

<차량용 통신 설계 과제 보고서>



수업명 : 차량용 통신

담당 교수님 : 박영일 교수님

학과 : 전자시스템공학과

학번 : 20211428

이름 : 안태현

<실습 2,3,4,5번에 대한 코드는 따로 텍스트 파일로 첨부하였습니다>

1. 설계 주제 : CAN 통신과 다양한 아두이노 센서를 활용한 통신 설계

2. 실험 결과

1) CAN 버스 케이블을 이용한 데이터 전송 및 오실로스코프를 이용한 파형 관측

ㄱ. ID : 0x01, ID : 0x02 통신(통신 속도 : 500KBPS)

```
n1.ino
17 unsigned char data01[8] = {'0','2','4','6','8','a','c','e'};
18 unsigned char data02[8] = {'1','3','5','7','9','b','d','f'};
19
20
21 void setup()
22 {
23   Serial.begin(9600);
24   int count = 50;
25   do {
26     CAN.init();
27     if(CAN_OK == CAN.begin(CAN_500KBPS))
28     {
29       Serial.println("CAN BUS Shield init ok!");
30       break;
31     }
32     else
33     {
34       delay(100);
35       if (count == 0)
36       {
37         Serial.println("CAN BUS Shield init failed");
38       }
39     }
40   } while(count--);
41 }
```

```
n2.ino
14 unsigned char buf[8];
15 unsigned char data01[8] = {'0','2','4','6','8','a','c','e'};
16 unsigned char data02[8] = {'1','3','5','7','9','b','d','f'};
17
18 void setup()
19 {
20   Serial.begin(9600);
21   int count = 50;
22   do {
23     CAN.init();
24     if(CAN_OK == CAN.begin(CAN_500KBPS))
25     {
26       Serial.println("CAN BUS Shield init ok!");
27       break;
28     }
29     else
30     {
31       delay(100);
32       if (count == 0)
33       {
34         Serial.println("CAN BUS Shield init failed");
35       }
36     }
37   } while(count--);
38 }
```

Output Serial Monitor x

Message (Enter to send message to 'Arduino Uno' on 'COM8') New Line 9600 baud

Data : 1 3 5 7 9 b d f

ID : 2

Data : 1 3 5 7 9 b d f

ID : 2

Data : 1 3 5 7 9 b d f

ID : 2

Data : 1 3 5 7 9 b d f

Output Serial Monitor x

Message (Enter to send message to 'Arduino Uno' on 'COM7') New Line 9600 baud

Data : 0 2 4 6 8 a c e

ID : 1

Data : 0 2 4 6 8 a c e

ID : 1

Data : 0 2 4 6 8 a c e

ID : 1

Data : 0 2 4 6 8 a c e



* 500KBPS 통신에서 1bit : 2μs

오실로스코프로 신호의 파형을 검출했을 때, 입력 값이 0일때 CAN_H(3.48V)와 CAN_L(1.33V)의 차이(ΔY)가 2.15V로 이론 값인 2V와 오차가 거의 없었습니다. 이 때, Trigger는 Edge를 사용했습니다.

또, 처음 $2\mu s(\Delta x)$ 간격으로 1bit 씩 전송되면서 SOF부터 EOF까지 데이터 프레임이 생성되었고 Bit Stuffing도 잘 나타나고 있음을 확인할 수 있었습니다.

ㄴ. ID : 0x01, ID : 0x02 통신(통신 속도 : 1000KBPS)

```

n1.ino
17 unsigned char data01[8] = {'0','2','4','6','8','a','c','e'};
18 unsigned char data02[8] = {'1','3','5','7','9','b','d','f'};
19
20
21 void setup()
22 {
23   Serial.begin(9600);
24   int count = 50;
25   do {
26     CAN.init();
27     if(CAN_OK == CAN.begin(CAN_1000KBPS))
28     {
29       Serial.println("CAN BUS Shield init ok!");
30       break;
31     }
32     else
33     {
34       delay(100);
35       if (count == 0)
36         Serial.println("CAN BUS Shield init failed");
37     }
38   } while(count--);

