

# IoT 시스템 설계 Final Project

전동 킥보드 제어 Controlling & Monitoring System

2021. 12. 08.

김성수 (21600086)  
이고은 (21800500)

# 목차

- 서론.....	3
- 설계 목적 및 주제 설명 .....	3
- 시나리오.....	3
- 제약조건.....	3
- 환경구축.....	4
- 본론.....	6
- 네트워크 설계.....	6
- 핵심 기능 구현.....	9
- 온도 센서 및 현재 킥보드 상태 표시.....	9
- 방향 지시등 및 비상등.....	11
- 조도 센서 및 헤드라이트.....	17
- 속도조절.....	25
- 현재시간 출력.....	30
- CAN Connection Indicator.....	31
- 결론.....	33
- 실험 결과 및 동작 방법 .....	33
- 느낀점 & 개선점 .....	33
- 참고.....	34
- 역할분담 .....	34
- 참고자료 .....	34

# 1. 서론

## a. 설계 목적 및 주제 설명

최근 전동킥보드를 교통수단으로 사용하는 사람들이 늘어났다. 공용 전동킥보드의 경우 추가적인 기능이 거의 없어 이를 사용하는데 불편함이 존재한다. 헤드라이트, 주행등과 같은 기본적인 기능만 존재하여 인도로 가는 것도 차도로 가는 것도 위험한 실정이다. 이를 조금이나마 보완하기 위해 전동 킥보드를 타는 사람들을 좀 더 보호할 수 있도록 비상등이나 방향지시등, 오토라이트를 제공하는 시스템을 설계하였다.

## b. 시나리오

Expansion Board에 있는 LED, 각종 센서, RTC 등을 전동킥보드를 가정하고 전동 킥보드를 제어하는 계기판과 이 전동 킥보드를 원격으로 모니터링 할 수 있는 시스템을 만든다.

## c. 제약조건

1 개의 Raspberry Pi 4B와 1 개의 Expansion Board를 하나의 세트로 구성한다. 해당 구성을 동일하게 가진 2개의 세트 사이에 데이터 통신은 CAN 통신으로 한다. 킥보드를 제어하는 컨트롤러는 GPIO와 I2C를 통하여 Expansion Board를 제어하게 된다.

#### d. 환경구축

본 프로그램을 실행하기 위해서는 2 개의 Raspberry Pi 4B와 1 개의 Extension Board가 필요하다. 아래와 같은 순서를 통하여 각 라즈베리파이를 작동 시키고 CAN 통신 세팅을 진행하면 된다. 라즈베리파이 한 개는 **Controller**, 나머지 한 개는 **Monitoring**을 위한 디바이스로 사용된다.

1) 라즈베리파이 OS가 탑재되어 있는 USB를 보드에 연결하고 2개의 Extension Board를 CAN cable를 이용하여 연결한 후 전원을 켠다.

- a) Expansion 보드에 있는 5V와 3.3V 전원 스위치를 on 상태로 바꾼다
  - i) 5V는 라즈베리파이에 전원을 공급하게 되고 3.3V는 expansion board에 전원을 공급하게 된다.
- b) 세트 구성 중 하나인 LCD display를 사용하기 위하여 LCD 전원 스위치를 ON 상태로 바꾼다. 다만, 본 프로그램은 LCD Display에서 출력하기에 화면 사이즈가 크기 때문에 가능하다면 모니터 사용을 추천한다. LCD Display로 화면을 볼 경우 실행은 가능하지만 전체 화면을 볼 수 없다.

2) 각 라즈베리파이가 정상적으로 부팅이 완료되면 아래 체크 사항을 차례로 정상 작동하는지 확인한다.

- a) 소스파일이 있는 호스트 컴퓨터와 같은 네트워크 상에 연결되어 있는지 확인한다.
- b) 터미널을 열고 'ifconfig' 명령어를 통하여 IP주소를 확인한다.
- c) QT creator가 정상적으로 설치되어 있고 작동하는지 확인한다.

3) 2개의 라즈베리파이 간 CAN 통신을 세팅을 하고 Test한다

2개 모두 can통신의 bitrate를 125000으로 설정하는 것을 기준으로 한다.

- a) 터미널에서 'ifconfig can0' 명령어를 이용하여 can0가 이미 활성화가 되어 있는지 확인한다.
- b) 만약 can0이 활성화 되어 있지 않다면 아래 순서대로 설정하면된다.
  - i) CAN 활성화

**\$ sudo ip link set can0 up type can bitrate 125000**

- ii) ifconfig 를 통해서 can 활성화 확인
- c) CAN 통신 Test 방법

2개의 라즈베리파이를 각 A, B로 명칭을 붙여 아래 내용을 설명하겠다.

- i) A은 CAN 통신에서 receiver 역할을 담당한다. 아래의 명령어를 이용하여 receiver로 활성화 시킨다

**\$ candump can0**

- ii) B는 CAN 통신에서 sender 역할을 담당한다. 아래의 명령어를 통해 패킷을 보낸다.

**\$ cansend can0 001#1122334455667788**

- iii) 패킷을 보낸 후 A의 터미널에 다음과 같이 정상적으로 메세지가 출력이 되면 정상적으로 CAN 통신이 작동하는 것을 알 수 있다.

```
pi@raspberrypi:~/myQt $ candump can0
can0 001 [8] 11 22 33 44 55 66 77 88
```

- iv) 만약 CAN 케이블을 이미 up시킨 상태로 연결한다면 제대로 통신이 작동하지 않을 것이다. 이와 같은 경우

**\$ sudo ip link set can0 down**

명령어로 down 시킨 후 다시 1번부터 진행한다.

- 4) 2개의 라즈베리파이에 모니터링과 컨트롤러 프로그램을 Qt Creator를 활용하여 실행시킨다.

## 2. 본론

### a. 네트워크 설계

두 개의 라즈베리파이를 CAN 프로토콜을 사용하여 통신하였다. CAN 통신은 차량 내에서 호스트 컴퓨터 없이 마이크로 컨트롤러나 장치들이 서로 통신하기 위해 설계된 표준 통신 규격이다. CAN 패킷은 SOF, 중재필드, 제어필드 등 다양한 필드로 구성되어있으나 Qt코딩에서는 모두 소켓통신으로 추상화되어 데이터 필드의 값만 고려하면 통신이 가능했다.

CAN 프로토콜로 컨트롤러 기기와 모니터링 기기의 통신이 필요했는데, 킥보드의 모든 LED값이나 센서값 등 실제 기기(컨트롤러의 **expansion board**)에서 나타나는 모든 정보가 동기화될 필요가 있었다. 특정 시간마다 모든 정보를 보내는 것이 가장 간단했지만 패킷의 갯수가 많아질수록 딜레이가 점점 증첩되어 통신 속도가 느려지고 컨트롤러에서 보낸 정보가 1분넘게 딜레이가 발생한 후 모니터링 결과에 나타나는 문제를 발견하여 이를 해결하기 위해 패킷 구조를 효율적으로 구성하였다.

CAN 패킷으로 보내주어야 할 정보는 RTC(6byte), 방향 지시등 및 비상등(1byte), 차량 상태표시등(1byte), 전조등(1byte), 악셀/브레이크/타이머(1byte), 조도센서(1byte), 온도센서(2byte)였다. 이 중에서 특히 RTC는 현재 시간으로 매초 업데이트 되어야하는 값이다. 1초마다 패킷을 전송할 경우 패킷의 갯수가 너무 많아지므로 킥보드에 시동을 켜면 현재 시간을 모니터링 기기에 전송해주면 내부에서 1초마다 시계를 업데이트하는 방식을 찾아냈었다. 조도센서나 온도센서의 경우 계속적으로 바뀌는 값이므로 바뀔 때마다 패킷을 전송할 경우 너무 자주 패킷의 전송이 이루어지므로 일정 시간(2초)마다 패킷을 전송하도록 설계하였고, RTC와 센서들을 제외한 나머지 정보의 경우 실시간으로 변화하지 않고 사용자가 변화를

주었을 때만 정보를 업데이트하면 되기 때문에 일정 시간마다 패킷을 전송하는 것보다 변화가 있을 때에만 패킷을 전송하는 것이 더 효율적이었다.

패킷에서 첫번째 데이터 필드는 패킷종류를 나타내는데 사용한다.

패킷종류	데이터 내용	통신 빈도
0	RTC	시동걸 때
1	방향 지시등 & 비상등 현재속도 킥보드 상태표시등 전조등 악셀/브레이크	값의 변화가 있을 때
2	조도센서 온도센서	2초마다

#### • packet type 0

value	D0	D1	D2	D3	D4	D5	D6
RTC	0	year	month	day	hour	min	sec

- RTC i2c값에서 읽어들인 값을 그대로 보낸다.
- packet type 0에서 D7는 현재상으로는 사용하지 않는다.

