The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 20, No. 2, pp.157-163, Apr. 30, 2020. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2020.20.2.157
JIIBC 2020-2-21

IoT기반 ECO 운전보조 시스템 구현

Implementation of ECO Driving Assistance System based on IoT

송현화*, 최진구**

Hyun-Hwa Song*, Jin-ku Choi**

요 약 최근 국내에서 미세먼지로 인해 심혈관계 질환이 유발되는 것으로 알려져 운전자들은 대기오염을 일으키는 자동차의 연료를 효율적으로 사용하여 배출 가스를 감소시키는 방안에 대한 관심이 높아지고 있다. 이에 따라, 연비운전으로 연료를 절약시키고 운전자의 잘못된 운전습관을 개선하는 운전 보조 시스템을 개발했다. 개발한 시스템은 라즈베리파이, 아두이노와 안드로이드를 사용했다. OBD-II에서 얻어지는 차량의 RPM, 속도, 연료 분사량 정보들과 자이로센서의 값들을 이용하여 사용자로 하여금 Fuel-Cut을 유도하여 최적의 관성주행 환경을 유도한다. 뿐만 아니라 GUI와 음성인식 기능을 통해 날씨, 주행환경, 졸음운전 방지 등 여러 가지 인포테이먼트 시스템을 제공한다. 안드로이드 애플리케이션을 이용하여 주행 기록 및 차량 고장 정보를 확인 할 수 있으며 IoT환경에 최적화 된 MQTT 프로토콜을 사용하여 메시지 전송의 오버헤드가 적게 구현했다.

Abstract Recently, fine dust has been known to cause cardiovascular diseases here, raising interest in ways to reduce emissions by efficiently using fuel from cars that cause air pollution. Accordingly, a driving assistance system was developed to save fuel by eco-driving and improve the driver's bad driving habits. The system was developed using raspberry pi, arduino and Android. Using RPM, speed, fuel injection information obtained from OBD-II, and gyro-sensor values, Fuel-Cut is induced to create an optimal inertial driving environment. It also provides various information system such as weather, driving environment, and preventing drowsy driving through GUI and voice recognition functions. It is possible to check driving records and vehicle fault information using Android application and has low overhead for message transmission using MQTT protocol optimized for IoT environment.

Key Words : Arduino, Android, IoT, MQTT, OBD-II, Raspberry Pi

Ⅰ. 서 론

최근 국내에서는 미세먼지로 인해 심혈관계 질환 등이 유발되는 것으로 알려져 관심과 우려가 높아지고, 미세먼지를 체감하는 불편함이 점점커지는 실정이다. 미세먼지 농도는 2012년 이후 정체하거나 오히려 증가하고 있으며, 미세먼지 고농도 일수 역시 점차 증가하는 추세이다.[1]

*비회원, 한국산업기술대학교 컴퓨터공학과 석사과정
**정회원, 한국산업기술대학교 컴퓨터공학부
접수일자: 2019년 11월 6일, 수정완료: 2020년 3월 2일
게재확정일자: 2020년 4월 3일

Received: 6 November, 2019/ Revised: 2 March, 2020
Accepted: 3 April, 2020
*Corresponding Author: jkchey@kpu.ac.kr
School of Computer Engineering, Korea Polytechnic University, Korea.

미세먼지를 배출하는 원인으로 수도권에서는 경유차(29%)가 가장 높은 비중을 차지하고 있으며, 경유차 중에서는 RV(경유), 화물차(경유), 승합차(경유) 순으로 비중이 높다.[2] 따라서 미세먼지 및 차량 배출가스 저감을 위해서 정부 및 기관에서 관련한 정책들을 추진하고 있으나 별다른 실적이 없는 상황이다. 따라서 최근 많은 대기오염을 일으키는 자동차의 연료를 효율적으로 사용하여 배출 가스를 감소시키는 방법에 대한 현실적인 대안을 필요로 하고 있다.

이에 따라, 본 논문에서는 연료를 효율적으로 쓰는 방법 즉, 연비운전을 통해 연료를 절약시키고, 운전자의 잘못된 습관을 개선하는 운전 보조 시스템을 개발했다.

