descending and the voltage is constant. This charging method is good for the battery due to the fact that if the voltage is constant from the beginning, the charging current will be very high and thus cause damage to the battery. According to Figure 16, the charging current is equal to C/4 where C is the ampere rate of

the battery. In the first state, the SoC increases linearly with a slope. With the same charging current (C/4), if the slope is higher than normal value (the charging time is shorter), the battery quality may be worse. Moreover, battery temperature is observed gradually increasing in the first stage, whereas it is gradually decreasing in the second stage. It is believed that battery temperature changes due to electric current and this change is proportional.

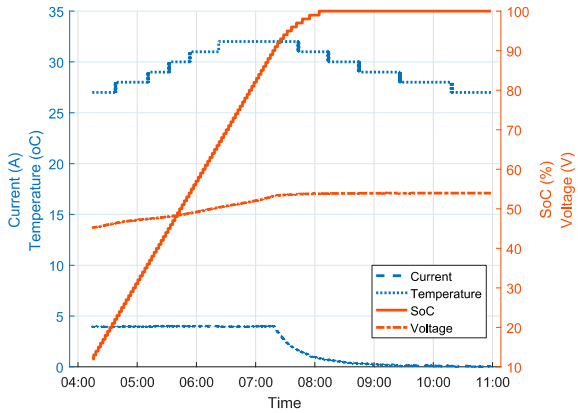


Figure 16. Battery charging process.

For the second scenario, an experience trip is monitored. The battery SoC and energy consumed by the rider during the trip are shown in Figure 17. The trip starts at 5:42 PM and ends at 8:00 p.m. The data on the elevation of the terrain are retrieved using the phone's GPS. For a higher slope, the SoC decreases faster. On a steep downhill road, the SoC increases slightly because the motor acts as a generator and recharges the battery. The rider's energy consumption increases quickly at the beginning but the gain decreases over time. This indicates that the rider's power level decreases along the trip. With this report, riders can track their energy expenditure in physical activity of cycling.

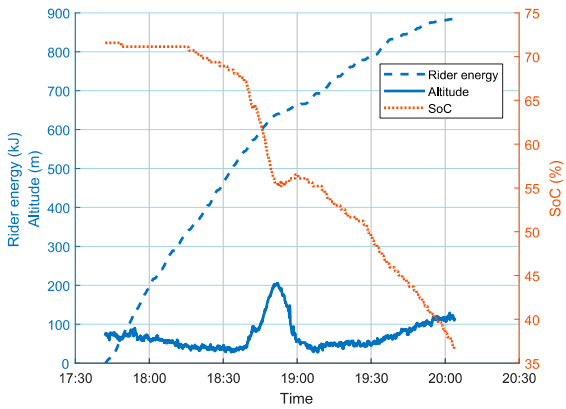


Figure 17. Battery and rider energy consumption during a trip.

6. Conclusions

In this paper, we proposed an EMS for the CANopen-based e-bikes. A gateway is designed for an EMS application installed on the rider's smart device to monitor his e-bike via the Bluetooth. The EMS application can monitor and access the CANopen devices of the e-bike to provide useful information, such as the status of the energy sources to the users. The gateway uses the CANopen protocol to communicate with the smart device. For the smart device to communicate with the CANopen devices on the e-bike, the gateway provides two main services. The first is the CAN message forwarding service that monitors, filters, and forwards the messages from the CAN bus side to the Bluetooth side. The filter is configured by a proposed RXFilter array located in the gateway's OD. This service is

modeled to calculate its parameters and ensure its performance. The second is the SDO services which are couples of the SDO server and SDO client working as relays between the SDO clients of the smart device and the SDO servers of the nodes on the e-bike. The gateway controls the throughput of its SDO services flow to keep the CAN bus load under safe value. The EMS application uses an SDO client to access the RXFilter array in the gateway to configure the filter so that it passes only PDO messages carrying energy-related data. The EMS application also uses its SDO client to access the OD of any node. The result shows that for the safety of the monitored system, the CAN bus load should not be greater than 80%. Our design ensures that the gateway can be used for a smart device to monitor any CANopen system in real-time without affecting the functionality of the monitored system. In the future, the system may be improved to connect to the cloud for further research and applications.