#### • packet type 1

value	D0	D1	D2	D3	D4	D5
방향 지시등 & 비상등	1	00 off 01 left 02 right 03 비상등				
현재속도		현재속도값				

상태표시등			00 화재 01 시동꺼짐 02 시동켜짐 03 결빙		
전조등			00 수동off 01 수동on 02 자동off 03 자동on		
악셀/브레이크				00 off 01-04 브레이크 (1-4초 타이머) 05-08 악셀 (1-4초 타이머)	

- 타이머 정보는 악셀/브레이크에 포함시켜 정보를 전달한다.
- packet type 1에서 D6, D7는 현재상으로는 사용하지 않는다.

- **packet type 2**

value	D0	D1	D2	D3
조도센서	2	조도센서		
온도센서			온도센서	

- packet type 2에서 D4,D5, D6, D7은 현재상으로는 사용하지 않는다.

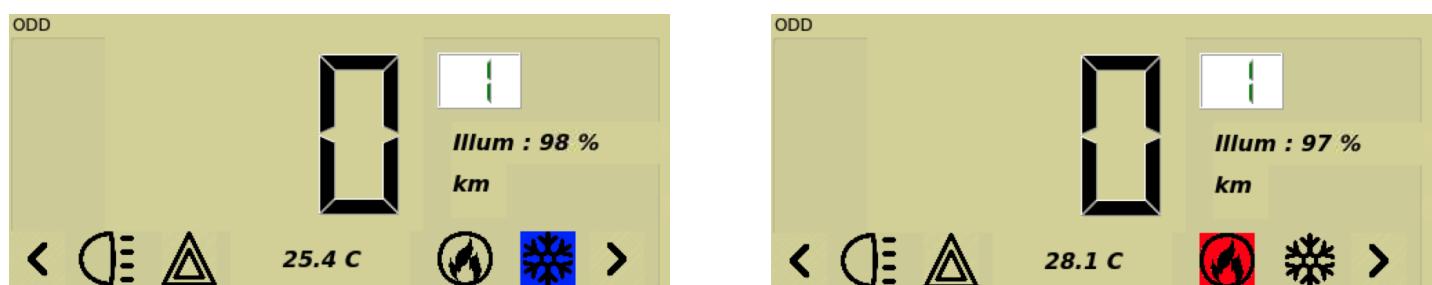
## b. 핵심 기능 구현

(Function flowchart, 하드웨어 및 소프트웨어 설계 설명,  
필요하다면 소스코드도)

프로그램이 실행이 되면 UI와 하드웨어를 초기화하는 과정이 가장 먼저 실행된다. I2C LED과 GPIO LED를 초기화(Off 상태) 시켜주고 킥보드의 상태는 시동이 꺼진 상태로 초기화 되게 된다. Light와 Turn Signal과 Emergency Signal은 모두 Off 상태로 초기화된다.

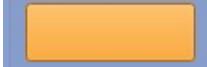
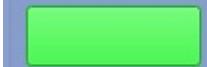
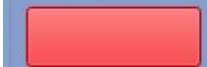
### i. 온도 센서 및 현재 킥보드 상태 표시

온도 센서는 I2C 통신을 통해서 **Expansion Board**으로 부터 현재 온도를 읽어들인다. 읽어온 온도에 따라서 화재 경보와 결빙 경보를 사용자에게 알리게 된다. 실제 실험 환경에서 낮은 온도는 실제로 재현하기 어려워 섭씨 24도를 기준으로 작동하도록 설정하였다. (설계 의도는 섭씨 5도를 기준으로 함) 반대로 높은 온도(섭씨 31도)의 경우 화재 경보를 사용자에게 알리게 된다. 2개의 경보는 킥보드 상태 표시창을 통하여 색에 따라서 다르게 표시하도록 한다. 킥보드 상태 표시창은 2개의 경보와 함께 시동의 상태도 사용자에게 알려주도록 한다. 화재 경보와 결빙경고는 **Critical**한 경보로 우선적으로 표시된다. 화재와 결빙 경고가 없을 경우 시동 상태에 대한 정보를 나타내도록 한다. 또한 화재와 결빙의 경우 다음 그림과



같이 계기판에 해당하는 ODD 파트에 불 모양과 눈 모양의 indicator에 표시되도록 한다.

현재 킥보드의 상태를 색상과 출력문으로 표기한다.

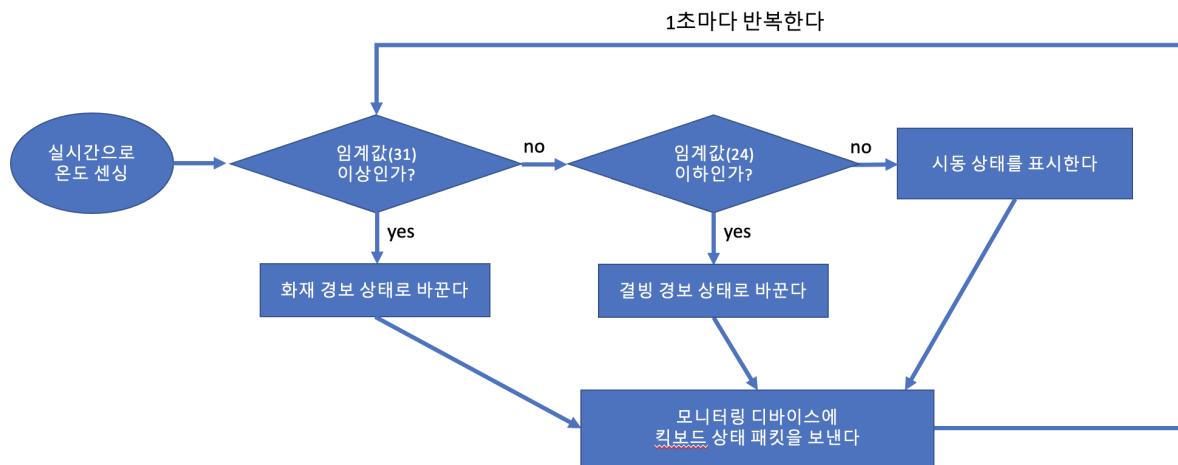
상태	컨트롤러	모니터링
시동 꺼짐	Power Off	<b>Scooter Status LED</b>   <b>TURN OFF</b>
시동 켜짐	Power On	<b>Scooter Status LED</b>   <b>TURN ON</b>
화재 경보	FIRE	<b>Scooter Status LED</b>   <b>FIRE WARNING</b>
결빙 경보	FREZZING	<b>Scooter Status LED</b>   <b>FREEZING WARNING</b>

컨트롤러	모니터링
<b>25.4 C</b>	<b>Sensor</b> <b>Temperature</b> 
<ul style="list-style-type: none"> <li>다른 시점에 스크린샷을 찍어 온도에 차이가 있습니다</li> </ul>	

컨트롤러가 expansion board에서 측정한 온도값을 패킷으로 전달해주면 그 값을 변환하여 온도값으로 만들어 화면에 표기한다. 컨트롤러에서 온도센서의 값에 따라

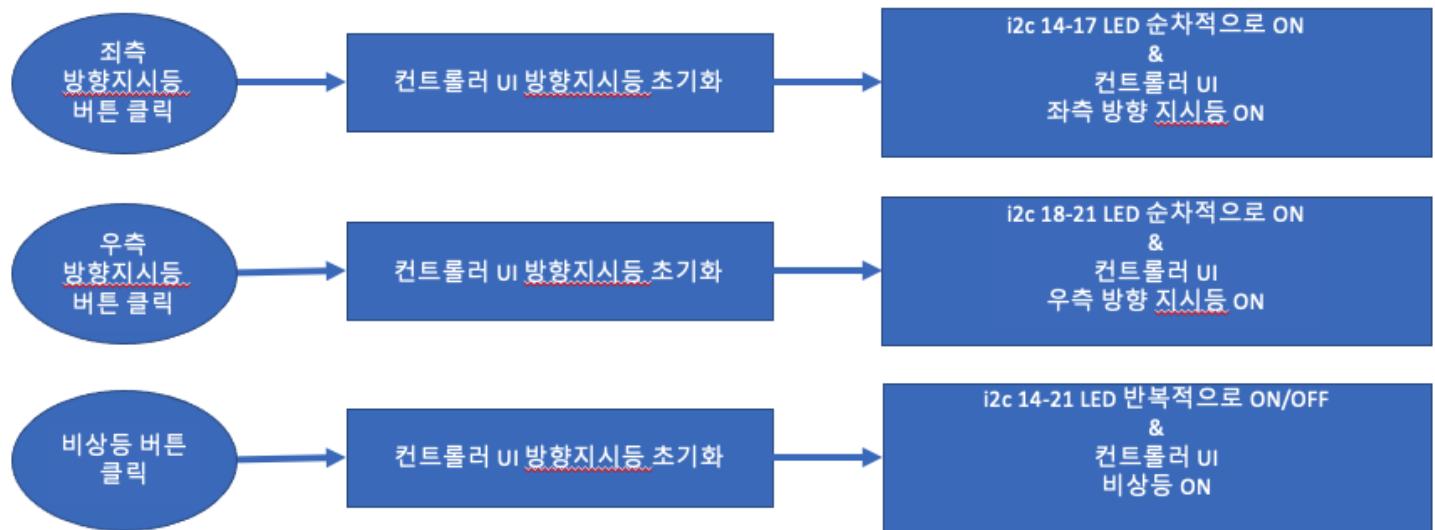
화재 & 결빙을 검출하고, 화재나 결빙과 같은 문제 상황이 아닐 경우 킥보드에 시동이 걸렸는지 걸리지 않았는지를 컨트롤러 화면에도 출력하고 이를 패킷으로 전송하여 모니터링 화면에도 출력한다.