우선 연비운전하기 위해서는 Fuel-Cut이 필요하다. Fuel-Cut이란 말 그대로 엔진에 공급되는 연료를 차단하는 것인데 내리막길이나 평지와 같이 바퀴에 동력이 전달되지 않아도 관성과 중력가속도에 의해 차량이 정속으로 운행 가능한 상태일 때 가속페달에서 발을 떼기만 하면 ECU에서 이를 자체적으로 판단하여 엔진으로 가는 연료를 차단하여 절감 시킨다.[3]

운전자의 연비운전과 운전 습관을 개선하기 위해 OBD-II에 연결하여 차량의 RPM, 속도, 연료 분사량 등의 정보를 얻어 연비를 계산하였고, 운행도로의 경사도를 감지하여 운전자에게 Fuel-Cut을 지시했다. 뿐만 아니라 차량의 고장정보를 읽어와 차량 화재와 같은 중대한 차량 결함을 미연에 방지 할 수 있다.

OBD-II에서 얻어진 데이터를 가공하여 운전자를 위해 화면에 정보들을 표시하였으며, 무선 네트워크를 통해서 차량 인포테이먼트 기능 등을 제공했다.

본 논문에서 구현한 시스템은 실시간으로 주행 정보에 대해 가시적으로 확인할 수 있으므로 연비운전에 대한 집중도를 높여 더 나은 연비개선에 대한 결과를 기대할 수 있다. 뿐만 아니라 가공된 데이터를 안드로이드 애플리케이션을 통해 언제 어디서나 손쉽게 볼 수 있고 차량 관리에 도움을 준다.

II. 관련 연구

1. 기존 유사 앱과의 비교

기존의 유사 애플리케이션과 비교는 표 1에서 보여주고, 마카롱의 경우에는 기본적인 차계부 기능을 지원하지만 그 정보를 수동으로 입력해야 하고, 차량 진단과 주행에 관련된 기능들이 상당수 부족하다.

표 1. 유사 애플리케이션과의 비교
Table 1. Comparison with similar application

App / 기능	마카롱	Torque	T-map
자동주행기록	X	X	O
연비운전보조	X	X	△
고장진단	X	O	X
주행분석	X	X	O
음성인식	X	X	O
실내공기 모니터링	X	X	X
기타	차계부	차량진단	내비게이션

또한, Toque의 경우에는 차량 진단을 위해 개발된 애플리케이션으로 차량에 지식이 없는 일반인이 사용하기 어렵고 편의성을 향상시키는 기능들이 부족하다. T-map의 경우에는 내비게이션 기능을 지원하는 애플리케이션으로 비교된 애플리케이션 중 가장 많은 기능을 지원한다. 하지만 차량 관리에 필요한 고장 진단 기능과 연비운전을 돕는 기능이 다소 부족한 편이다.

따라서 본 논문에서는 표 1에서 각 애플리케이션의 부족한 기능들을 하나의 시스템으로 구현하여 보완된 운전 보조 시스템을 구현했다.

III. 설계 및 구현

1. 개발환경

표 2. 개발환경
Table 2. Development environment

구분		상세내용
S/W 개발환경	OS	Raspbian(Linux), Windows10
	IDE	PyCharm, Android Studio
	개발도구	Sublime Text3, Nano
	개발언어	Python3, Java
	기타사항	Web Server와 Json방식으로 통신
H/W 구성	디바이스	Raspberry Pi 3(LCD 포함), ESP8266
	센서	NEO-6M, MPU6050, DHT11, MQ-2
	통신	E8372(LTE Module), ELM327(OBD2), ESP8266(WiFi)
	언어	Python3, C++
	기타사항	Sensor와의 통신에는 MQTT Protocol 사용

2. 시스템 구성

본 논문에서 구현하려는 시스템 구성은 아래 그림 1에 나타내었다. 기본적인 동작방식은 차량의 OBD-II와 블루투스 통신을 인터페이스하는 모듈(ELM327)과 각종 센서들로(자이로, 이산화탄소, 온습도, GPS)부터 수집된 데이터들 WiFi모듈(ESP8266)을 통해서 라즈베리파이로 전송한다. 이 전송에서는 MQTT 프로토콜을 이용한다.[4] 라즈베리파이에서는 전송 받은 데이터들을 가공한 후 GUI로 출력하고 수집된 데이터를 실시간으로 클라우드 서버로 전송한다. 서버로 전송된 데이터는 안드로이드 애플리케이션을 이용하여 확인 할 수 있다.