```

```

n2.ino
14 unsigned char but[8];
15 unsigned char data01[8] = {'0','2','4','6','8','a','c','e'};
16 unsigned char data02[8] = {'1','3','5','7','9','b','d','f'};
17
18 void setup()
19 {
20   Serial.begin(9600);
21   int count = 50;
22   do {
23     CAN.init();
24     if(CAN_OK == CAN.begin(CAN_1000KBPS))
25     {
26       Serial.println("CAN BUS Shield init ok!");
27       break;
28     }
29     else
30     {
31       delay(100);
32       if (count == 0)
33         Serial.println("CAN BUS Shield init failed");
34     }
35   } while(count--);

```

Output Serial Monitor x

Message (Enter to send message to 'Arduino Uno' on 'COM8') New Line 9600 baud

Data : 1 3 5 7 9 b d f

ID : 2

Data : 1 3 5 7 9 b d f

ID : 2

Data : 1 3 5 7 9 b d f

ID : 2

Data : 1 3 5 7 9 b d f

Output Serial Monitor x

Message (Enter to send message to 'Arduino Uno' on 'COM7') New Line 9600 baud

Data : 0 2 4 6 8 a c e

ID : 1

Data : 0 2 4 6 8 a c e

ID : 1

Data : 0 2 4 6 8 a c e

ID : 1

Data : 0 2 4 6 8 a c e



* 1000KBPS 통신에서 1bit : $1\mu s$

오실로스코프로 신호의 파형을 검출했을 때, 입력 값이 0일때 CAN_H(3.61V)와 CAN_L(1.36V)의 차이(ΔY)가 2.25V였습니다. 이는 통신 속도가 500KBPS 일 때보다 더 큰 오차가 발생했음을 알 수 있는데, 통신 속도가 너무 빨랐거나 혹은 측정 기구의 노후화로 인한 노이즈가 끼어서 발생한 것으로 생각합니다.

또, 처음 $1\mu s(\Delta x)$ 간격으로 1bit 씩 전송되면서 SOF부터 EOF까지 데이터 프레임이 생성되면서 Bit Stuffing도 잘 나타나고 있음을 확인할 수 있었습니다.

데이터 프레임의 모양이 500KBPS일 때와 비교했을 때 좌우로 늘어난 모습을 볼 수 있었습니다. 이는 통신 속도가 빨라져서 같은 데이터를 보내는데 걸리는 시간이 짧아서, 더 자세한 파형을 보기 위해 스케일을 키웠기 때문입니다.

ㄷ. ID : 0x01, ID : 0x02 통신(통신 속도 : 100KBPS)

```

n1.ino
17 unsigned char data01[8] = {'0','2','4','6','8','a','c','e'};
18 unsigned char data02[8] = {'1','3','5','7','9','b','d','f'};
19
20
21 void setup()
22 {
23   Serial.begin(9600);
24   int count = 50;
25   do {
26     CAN.init();
27     if(CAN_OK == CAN.begin(CAN_100KBPS))
28     {
29       Serial.println("CAN BUS Shield init ok!");
30       break;
31     }
32     else
33     {
34       delay(100);
35       if (count == 0)
36       {
37         Serial.println("CAN BUS Shield init failed");
38       }
39     }
40   } while(count--);

```

```

n2.ino
14 unsigned char but[8];
15 unsigned char data01[8] = {'0','2','4','6','8','a','c','e'};
16 unsigned char data02[8] = {'1','3','5','7','9','b','d','f'};
17
18 void setup()
19 {
20   Serial.begin(9600);
21   int count = 50;
22   do {
23     CAN.init();
24     if(CAN_OK == CAN.begin(CAN_100KBPS))
25     {
26       Serial.println("CAN BUS Shield init ok!");
27       break;
28     }
29     else
30     {
31       delay(100);
32       if (count == 0)
33       {
34         Serial.println("CAN BUS Shield init failed");
35       }
36     }
37   } while(count--);