온도 센서와 상태 표시 기능을 Function Flow Chart를 이용하여 아래와 같이 나타내었다.

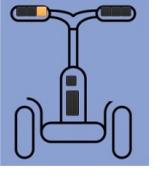
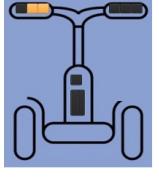
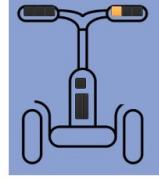
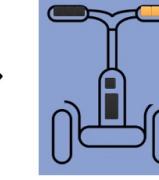
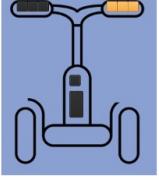


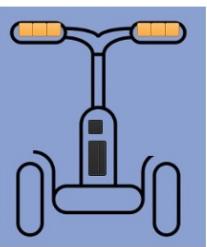
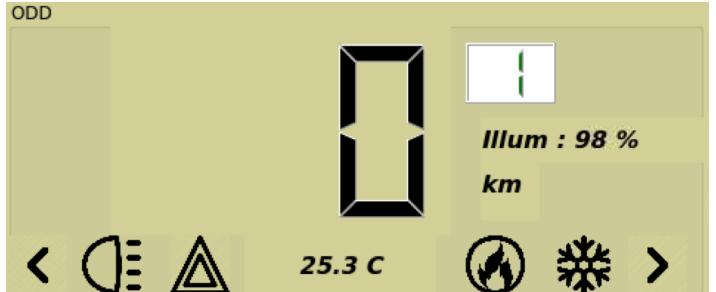
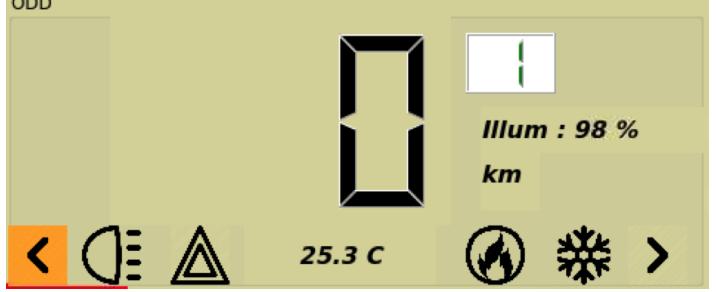
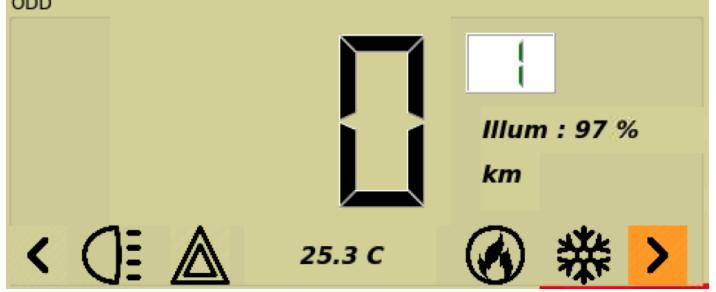
## ii. 방향 지시등 및 비상등

컨트롤러에 사용자의 입력에 따라 방향 지시등 및 비상등이 제어된다. 방향 지시등과 비상등은 하나의 그룹으로 묶여 동시에 눌리지 않도록 하였다. I2C expander의 Rotary의 8개의 LED를 이용하여 3가지 지시등을 표현하였다. 방향지시등은 QTimer를 사용하여 매초마다 중앙 기준으로 좌우로 하나씩 누적되어 LED가 켜지게 설계하였고 비상등은 8개의 LED가 매초마다 깜빡이도록 설계하였다. 비상등 버튼의 경우 토글 스위치처럼 사용 가능하도록 하였으며 방향지시등은 동시에 켜지지 않지만 완전히 끄기 위한 Signal Off 버튼을 따로 마련하였다.



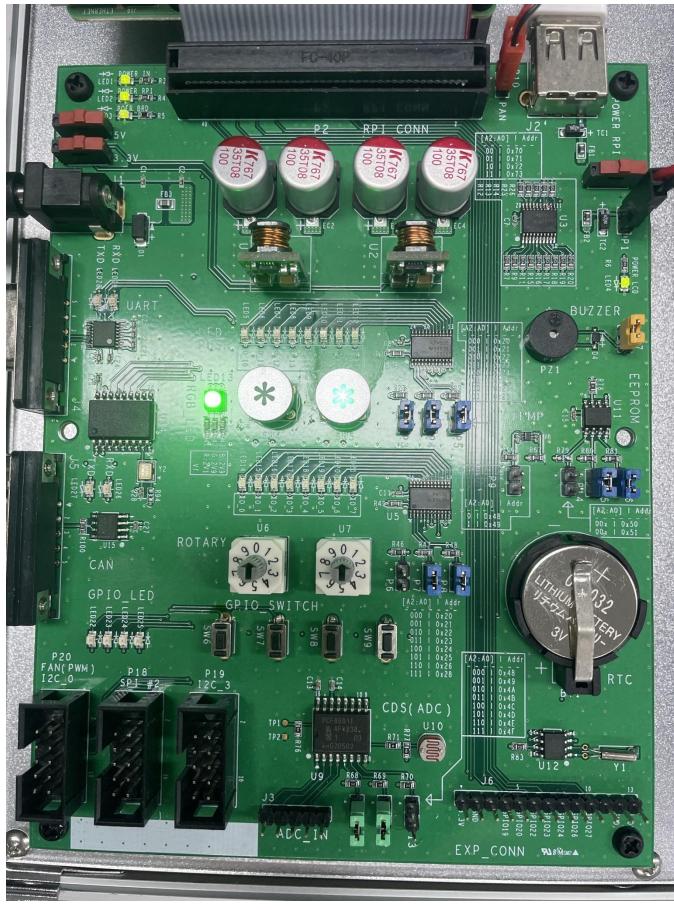
- 모니터링 및 컨트롤러 화면

모니터링		
OFF	<p>Turn Signal Light Status</p> 	
좌측 방향 지시등	<p>Turn Signal Light Status</p> 	  
우측 방향 지시등	<p>Turn Signal Light Status</p> 	  