Author Contributions: Conceptualization, C.-R.D., V.-T.B., and Y.-C.H.; methodology, V.-T.B.; system implementation, V.-T.B. and Y.-C.H.; performing the simulations and experiments, V.-T.B. and Y.-C.H.; analyzing the simulation and experimental results, V.-T.B., C.-R.D., and P.L.; writing—original draft preparation, V.-T.B.; writing—review and editing, C.-R.D., P.L., and V.-T.B.; visualization, V.-T.B. and V.D.T.; supervision, C.-R.D. All of the authors contributed significantly to the completion of this manuscript, conceiving and designing the research, and writing and improving the paper. All authors have read and approved the manuscript.

Funding: The work was supported by the Ministry of Science and Technology of Taiwan under contract No. E10500006007-007.

Acknowledgments: The authors would like to thank the Ministry of Science and Technology of Taiwan for financially supporting this research.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Birol, F. *CO_2 Emissions from Fuel Combustion Highlights 2019*; International Energy Agency: Paris, France, November 2019; Available online: https://iea.blob.core.windows.net/assets/eb3b2e8d-28e0-47fd-a8ba-160f7ed42bc3/CO2_Emissions_from_Fuel_Combustion_2019_Highlights.pdf (accessed on 19 May 2020).
2. Benoît, B.; Chloé, M.; Julian, F. *Quantifying CO_2 Savings of Cycling*; The European Cyclists' Federation: Brussels, Belgium, November 2011; Available online: https://ecf.com/sites/ecf.com/files/ECF_CO2_WEB.pdf (accessed on 19 May 2020).
3. Tal, I.; Ciubotaru, B.; Muntean, G. Vehicular-Communications-Based Speed Advisory System for Electric Bicycles. *IEEE Trans. Veh. Technol.* **2016**, *65*, 4129–4143. [CrossRef]
4. Ferreira, J.C.; Monteiro, V.; Afonso, J.A.; Afonso, J.L. Mobile Cockpit System for Enhanced Electric Bicycle Use. *IEEE Trans. Ind. Inform.* **2015**, *11*, 1017–1027. [CrossRef]
5. Li, P.; Zhang, Z.; Xiong, Q.; Ding, B.; Hou, J.; Luo, D.; Rong, Y.; Li, S. State-of-health Estimation and Remaining Useful Life Prediction for the Lithium-ion Battery based on a Variant Long Short Term Memory Neural Network. *J. Power Sources* **2020**, *459*, 1–12. [CrossRef]
6. Torsten, G. EnergyBus: An Open Specification for LEVs. *CAN Newletter*. 2013, pp. 28–30. Available online: https://can-newsletter.org (accessed on 19 May 2020).
7. Torsten, G. EnergyBus—The CANopen-based Communication Standard for LEVs and More. Available online: http://www.energybus.org (accessed on 19 May 2020).
8. Holger, Z. CANopen in Light Electric Vehicles. In Proceedings of the 12th International CAN Conference, Barcelona, Spain, 11–13 March 2008; pp. 1–3.
9. *CANopen Application Layer and Communication Profile, CiA301*; Version 4.2.0; 2011; Available online: https://www.can-cia.org/groups/specifications (accessed on 19 May 2020).
10. *Application Profile for Energy Management Systems, CiA454*; Version 2.0.0; 2014; Available online: https://www.can-cia.org/groups/specifications (accessed on 19 May 2020).
11. *Interfacing CANopen with TCP/IP, CiA309*; Version 1.1; 2006; Available online: https://www.can-cia.org/groups/specifications (accessed on 19 May 2020).
12. Gorenflo, C.; Rios, I.; Golab, L.; Keshav, S. Usage Patterns of Electric Bicycles: An Analysis of the WeBike Project. *J. Adv. Transp.* **2017**, *2017*, 1–14. [CrossRef]
13. Phidgets Inc. Unique and Easy to Use USB Interfaces. 2017. Available online: http://www.phidgets.com (accessed on 19 May 2020).