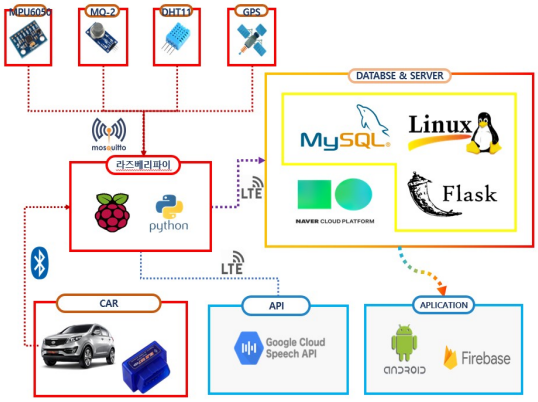

그림 1. 시스템 구성도
Fig. 1. System Configuration

3. 기능 구현

가. OBD2 데이터 가공 및 관성주행 가능 조건 판단

차량은 급한 경사로를 만나거나 특정 RPM과 속도를 만족했을 때 가속페달에서 발을 떼면 Fuel-Cut 기능이 동작하도록 설계되어 있다. 이를 이용하여 Fuel-Cut 기능이 동작할 수 있는 환경을 사전에 예측하여 사용자에게 제공 할 수 있다. 동작 조건은 RPM≥2000, 차량 속력≥40km/h, 경사로 각도≥20°이며 기아자동차 니로, 현대자동차의 아이오닉의 관성주행 보조시스템을 참조했다.[5]

```
def _on_update_eco_das(self, r):  # 관성주행 보조 기능
    current_eco_das = True if r > 20 and self.speed > 40 and
self.rpm > 2000 else False

    if self.is_eco_das is not current_eco_das:
        self.on_changed_eco_das.emit(current_eco_das)
        self.is_eco_das = current_eco_das
```

그림 2. 관성주행 조건 판단
Fig. 2. Determine Inertia Driving Conditions

그림 2에서 차량 연식에 따라서 MAF 값이 제공되지 않는 차량이 존재하므로 MAP 값을 이용하여 MAF값을 계산했다. 또한 MAF 값을 이용하여 이론 공연비와 체적 효율을 이용하여 순간 연비를 산출했다.

```
def _get_maf(sel_f):          # MAF 개선
    maf = 28.97 * ( self.volume * 0.85 * ((self.rpm * self.map
/ (self.iat + 273.15)) / 120)) / 8.314
    if maf == 0:
        maf = 0.01
    return maf

def _update_instance_fuel_efy(self):
    self.ife = (14.7 * 6.17 * 454 * 0.621371 * self.speed *
0.425144) / (3600 * self._get_maf())
    if self.is_fct:
        self.on_changed_ife.emit(99.9)
    else:
        self.on_changed_ife.emit(self.ife)

# 1초에 1번만 호출되는 함수들
def _update_fuel_use(self):
# 기름 소모량 계산(1초에 1번 호출)
    if self.eng_stat is not True:
        return

    if not self.is_fct:
        self.fuel_use = self.fuel_use + self._get_maf() / (14.7 *
0.73 * 1000)
        self.on_changed_fuel_use.emit(self.fuel_use)

    if self.fuel_use != 0:    #평균 연비 계산
        avr_fuel = self.distance / self.fuel_use
        self.on_changed_avr_fuel.emit(avr_fuel)
```

그림 3. MAP값을 이용한 MAF값 및 순간 연비 계산
Fig. 3. MAF value and instantaneous fuel economy calculation using MAP values

상기 그림 3에서 callback으로 등록된 함수의 파라미터(r)는 자이로센서(MPU-6050)에서 측정된 기울기 값으로 RPM과 속도, 기울기 값 조건을 판단하여 관성주행 여부를 current_eco_das 함수에 제공한다.