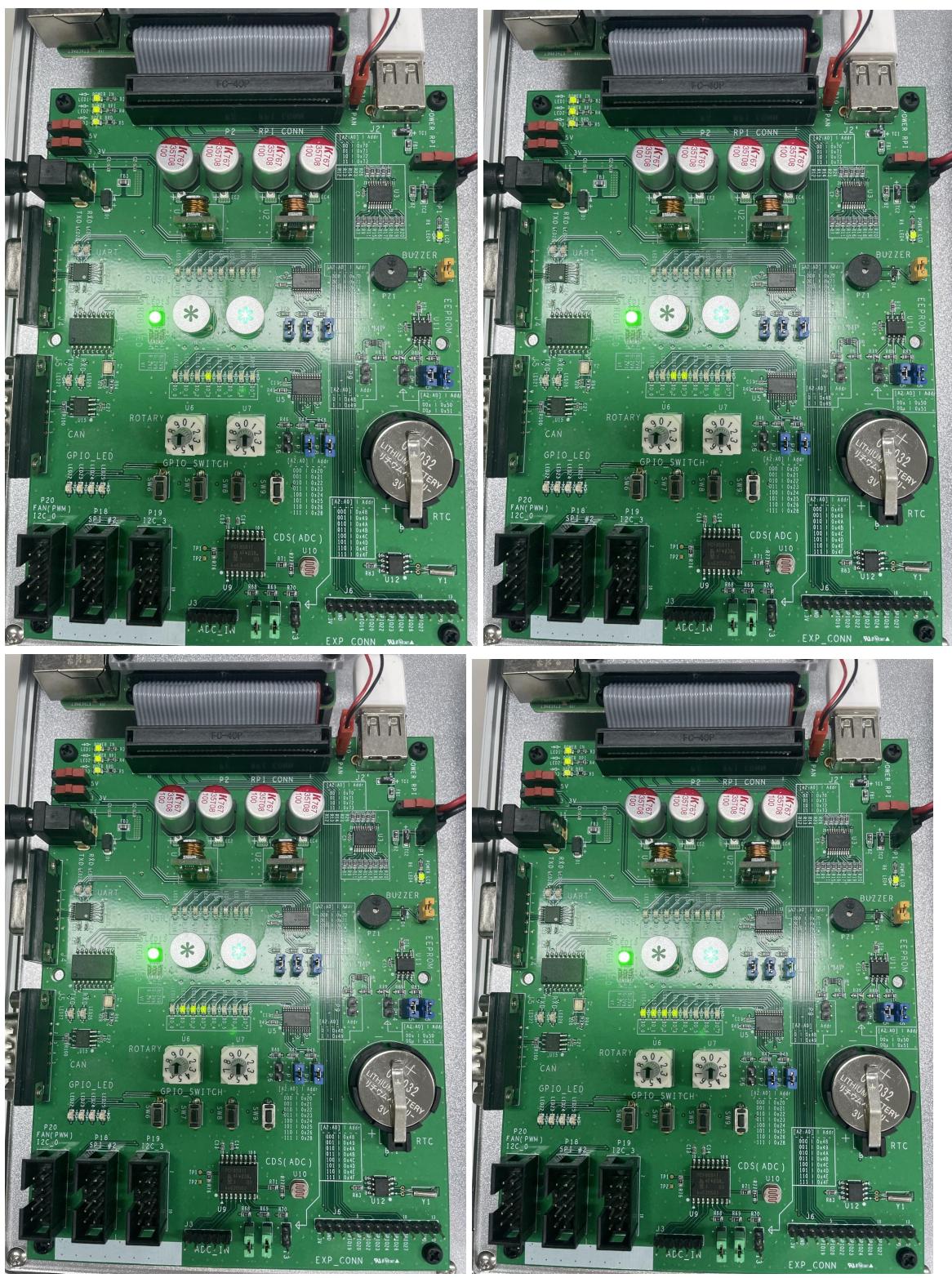
비상등	<p>Turn Signal Light Status</p> 	 <span>→</span> 
컨트롤러		
OFF	<p>Signal Off</p> 	
좌측 방향지시등	<p>Signal Off</p> 	
우측 방향지시등	<p>Signal Off</p> 	
비상등	<p>Signal Off</p> 	

## 하드웨어(Expansion Board)

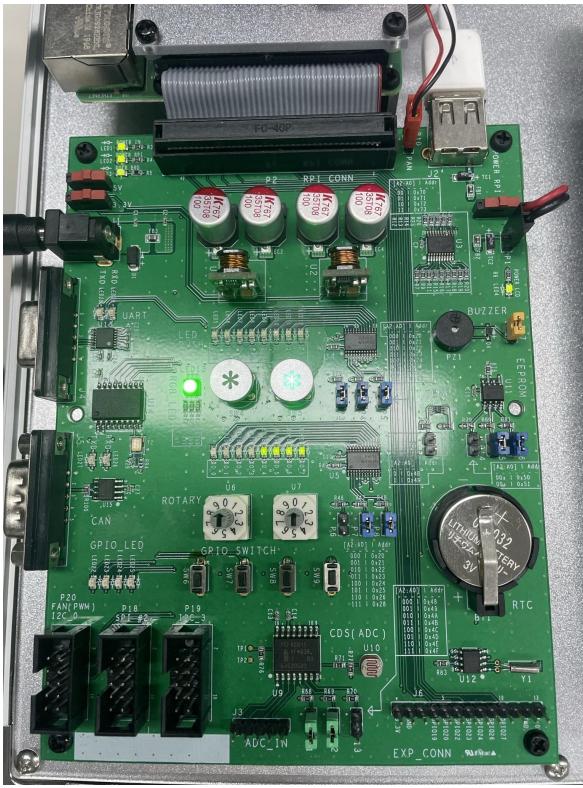
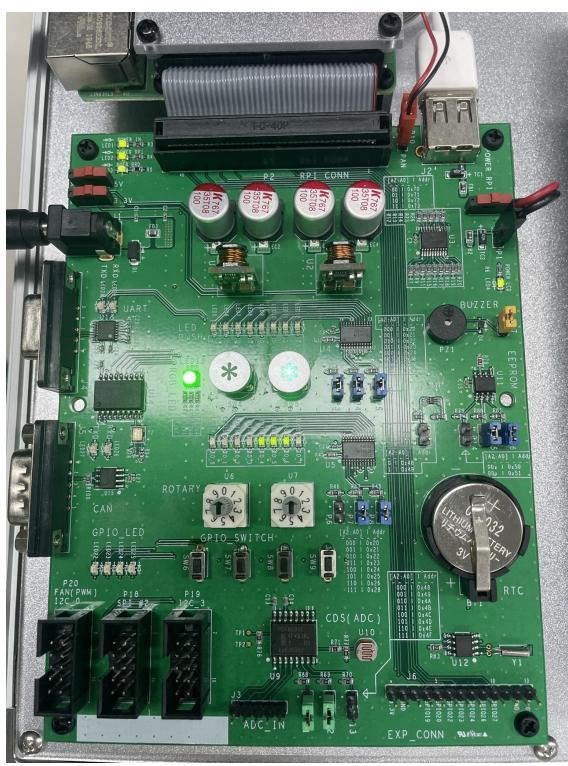
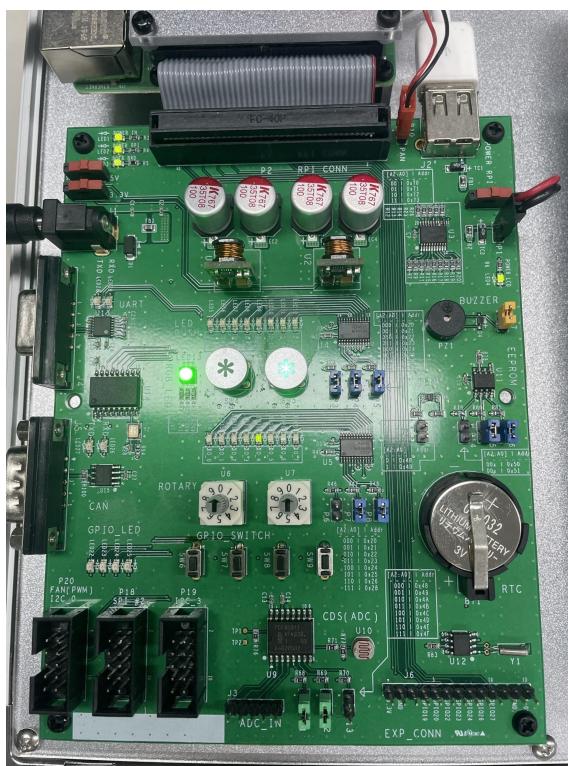
OFF