```

Output Serial Monitor x

Message (Enter to send message to 'Arduino Uno' on 'COM8') New Line 9600 baud

ID : 2
Data : 1 3 5 7 9 b d f

ID : 2
Data : 1 3 5 7 9 b d f

Output Serial Monitor x

Message (Enter to send message to 'Arduino Uno' on 'COM7') New Line 9600 baud

ID : 1
Data : 0 2 4 6 8 a c e

ID : 1
Data : 0 2 4 6 8 a c e



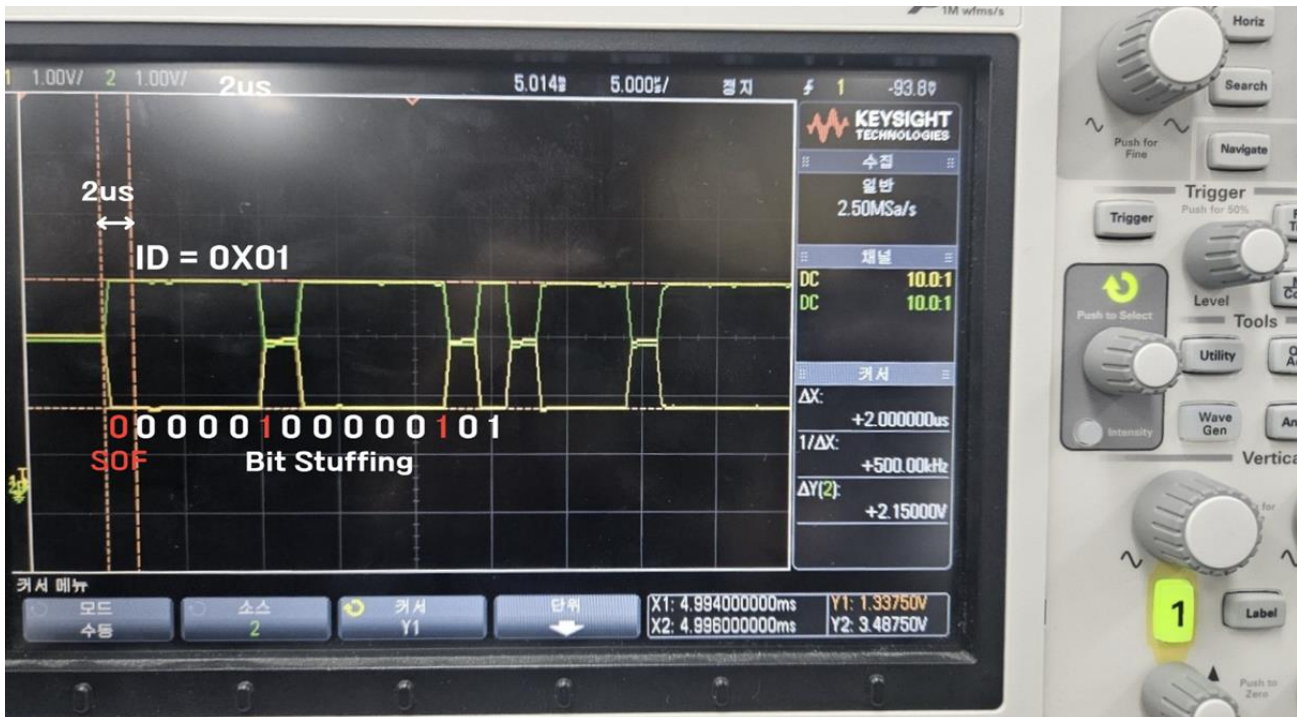
* 100KBPS 통신에서 1bit : 10μs

오실로스코프로 신호의 파형을 검출했을 때, 입력 값이 0일때 CAN_H(3.61V)와 CAN_L(1.36V)의 차이(ΔY)가 2.25V였습니다. 이는 통신 속도가 500KBPS 일 때보다 더 큰 오차가 발생했음을 알 수 있는데, 통신 속도가 너무 느렸거나 혹은 측정 기구의 노후화로 인한 노이즈가 생겨 발생한 것으로 생각됩니다.

또, 처음 10μs (Δx) 간격으로 1bit 씩 전송되면서 SOF부터 EOF까지 데이터 프레임이 생성되면서 Bit Stuffing도 잘 나타나고 있음을 확인할 수 있었습니다.