나. MQTT 프로토콜을 이용한 사물간 통신

MQTT 프로토콜은 패킷의 크기가 매우 작아 성능이 제한적인 장치와 IoT환경에서 주목받고 있는 프로토콜이다. WiFi 모듈(ESP8266)과 자이로 센서, CO2센서, 온습도센서, GPS센서 등을 확장성을 고려하여 각각 연결했다. WiFi 모듈의 설정모드에서는 WiFi-AP모드로 동작하며 HTTP 프로토콜을 통해 차량의 ID, 연결할 WiFi의 SSID, Password 정보 등을 입력하고 입력이 끝나면 EEPROM에 데이터를 저장하고 자동으로 재부팅되면서 MQTT 프로토콜을 이용하여 서버에 센싱된 데이터를 Publish한다. 라즈베리파이에서는 해당 Topic에 Subscribe하고 실시간으로 데이터를 수신한다.

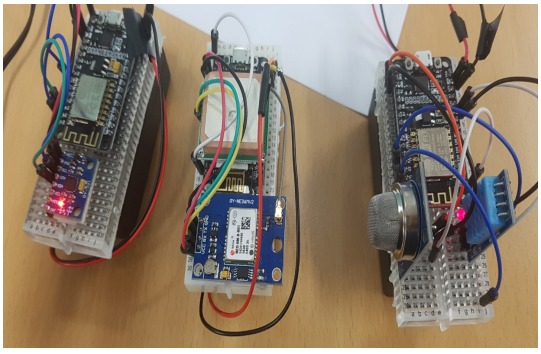

그림 4. WiFi 모듈(ESP8266)과 센서 인터페이스 구현
Fig. 4. Implementation interface ESP8266 and Sensors

```
EEPROM.begin(EEPROM_LENGTH);
pinMode(5, INPUT_PULLUP);
attachInterrupt(5, GPIOS, FALLING);
ReadString(0, 30);
if(!strcmp(eRead, "")) {
    setup_captive();
} else {
    captive = false;
    strcpy(ssid, eRead);
    ReadString(30, 30);
    strcpy(password, eRead);
    ReadString(60, 30);
    strcpy(id, eRead);
    // make topic
    strcat(topic, id);
    strcat(topic, "/gps");
    Serial.println(topic);
    //--------------------
    client,setServer(mqttServer, mqttPort);
    while(!clien.connected()) {
        Serial.println("Connecting to MQTT...");
        if (!client.connected("Chan_GPS", mqttUser,
mqttPassword)) {
            Serial.println("connected");
            client.publish("gps/status", topic);
        } else {
            Serial.print("failed with state");
            Serial.println(client, state());
            delay(2000);
        }
    }
}
```

그림 5. EEPROM에서 데이터를 읽어 AP에 연결
Fig. 5. Read data from EEPROM to connect to AP

다. PyQt를 이용한 GUI 환경

PyQt는 Python 언어 환경에서 GUI 프로그래밍을 할 수 있게 도와주는 도구로써 Windows, Linux 등 다양한 동작 환경을 제공한다. 라즈베리파이의 7‘ Touch LCD를 통해 GUI 환경을 제공하고 차량의 주행 정보(연비, 속도, RPM, 기름 사용량)와 날씨, 세차지수 등의 인포테이먼트를 제공한다.