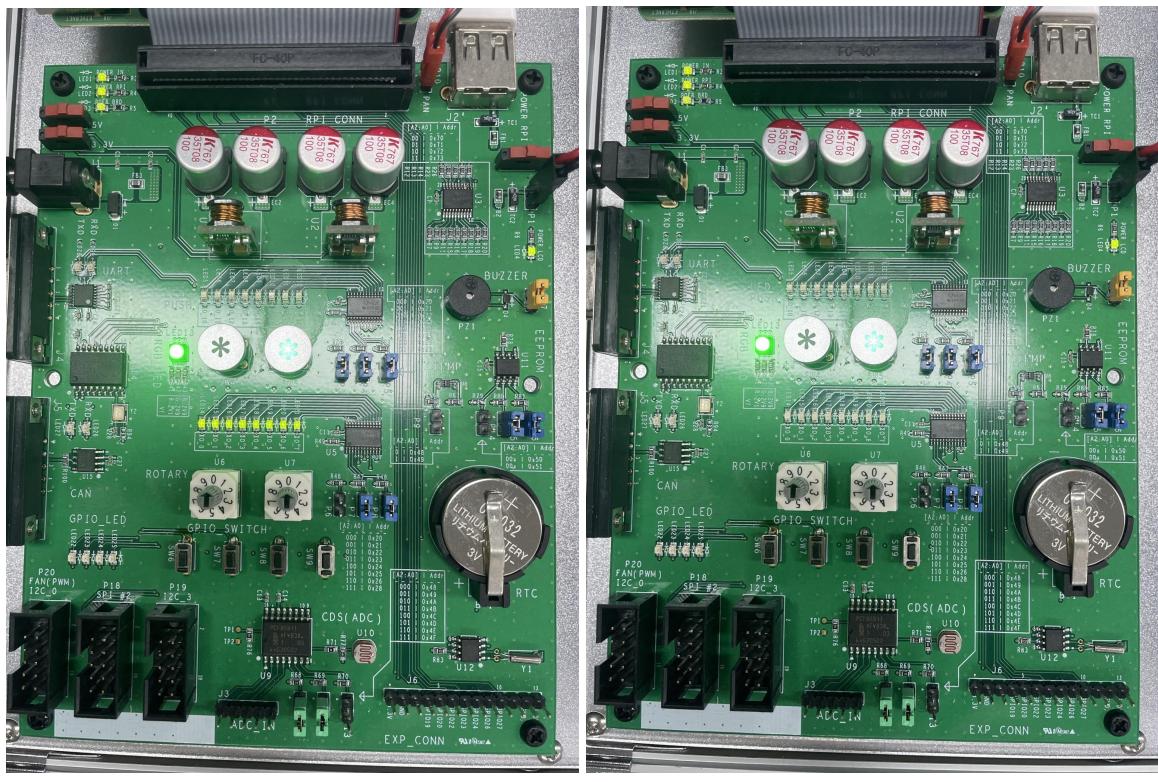
좌측  
방향지시등



방향지시등  
O



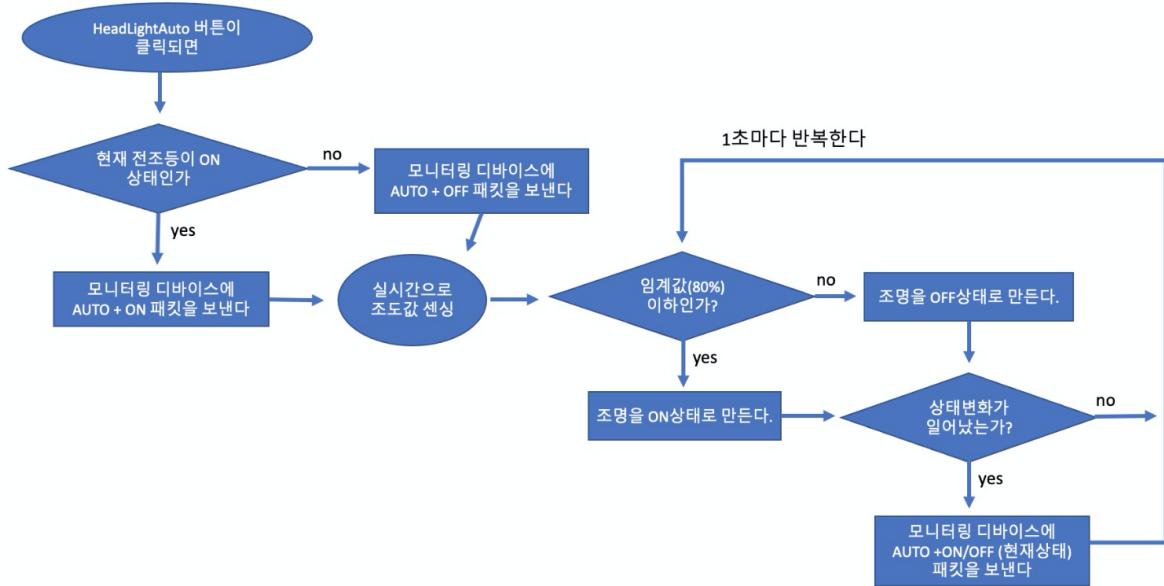
비상등



### iii. 조도 센서 및 전조등

조도 센서의 I2C를 이용하여 조도 센서의 값을 읽어오게 된다. 읽어온 조도센서의 값을 이용하여 백분율로 나타내어 컨트롤러 UI에서 ODD 내부에 표시하도록 하였는데 80%를 기준으로 헤드라이트 모드가 Auto일 경우 전조등이 켜지고 꺼지도록 하였다. 즉 조도센서의 값을 백분율로 나타내었을 때 80%보다 낮을 경우 헤드라이트 Auto 모드에서 헤드라이트를 자동으로 작동시키도록 하였다. 80%보다 높을 경우 Auto 모드에서 헤드라이트를 꼬도록 설계하였다. 사용자는 헤드라이트 모드를 설정할 수 있으며 Auto일 경우 조도센서의 값에 따라서 헤드라이트가 작동을 하고 Manual일 경우 사용자가 헤드라이트를 직접 제어를 하여야 한다. 헤드라이트 작동 여부는 Expansion Board의 SW4번의 LED로 나타내었다. 헤드라이트가 작동할 경우 해당 LED가 점등이 되며 작동을 하지 않을 경우 점멸하도록 하였다.

- 오토라이트 모드일 때 컨트롤러의 알고리즘을 flowchart로 나타낸 그림이다.

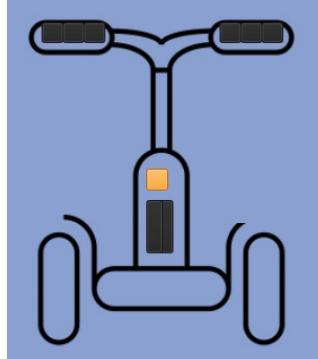
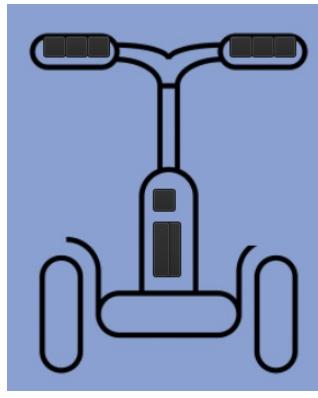


컨트롤러가 **expansion board**에서 측정한 조도값을 컨트롤러 화면에도 표기하고 그 값을 패킷으로 전달해주면 그 값을 변환하여 조도값으로 만들어 화면에 표기한다. 이 조도값에 따라 전조등의 오토라이트를 구현하였다.

**default** 값은 수동 모드이나 컨트롤러 화면의 버튼을 눌러 자동 모드로 모든 전환이 가능하다.

수동 모드일 경우 컨트롤러에서 **ON/OFF** 버튼을 직접 눌러 전조등을 제어한다. 모니터링 화면에서 **MANUAL** 초록색 버튼이 현재 수동모드임을 나타낸다.

모니터링

ON	<p><b>Headlight Status</b></p> <p><b>OFF</b> <b>ON</b> <b>MANUAL</b></p> 
OFF	<p><b>Headlight Status</b></p> <p><b>OFF</b> <b>ON</b> <b>MANUAL</b></p> 
컨트롤러	

ON

Lamp Control



On

Auto



Off

Manual

ODD



OFF

### Lamp Control



On

Auto



Off

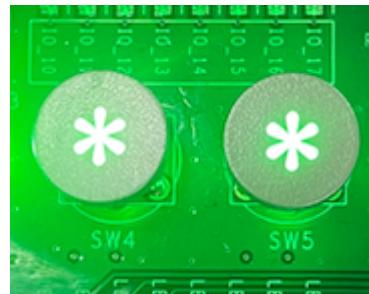
Manual

ODD

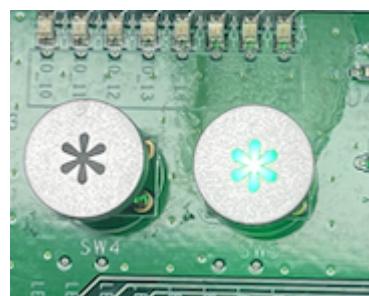


하드웨어(Expansion Board)