14. Song, C.; Shao, Y.; Song, S.; Peng, S.; Xiao, F. A Novel Electric Bicycle Battery Monitoring System Based on Android Client. *J. Eng.* **2017**, *2017*, 1–11. [CrossRef]
15. Kiefer, C.; Behrendt, F. Smart E-bike Monitoring System: Real-Time Open Source and Open Hardware GPS Assistance and Sensor Data for Electrically-Assisted Bicycles. *IET Intell. Transp. Syst.* **2016**, *10*, 79–88. [CrossRef]
16. Catargiu, G.; Dulf, E.-H.; Miclea, C.L. Connected Bike-smart IoT-based Cycling Training Solution. *Sensors* **2020**, *20*, 1473. [CrossRef] [PubMed]
17. Revuelta, J.; Villarrubia, G.; Barriuso, A.L.; Hernández, D.; Lozano, Á.; de la Serna González, M.A. New Architecture for Electric Bikes Control Based on Smartphones and Wireless Sensors. In Proceedings of the Trends in Practical Applications of Scalable Multi-Agent Systems, the PAAMS Collection, Sevilla, Spain, 1–3 June 2016; pp. 125–134.
18. *Additional Application Layer Functions, CiA302*; Version 4.1.0; 2009; Available online: https://www.can-cia.org/groups/specifications (accessed on 19 May 2020).
19. *CANopen Layer Setting Services (LSS) and Protocols, CiA305*; Version 3.0.0; 2013; Available online: https://www.can-cia.org/groups/specifications (accessed on 19 May 2020).
20. *CANopen Services and Protocols for Sleep and Wake-up Handling, CiA320*; Version 1.0.0; 2018; Available online: https://www.can-cia.org/groups/specifications (accessed on 19 May 2020).
21. *CANopen Generic Frame for Wireless Tunneling of CAN Messages and for Transfer of Diagnostic Data, CiA315*; Version 1.0.0; 2011; Available online: https://www.can-cia.org/groups/specifications (accessed on 19 May 2020).
22. *CANopen Device Profile for Wireless Transmission Media Based CANopen Devices, CiA457*; Version 1.0.0; 2011; Available online: https://www.can-cia.org/groups/specifications (accessed on 19 May 2020).
23. Zitzmann, H.; Neupert, H.; Wachtel, H.; Boebel, H. *CiA 454 Infotag*; CAN in Automation: Nuremberg, Germany, 2012; Available online: http://www.energybus.org/content/download/20507/105621/file/20.06.2012-Cia_presentation.pdf (accessed on 19 May 2020).
24. Bosch, R. *CAN Specification Version 2.0*; Robert Bosch GmbH: Stuttgart, Germany, 1991; Available online: http://esd.cs.ucr.edu/webres/can20.pdf (accessed on 19 May 2020).
25. Shortle, J.F.; Thompson, J.M.; Gross, D.; Harris, C.M. Simple Markovian Queueing Models. In *Fundamentals of Queueing Theory*, 5th ed.; Wiley: Hoboken, NJ, USA, 2018; pp. 73–145.
26. Davis, R.I.; Burns, A.; Bril, R.J.; Lukkien, J.J. Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. *Real-Time Syst.* **2007**, *35*, 239–272. [CrossRef]
27. Pfeiffer, O.; Ayre, A.; Keydel, C. *Embedded Networking with CAN and CANopen*, Rivised 1st ed.; Copperhill Media Corporation: Greenfield, MA, USA, 2008.
28. MPLAB X IDE v3.4. Available online: https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive (accessed on 19 May 2020).
29. MPLAB C for PIC18 v3.47 Standard-Eval Version. Available online: https://www.microchip.com/Developmenttools/ProductDetails/SW006011#additional-summary (accessed on 19 May 2020).
30. CANopenNode-1.10. Available online: https://sourceforge.net/projects/canopennode/files/canopennode/CANopenNode-1.10/ (accessed on 19 May 2020).
31. HM 10 Bluetooth Module. Available online: https://components101.com/wireless/hm-10-bluetooth-module (accessed on 19 May 2020).