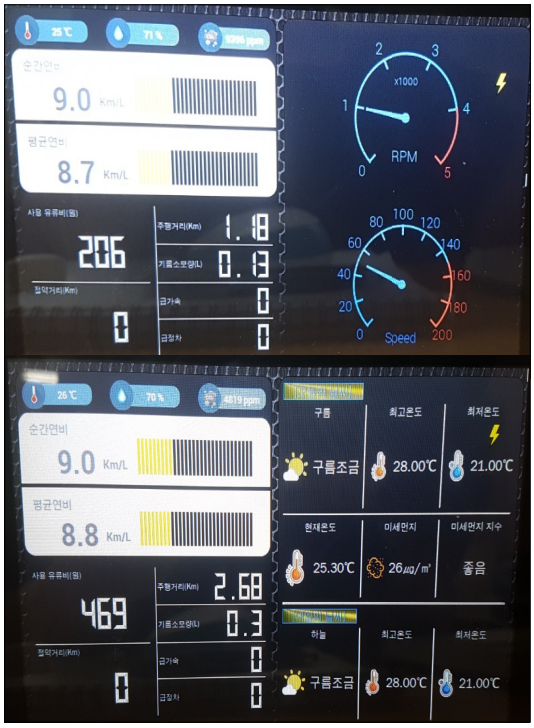

그림 6. PyQt5를 이용한 GUI 환경
Fig. 6. GUI Environment Using PyQt5

라. 음성인식 환경

사용자는 운전에 집중해야하기 때문에 운행 도중 해당 기능을 사용하는데 있어 제약이 따른다.[6] 따라서 간단한 음성 명령을 이용하여 여러 가지 기능을 제어할 수 있도록 했다. 동승자와의 대화 및 잡음으로 인해 기능이 오작동 할 가능성이 있기 때문에 snowboy 라이브러리를 이용해 특정 호출 커맨드에만 반응하도록 개발했다. 지정된 이름을 부르게 되면 등록된 callback함수에 의해 본격적인 음성 인식이 시작되는데 Google의 Cloud Speech 라이브러리를 사용했다.

기능을 제어하는 것뿐만 아니라 정보를 얻을 때에도 운전자가 화면에 시선을 두는 것이 위험하므로 쉽게 정보들을 얻을 수 있다.

아래 그림 7에서 STT 객체는 음성 인식에 관한 동작을 담당하는 객체로써, get_str 함수를 호출하면 인식된 문자열이 반환된다. 매개변수로 callback을 등록하면 등록된 함수의 인자로 인식된 문자열이 반환된다.

snowboy를 생성할 때 사전에 제작된 음성 커맨드 파일(doraemon.pmdl)과 민감도를 설정할 수 있다. start 함수를 호출할 때 음성 인식 시 호출할 함수를 등록하면 이름을 불렀을 때 등록된 함수(self.gg)가 호출된다.

```
        self.tts = TTS()
        self.stt = STT()
        self.speaker = 'mijin' #TTS 목소리 생성
        # 음성 커멘드 설정
        self.detector =
snowboydecoder.HotwordDetector('snowboy/resources/dorae
mon.pmdl', sensitivity=0.5)
        # 음성 인식은 별도의 쓰레드에서 동작
        speech_thread =
threading.Thread(target=self.spechRecogStart)
        speech_thread.daemon = True
        speech_thread.start()

def speechRecogStart(self):
    self.detector.start(detected_collback=self.gg,
sleep_time=0.03)

def gg(self):
        self.detector.terminate()
snowboydecoder.play_audio_file(snowboydecoder.DERECT_D
ING)
        if self.currentwidget != None:
            self.label_stt.setText("...")
            self.label_tts.setText("...")
            self.currentwidget.hide()
        self.label_stt.show()
        self.label_tts.show()
        text = self.stt.get_str(self.label_stt.setText)

snowboydecoder.play_audio_file(snowboydecoder.DETECT_D
ONG)
        self.label_stt.setText(text)
```

그림 7. TTS, STT 객체 생성 및 calback 등록
Fig. 7. Create TTS, STT objects and register calback

4. 안드로이드 앱 구현

라즈베리파이의 Touch LCD에 출력되는 데이터는 현재 주행에 대한 실시간 정보 제공에 초점이 맞춰져있는 한편 안드로이드는 클라우드 서버의 DB에 저장된 데이터를 운전자에게 제공하기 위해 활용된다. 애플리케이션을 최초 실행할 때 로그인 정보로 활용하는 Google 계정과 고유한 디바이스 ID를 사용해 DB로부터 사용자 본인의 데이터를 정확하게 불러올 수 있다.