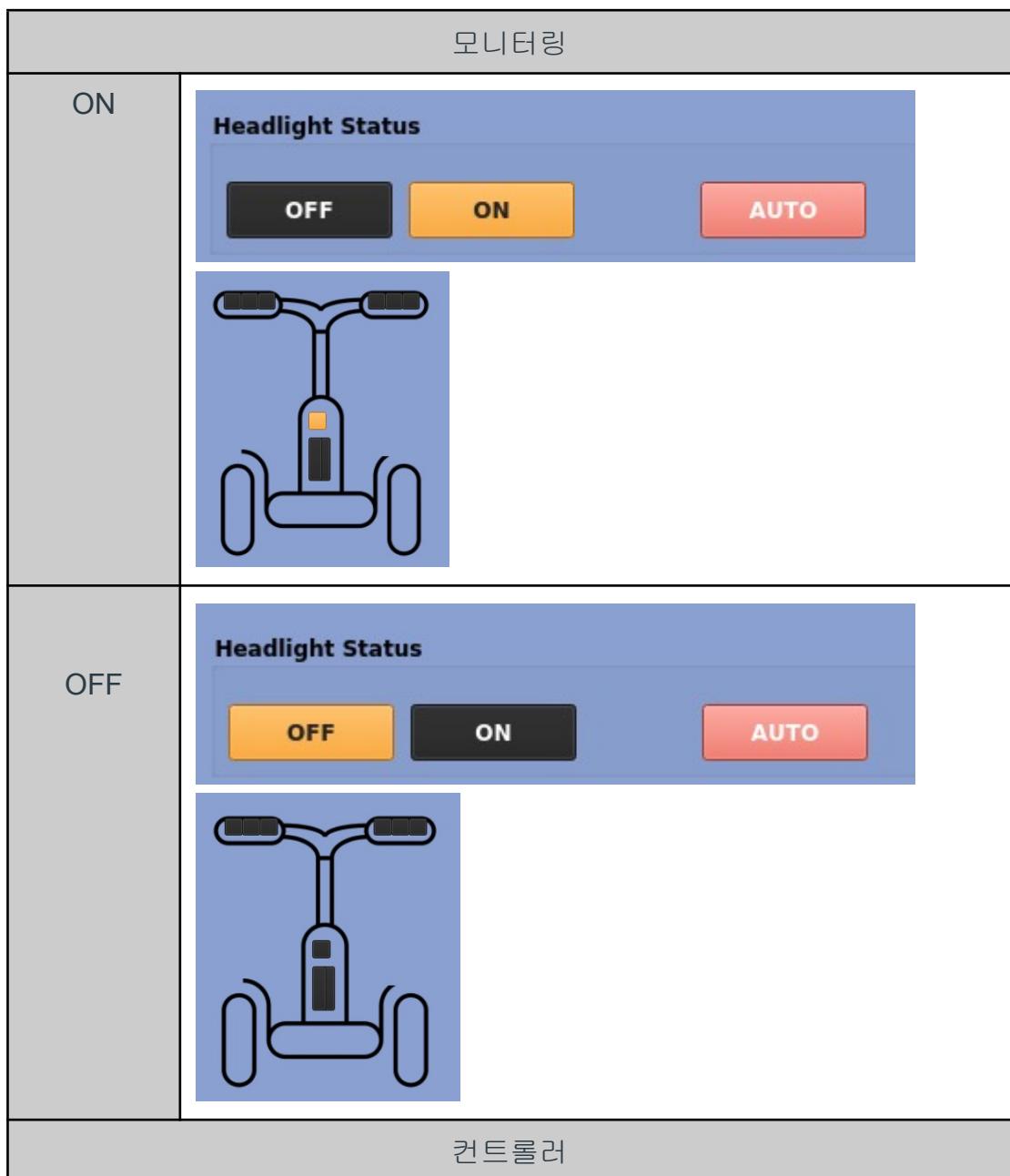
ON



OFF



컨트롤러 보드에서 자동 전조등으로 모드를 변경하면 현재 조도값에 따라 일정 threshold 이하일 경우(어두울 경우) led가 켜진다. 전조등의 ON/OFF 여부를 패킷으로 보낸다. 패킷의 내용에 따라 ON/OFF값이 버튼 UI 및 그림 UI에 표시되고 버튼 UI에는 AUTO 분홍색 button으로 모드도 표시된다.



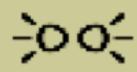
ON

Lamp Control



On

Auto



Off

Manual

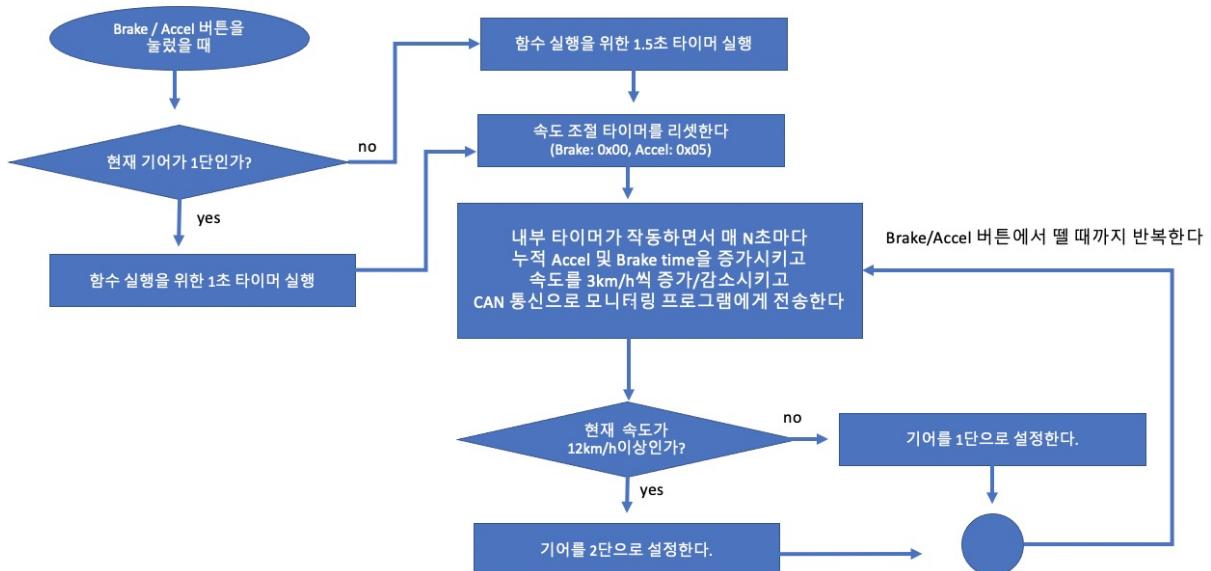
ODD



OFF	<p>Lamp Control</p> <p>ODD</p>
하드웨어(Expansion Board)	
ON	
OFF	

#### iv. 속도조절

악셀이나 브레이크를 누르는 시간이 속도에 반영되도록 타이머를 도입하였다. 다만 법률 규제상 현재 전동 킥보드의 최대 속력은 25km/h임을 고려하여 총 8단계로 나누어 속도를 제어하기 위해 24km/h까지 속력을 낼 수 있도록 시스템을 설계하였다. 악셀을 3초동안 눌렀다면 현재 속도에서 9km/h만큼 증가하는 방식이다. 또한 높은 속력에서는 더 낮은 속도로 속력이 증가할 수 있도록 기어를 도입하였다. 1단 기어는 12km/h까지 담당을 하게 되며 2단 기어의 경우 24km/h까지 담당하게 된다. 각 기어의 가속 기준 시간은 QTimer를 이용하여 다르게 하였으며 1단의 경우 3키로 가속하는데 1초가 걸리는 반면 2단의 경우 1.5초가 걸리도록 설계하였다. 1단 한계 속도가 되었음에도 사용자가 악셀을 누르고 있으면 3초 동안 변속 시간을 두었다. 3초가 지나게 되면 2단으로 변하게 되며 속도는 계속 증가하게 된다. 반면 브레이크의 경우 매초마다 3키로 씩 감소하도록 설계하였으며 0km/h 까지 감속 가능하도록 설계하였다. 악셀 또는 브레이크 버튼을 누르고 있는 시간은 GPIO LED를 통하여 표시하도록 하였다.



총 4개의 LED가 매초마다 누적적으로 점등되게 되는데 4개 LED가 모두 점등이 되었을 경우 3초 동안은 매초마다 4개의 LED가 동시에 점멸과 점등을 반복하게 된다.



모니터링 화면에서 브레이크나 압셀을 눌렀을 경우 그림UI에 표시된다.

자동차에서 브레이크 버튼이 더 넓은 것을 반영한 그림 UI이다.