데이터 프레임의 모양이 500KBPS일 때와 비교했을 때 좌우로 늘어난 모습을 볼 수 있었습니다. 이는 통신 속도가 느려서 데이터가 전송되는 속도가 느리기 때문입니다. 전체적인 데이터 프레임은 잘 생성되었지만, 데이터 프레임 전체가 다 생성되기까지 시간이 오래 걸림을 확인할 수 있었습니다.

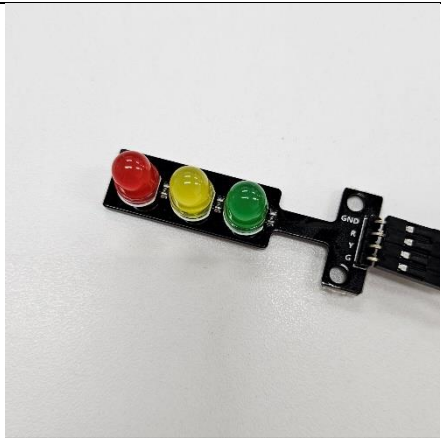
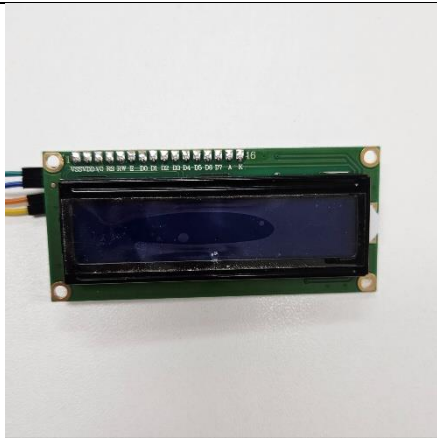
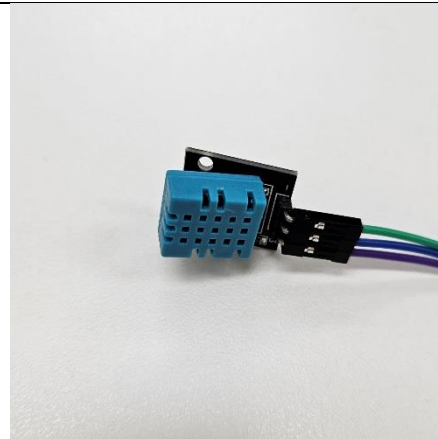
ㄹ. Bit Stuffing(비트 스템핑)



통신되고 있는 신호의 데이터 프레임을 보면, 동일한 유형의 신호가 연속해서 5bit 발생하면 강제로 반전된 bit를 내보내는 Bit Stuffing을 확인할 수 있습니다. 처음 SOF(1bit)로 동기화 시작되고 뒤이어 Arbitration Field bit들이 시작되는데, 이 부분에서 5번 연속으로 0이 출력되면서 중간에 0이 1로 Stuffing되는 모습을 확인할 수 있었습니다. 그 후로도, 연속해서 같은 유형의 신호가 5번 연속으로 들어올 경우 bit type이 바뀌며 데이터 프레임 구조가 생성됨을 확인할 수 있었습니다.