가. 주행 기록 및 주행 습관 분석

사용자가 찾고자 하는 주행 기록의 범위를 줄이고 DB에 과부하를 예방하기 위해 사용자가 기간을 설정해 주행 기록을 검색할 수 있게 했다. 특정 주행 기록을 선택하면 그림 8과 같이 GPS 데이터를 기반으로 한 주행 경로, 주행 시간, 주행 점수, 평균 주행 연비, 주행 거리, 평균 주행 속력 및 주행 점수의 감점 요인과 같은 기본적인 정보와 더불어 경제운전 정도와 공회전의 비율을 원형 그래프로 나타낸 주행 성향 그래프, 주행 시간에 따른 속력 변화 그래프를 볼 수 있도록 개발했다.

그림 8. 주행 기록(좌), 주행 성향 및 속력 그래프(우)
Fig. 8. Drive History(Left), Driving propensity Graphs(Right)

나. 고장 정보 확인

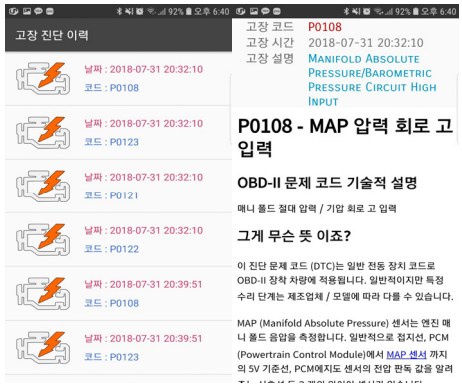

그림 9. 고장 진단 기록(좌)과 상세 고장 정보(우)
Fig. 9. Diagnostic fault history(Left) and fault information (Right)

다. Firebase를 이용한 GCM과 Map, Login 기능

Firebase는 인증, 호스팅, DB 등과 관련된 API를 하나의 SDK에 직관적으로 포함시킨 통합 크로스 플랫폼 SDK이다.

그림 10과 같이 본 애플리케이션에서는 사용자 등록, 인증을 위한 Login 기능 구현과 차량 위치, 주행 경로를 지도에 나타내기 위해 Firebase를 사용해 개발했다. 또한 사용자가 주행을 종료하면 주행에 대한 정보가 DB에 저장됨과 동시에 사용자의 스마트폰으로 전송되도록 구현하는 데에도 Firebase를 사용했다.

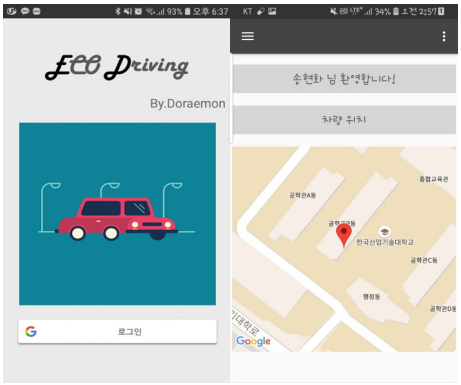

그림 10. Firebase로 구현한 Login 기능(좌)과 Google Map(우)
Fig. 10. Login Implemented as Firebase(Left), Google Map(Right)