- 압셀/브레이크 버튼 UI

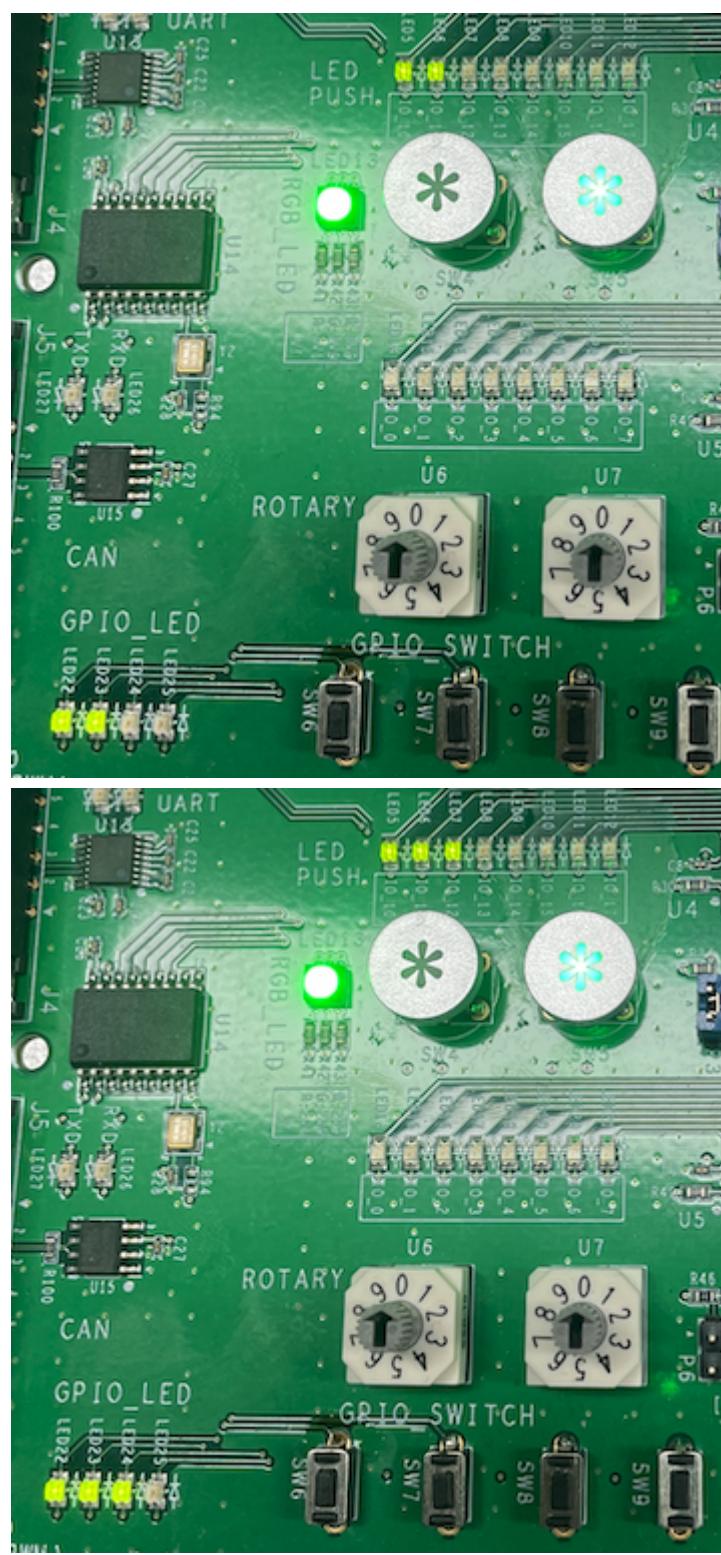
	브레이크	압셀
모니터링		

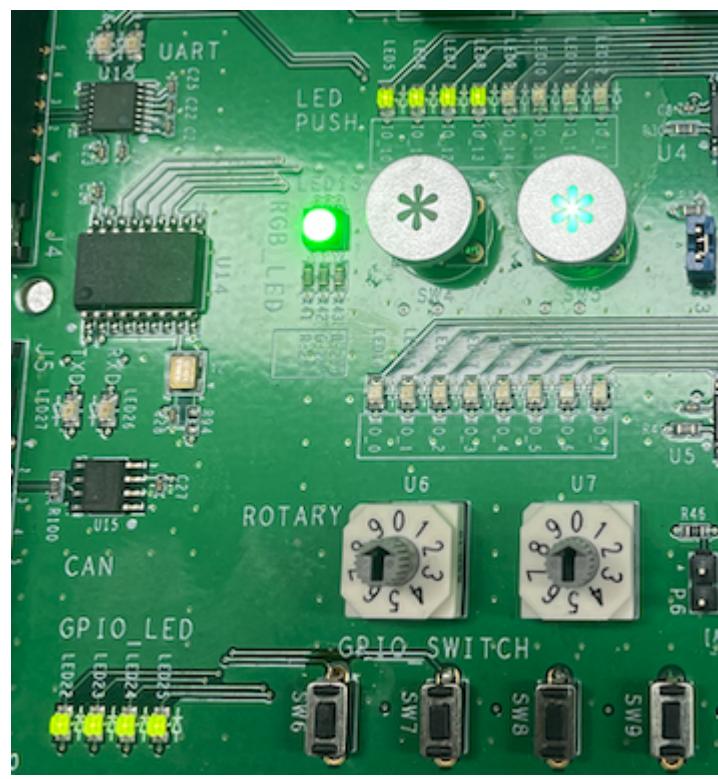


브레이크와 액셀을 타이머와 연동하여 사용하기 때문에 타이머의 표기도 필요하다.

- 속도조절 타이머 UI

모니터링	
하드웨어 (Expansion board)	





## V. 현재시간 출력

현재 시간 출력은 컨트롤러에서 I2C expansion 중 RTC의 값을 읽어와 컨트롤러 UI에 표시하게 된다. 시간 표시할 때에 24시간 기준으로 표시한다. 매초마다 변화하게 되어 CAN 통신을 통해 변화할 때마다 패킷에 시간을 실어 보내게되면 `delay`가 너무 심하게 되어 시동이 꺼진 상태에서 시동을 켈 때 한번만 보내도록 설계하였다.

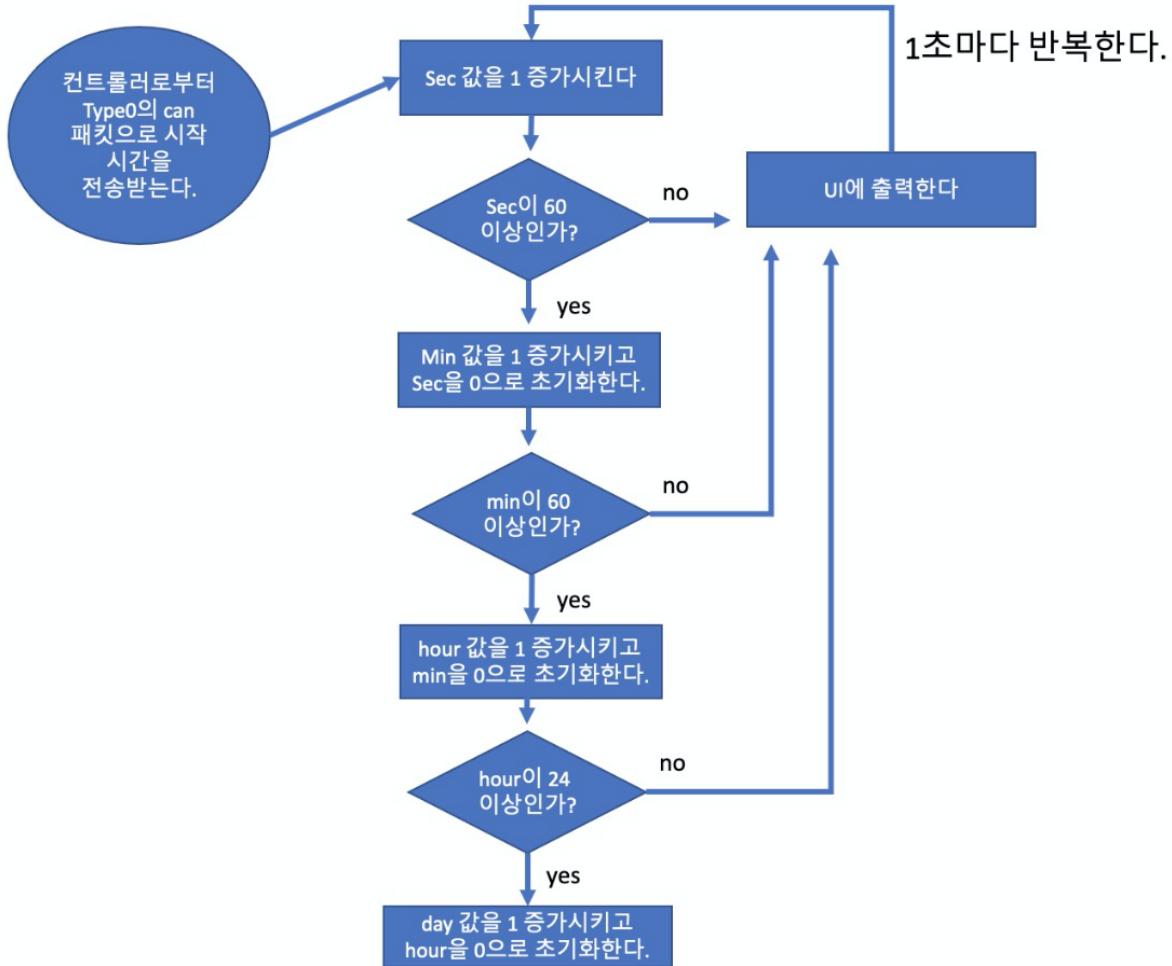
[System] Type0 Data Receive!

2021-12-08

03:12:04

다음과 같이 Type0 패킷을 시동 켈 때 받고 난 후 내부적으로 매초 시간을 업데이트하여 시간을 표시한다.





전동 킥보드이므로 며칠간 전원을 끄지 않고 사용하기는 어렵기 때문에 하루 단위가 넘어가는 것 까지만 알고리즘으로 처리해주었다.