2) 온습도 센서를 이용한 데이터 통신과 모듈 제어 설계

ㄱ. 사용한 센서

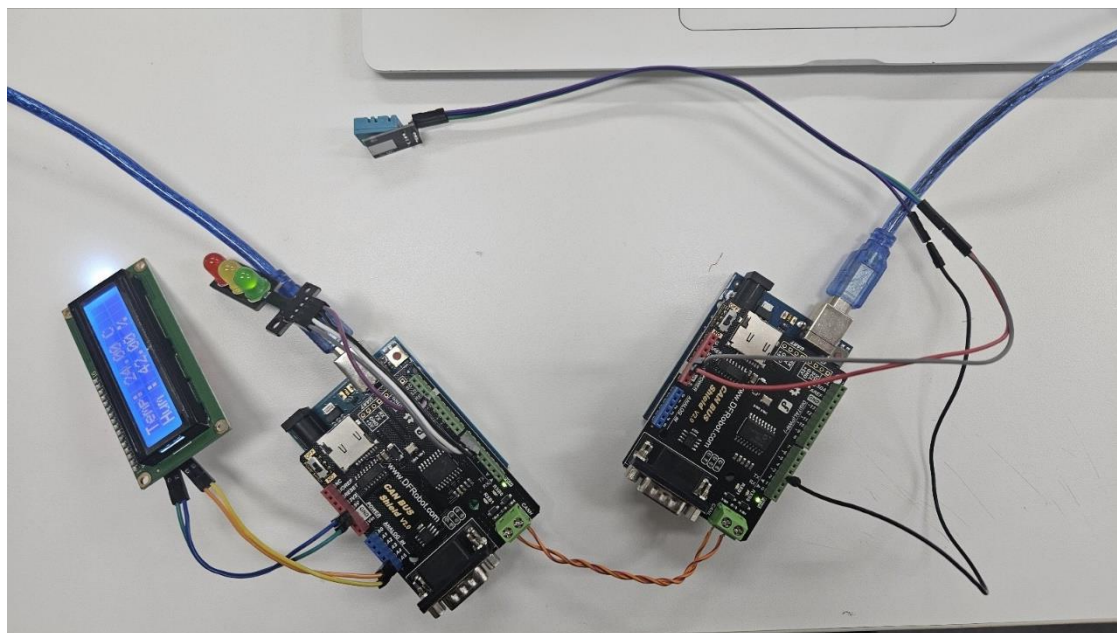
CAN1	CAN1	CAN2
		
신호등 LED	LCD	온습도 센서

CAN2에는 온습도 센서를, CAN1에는 LCD 모듈과 신호등 LED를 연결하였습니다.

ㄴ. 실험 개요

CAN2에서 온습도 센서를 통해 온도와 습도 데이터를 입력 받습니다. 이 데이터들을 CAN 버스를 통해 CAN1로 전송합니다. CAN1은 온습도 데이터를 수신하여 LCD와 신호등 LED로 표현합니다.

LCD에서는 온도와 습도 데이터를 1초마다 받아와 화면에 표시하였고, 신호등 LED에서는 습도 값의 범위에 따라 세 범위로 나누어 표시하였습니다.



<회로 사진>

- CAN1 : LCD + RGY LED
- CAN2 : 온습도 센서

ㄷ. 실험 결과

- 습도가 50% 이하일때



LCD에는 현재 센서가 인식한 온도와 습도의 정보를 출력하였습니다. 신호등 LED에서는 현재 습도의 수치가 50%를 넘지 않으면 초록색 LED가 켜지도록 프로그램을 설계하였습니다.

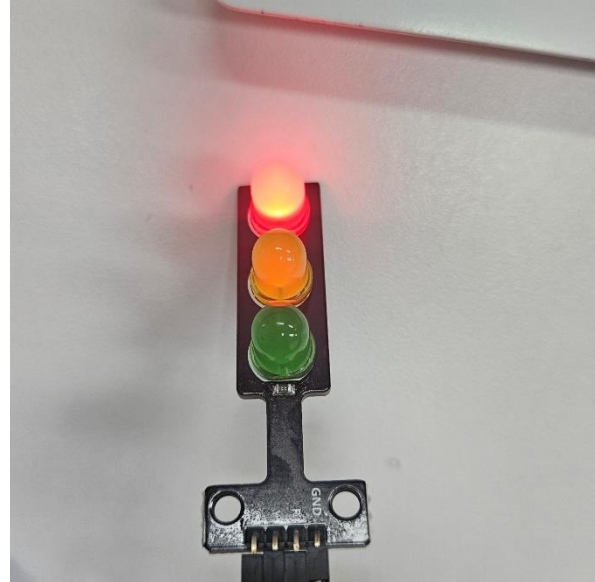
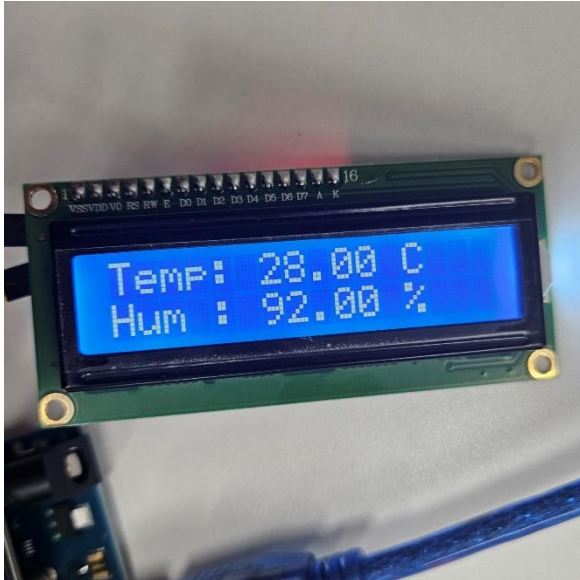
- 습도가 50% 초과 75% 이하일때