라. 주행종료 알람 기능

```
private void sendNotification(String messageBody){
        Intent intent = new Intent(this, NotifyActivity.class);
// 눌렀을 때 실행할 액티비티
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        intent.putExtra("message", messageBody); //여기에
넘겨줄 데이터를 추가
          Log.d("받은 메세지" , messageBody);
        NotificationManager mNotificationManager =
(NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
        String id = "my_channel_01";
        CharSequence name = "test";
        int importance =
NotificationManager.IMPORTANCE_HIGH;
        NotificationChannel mChannel = new
NotificationChannel(id, name, importance);
        mChannel.enableLights(true);
        mChannel.setLightColor(Color.RED);

mChannel.setSound(RingtoneManager.getDefaultUri(Ringtone
Manager.TYPE_NOTIFICATION), null);
        mChannel.enableVibration(true);
        mChannel.setVibrationPattern(new long[]{100, 200,
300, 400, 500, 400, 300, 200, 400});
        mChannel.setShowBadge(false);
mNotificationManager.createNotificationChannel(mChannel);
        int notifyID = 1;
        String CHANNEL_ID = "my_channel_01";
        try {
            JSONObject jsonMeg = new
JSONObject(messageBody);
            Notification notification = new
Notification.Builder(this)
              .setContentTitle("주행 종료")
              .setContentText(String.format("주행연비 : %.2f
Km/L     주행거리 : %.2f Km",
jsonMeg.getDouble("fuel_efi"),jsonMeg.getDouble("distance")))
              .setSmallIcon(R.drawable.img_car)
              .setChannelId(CHANNEL_ID)
              .build();
          mNotificationManager.notify(notifyID, notification);
        } catch (JSONException e) {
            e.printStackTrace();
        }
```

그림 11. 주행 종료 시 스마트폰으로 Push 메시지 전송
Fig. 11. Push message sent to Phone at end of drive

본 애플리케이션에서는 사용자가 주행을 종료할 시 스마트폰으로 Push 메시지로 주행정보(주행 시 사용된 연비, 주행 거리 등)를 전달하도록 개발했다.

Ⅳ. 구현 결과 및 시험

라즈베리파이와 서버, 각 센서와 라즈베리파이 등의 통신, OBD-II와 센서들의 데이터 파싱, 가공 및 DB 전송, 조회와 같은 핵심 기능은 모두 구현되었다.

OBD-II 모듈에서 나오는 데이터(평균 속력, 연비, 주행시간, 주행거리, 주행점수, 급가속/급정차 횟수)를 파싱하여 데이터베이스에 저장했다. 또한, GPS 모듈을 사용하여 매 초마다 위도, 경도를 이용하여 현재 차량에 대한 위치 정보를 저장했다. 이 데이터들을 사용하여 관성주행 정보를 포함한 주행정보(RPM, 속도, 연비 등)를 LCD에 표시했다. 또한, 음성인식 기능을 통해 LCD에 시각적으로 표현되어있는 정보들을 청각으로 확인할 수 있다.

Eco-DAS 모듈을 사용하여 Fuel-Cut이 작동 가능한 상황(자이로 센서를 통해 기울기를 획득)에서 운전자에게 알림이 가도록 했다. 기울기가 진행방향으로 10도 이상 기운 상태로 3초 이상 유지되었을 때를 True, Fuel-Cut 상태이거나 위 조건을 만족하지 않을 경우 False로 정했다.

주행 점수를 계산해주는 상용화된 서비스들은 주행 시간이 길어짐에 따라 점수가 유리하게 판정되는 경향이 있다. 본 시스템에서는 주행 시간이 길어지는 만큼 감점 요인에 어드밴티지 비율을 다르게 설정해서 일관적인 주행 점수가 나오도록 설계했으나 신뢰도를 높이기 위해 많은 중/장거리 주행 데이터를 분석해 점수 계산 방식이나 감점 정도의 조정이 필요할 수 있다.

애플리케이션에서는 로그인과 차량 등록을 통해 최근 주행 기록(경로, 경제운전 비율, 차량이 주차된 위치)에 대해 확인할 수 있다. 또한, 주행 속력과 연비, 급가속/급정차를 분석한 주행 점수를 통해 자신의 운전에 대한 평가를 받을 수 있으며, 경제운전 정도에 따른 비율을 그래프를 통해 확인할 수 있다. 고장 진단 기능이 존재하며 고장내역에 대한 코드를 제공하여 코드 별 고장 시간과 증상에 대한 설명을 볼 수 있다.