## vi. CAN Connection Indicator

컨트롤러에서 CAN 통신이 정상적으로 이루어지고 있는지 확인을 위하여 상태를 표시하도록 설계하였다. CAN 통신 중 선이 제거하게 되면 'CAN No Signal' 메세지를 표시하고 정상적으로 연결이 되어있으면 'CAN CONNECTED'를 표시하도록 한다. 이는 CanOpen 하는 함수에서 정상적으로 CAN 통신이 열렸을 경우 connection 상태를 나타내는 boolean 타입의 변수 m\_CanConnection이 true로 바뀌고 정상적으로 통신이 안될 경우 false로 바뀐다. 또한 통신을 할 때마다 write를 통하여 패킷을 보내게 되는데 정상적으로

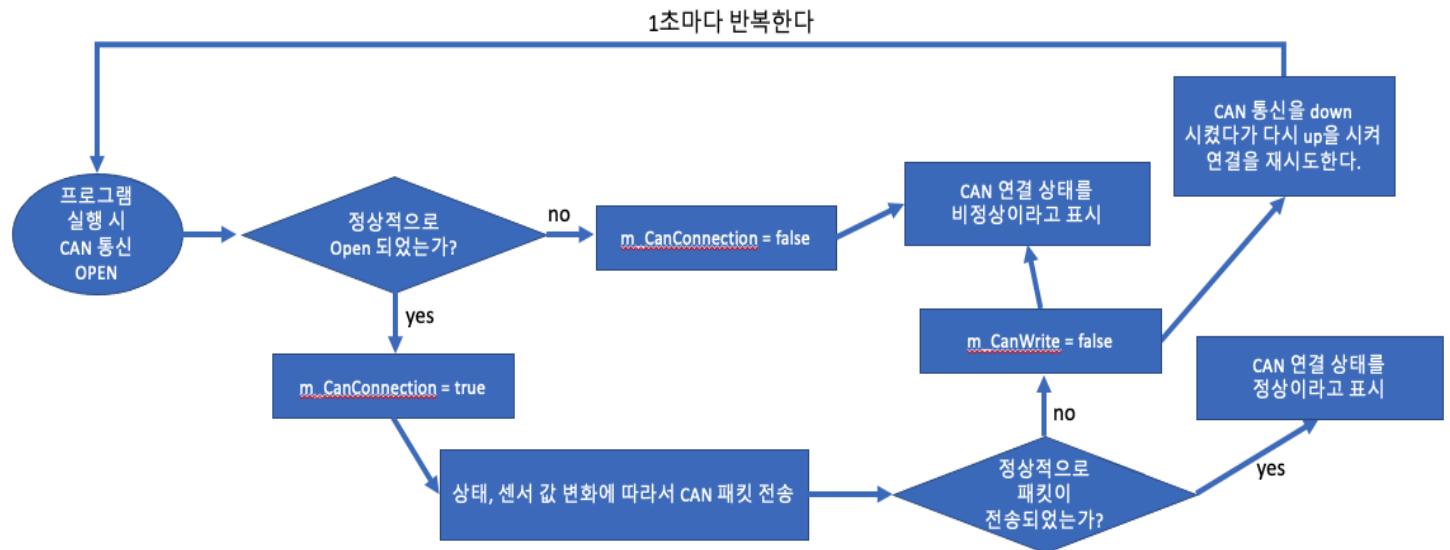
보내지면 1을 return하고 보내지지 않을 경우 0을 return하게 되는데 이것을 boolean 타입의 `m_CanWrite` 변수를 통하여 정상적으로 통신이 될 경우 `true`를 아닐 경우 `false`로 바꾼다. 두 가지의 boolean 타입 변수를 이용하여 하나라도 정상적이지 않을 경우 'CAN No Signal' 메세지를 표시하게 된다.

```
if(m_CanConnection && m_CanWrite) {  
    Can_val = CAN_OK;  
    ui->lbCan->setStyleSheet("color : white; background:  
limegreen;");  
    ui->lbCan->setText("CAN CONNECTED");  
    qDebug("canc : %d",m_CanConnection);  
    qDebug("\ncanw : %d",m_CanWrite);  
} else if(m_CanConnection && !m_CanWrite){  
    qDebug("canc : %d",m_CanConnection);  
    qDebug("\ncanw : %d",m_CanWrite);  
    system("sudo ip link set can0 down");  
    system("sudo ip link set can0 up type can bitrate  
125000");  
    Can_val = CAN_ERR;  
    ui->lbCan->setStyleSheet("color : red; background:  
transparent");  
    ui->lbCan->setText("CAN NO SIGNAL");  
} else{  
    qDebug("canc : %d",m_CanConnection);  
    qDebug("\ncanw : %d",m_CanWrite);  
    Can_val = CAN_ERR;  
    ui->lbCan->setStyleSheet("color : red; background:  
transparent");  
    ui->lbCan->setText("CAN NO SIGNAL");  
    openCanDevice();  
}
```

위와 같이 코드를 구성하였는데 만약 `write`가 비정상일 경우 컨트롤러의 CAN을 비활성화 시켰다가 다시 활성화를 시켜 다시 시도하도록 설계하였다. 다만 시간의 `delay`가

있음을 확인하였다. CAN 통신이 끊겼을 경우 센서 값을 실은 패킷이 전송되지 않아 에러 메세지를 금방 확인 할 수 있지만 다시 재개할 경우 **delay**가 있음을 확인하였다.

위의 설계를 Function Flow Chart 방법으로 나타내었다.



### 3. 결론

#### a. 실험 결과 및 동작방법

2개의 라즈베리파이와 Expansion Board를 이용하여 최종적으로 전동 킥보드를 제어하는 컨트롤러와 모니터링 프로그램을 구현하였다. 컨트롤러와 모니터링 프로그램 간의 통신은 CAN 프로토콜을 이용하여 컨트롤러에서 제어되는 정보와 센서 정보 그리고 상태 정보를 전달 받아 모니터링 프로그램에 표시하도록 설계하였다. 컨트롤러에서 제어되는 정보는 시동 버튼, 헤드라이트 작동 버튼, 액셀과 브레이크 버튼, 그리고 방향 지시등과 비상등 버튼이 있다. 센서 정보로는 조도 센서와 온도 센서가 있으며 조도 센서는 헤드라이트 작동에 관여를 하고 온도 센서는 화재와 결빙 경보를 표시하는 데 사용된다. 이후 모든 기능의 상태 정보는 CAN 통신을 통하여 모니터링 프로그램에 표시하였다. 각 기능을 하드웨어에 표시하기 위하여 I2C와 GPIO를 이용하여 LED 제어를 통하여 Expansion Board에 나타내었다. 컨트롤러와 모니터링 프로그램의 UI 구성은 QT creator를 이용하여 가장 직관적으로 구성 하려고 노력하였다.

#### b. 느낀점 & 개선점

##### i. 김성수

초기 디자인과 구현할 기능을 정할 당시와 개발하면서 추가된 부분도 많았고 다시 설계해야하는 기능들이 생겼다. 수업 중에 실습하는 I2C와 센서를 이용한 코딩은 그저 따라가는데 급급해 이해하는 시간이 필요했는데 마지막 프로젝트를 통하여 이해를 많이 하게되는 계기가 되었습니다.

##### ii. 이고은

전산 전공으로 라즈베리파이를 처음 다뤄보는 계기가 되었습니다. 핀에 따라 주소가 달라지는

것도 흥미로웠고 평소 객체지향이나 프로시저 프로그래밍에만 익숙했었는데, interrupt 기반으로 동작하는 임베디드 시스템 프로그래밍을 경험해볼 수 있었습니다.

### iii. 개선점

(1). 비밀번호 페이지를 만들어 로그인하면 창이 바뀌는 형태로 만들 계획이었는데 로그인 버튼을 눌렀을 때 창이 바뀌는 구현은 성공했지만 비밀번호를 입력하는 부분의 코딩에서 메모리 문제가 생겨 최종적으로 구현하지 못했습니다. 이를 구현한다면 좀 더 전동 킥보드의 보안성을 높일 수 있을 것이라 생각합니다.

(2). CAN 통신 연결 상태를 실시간으로 나타낼 수 있도록 설계하려고 노력하였다. 그러나 케이블이 제거되거나 통신에 문제가 있을 경우 **detect**하는 방법은 알았으나 다시 케이블이 연결되었을 때 딜레이 없이 효율적으로 확인하는 방법을 정확하게 찾지 못하였다. **Reference**를 더 많이 찾고 방법에 대한 연구를 하게 된다면 효율적이고 빠르게 **detection** 방법을 개선 할 수 있다.

## 4. 참고

### a. 역할분담

#### i. 김성수

- 컨트롤러 UI 설계, 컨트롤러 알고리즘 설계, 컨트롤러 코드 작성, I2C LED 제어, 센서 제어, **readme.pdf** 작성, 보고서 작성

#### ii. 이고은

- 효율적인 CAN 프로토콜 설계, 모니터링 프로그램 UI 설계, 모니터링 프로그램 구현, 보고서 작성, 최종 발표자료 작성

### b. 참고자료

CAN 프로토콜 디버깅에 사용

[https://wiki.rdu.im/\\_pages/Application-Notes/Software/can-bus-in-linux.html](https://wiki.rdu.im/_pages/Application-Notes/Software/can-bus-in-linux.html)

킥보드 속도 (24km/h로 제한)

<https://m.lawtimes.co.kr/Content/Article?serial=160119>