이 시스템은 대부분 프로그래밍 언어로 Python3가 사용되었다. 따라서 오픈소스나 API를 사용하기에 유용하다. 이미 날씨API나 STT/TTS와 같은 API가 사용되었고 웹 크롤링을 통해 세차지수를 알려주는 기능을 포함

하고 있다. 개발이 끝난 이후에도 다른 API를 활용하거나 웹 기반 데이터를 바탕으로 제공하는 기능을 쉽게 추가할 수 있다.

각 센서들은 독자적인 회로를 가지고 라즈베리파이와 무선으로 통신하므로 설치가 간편하고 다른 센서를 간단하게 추가할 수 있어 확장성이 있다.

V. 결 론

운전습관 개선으로 효율적으로 연비를 사용하기 위해 주행정보가 필요함에 따라 OBD-II를 통해서 정보를 가져오고, 내리막길(기울기)을 감지하기 위해 자이로센서를 사용하여 실시간으로 차량의 주행정보(속도, RPM, 연비, 관성주행 여부, 급가속/급정차 횟수, 절약 거리, 사용 유류비)를 확인할 수 있도록 구현했다. 또한, 음성인식 환경을 통해 날씨, 세차정보, 주행정보 등 여러 가지 인포테이먼트 시스템을 제공받으며 시각, 청각으로 동시에 서비스를 제공하였으며, 차량의 고장 및 진단 모니터링을 통해 차량 화재와 같은 중대한 차량 결함을 미연에 방지 할 수 있도록 했다.

안드로이드 애플리케이션을 통해 언제 어디서나 손쉽게 차량 정보를 볼 수 있으며, 여러 센서를 통해 얻어진 가공된 데이터를 분석하여 주행점수를 매김으로써 자신의 주행습관에 대해 개선 할 수 있도록 했다. 또한, 최근 차량의 위치를 파악하거나 주행 경로를 확인할 수 있고, 총 주행거리로 유류 소모량을 계산하여 월별 유류비 지출액을 계산할 수 있도록 했다. 이러한 정보들을 이용해 차량을 효율적으로 관리할 수 있도록 도움을 주었다.

구현한 시스템은 한 가지 시스템으로 차량에 관련된 여러 가지 기능을 제공함으로써 사용자들이 더욱 편리하게 사용할 수 있다는 장점이 있다. 차량의 AVN(Audio Video Navigation system)에 탑재됨으로써 차량 제조사의 독자적 기능 제공 및 차량 인포테이먼트 시스템의 발전 가능성을 기대할 수 있다.

References

[1] Ministry of Environment, “You can see it if you know it right away. Fine dust, what the hell is it?”, A booklet of the Ministry of Environment, pp. 5-26, April 2016.

[2] In-Chang Hwang, Jin-Seok Han, “A Feasibility Study of a New Urban Access Regulation in Seoul : Policy Design, Public Acceptance, and the Expected Effects”, The Seoul Institute, pp. 15-18, Sept 2018.

[3] Chae-Hwi Kim, “Modern Automotive Engineering Series5”, Golden Bell, April 2011.

[4] Sung-Woo Kim, “Raspberry Pi Embracing the Internet of Things”, Jeipub, pp. 563-589, Jan 2016

[5] Hyundai Motors, “Ioniq Electric, the fuel economy king run 351 kilometers on a single charge”, HMG Journal, Oct 2016.

[6] You-Sik Hong, Myeong-Bok Choi, “Research on Safe Application Program of Smart Phone for Auto Receiving and Answering during a Car Driving”, The Journal of The Institute of Internet, Broadcasting and Communication, Vol. 15, No. 6, pp. 43-49, Dec 2015.
DOI: http://dx.doi.org/10.7236/JIIBC.2015.15.6.43

저 자 소 개

송 현 화(비회원)

• Hyun-Hwa Song is currently in a master's course in computer engineering at Korea Polytechnic University in 2020. She is studying with an interest in embedded systems and Android programming.

최 진 구(정회원)

• Jin-Ku Choi received the Dr. Degree in electrical engineering from Waseda University, Japan in 2003. He is currently a Professor at the Department of Computer Engineering of Korea Polytechnic University. His research interests include Embedded system and design IoT platform, etc.