LCD에는 현재 센서가 인식한 온도와 습도의 정보를 출력하였습니다. 신호등 LED에서는

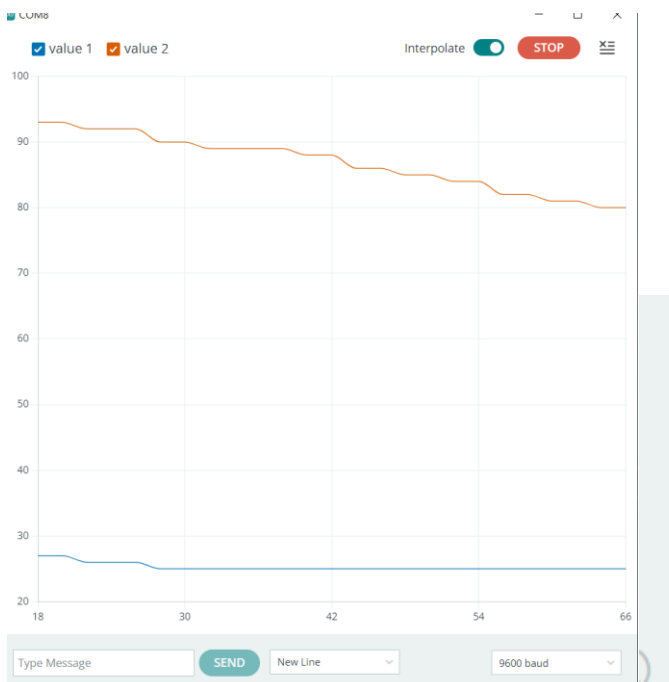
현재 습도의 수치가 50%초과 75% 이하이면 노란색 LED가 켜지도록 프로그램을 설계 하였습니다.

- 습도가 75% 초과일때



LCD에는 현재 센서가 인식한 온도와 습도의 정보를 출력하였습니다. 신호등 LED에서는 현재 습도의 수치가 75%를 초과하면 빨간색 LED가 켜지도록 프로그램을 설계하였습니다.

ㄷ. 실시간 데이터 확인



```
24.00 89.00
Received Temp: 24.00 C, Hum: 89.00 %
24.00 89.00
Received Temp: 24.00 C, Hum: 89.00 %
24.00 89.00
Received Temp: 24.00 C, Hum: 89.00 %
24.00 89.00
Received Temp: 24.00 C, Hum: 87.00 %
24.00 87.00
Received Temp: 24.00 C, Hum: 86.00 %
24.00 86.00
```


실시간으로 변하는 온도와 습도 값을 수신해 시리얼 모니터와 시리얼 플롯터로 시각화 하였습니다. 시리얼 플롯터에 Value1은 온도 값, Value2는 습도 값입니다.

3) CAN 함수의 Mask 및 Filter 기능 구현

ㄱ. Mask 및 Filter 기능을 위해 추가한 코드

```
CAN.init_Mask(MCP_RXM0, 0, 0x7ff);  
CAN.init_Mask(MCP_RXM1, 0, 0x7ff);  
CAN.init_Filter(MCP_RXF0, 0, 0x03);
```

실험1의 코드에 Mask와 Filter 기능을 위한 코드를 추가해 주었습니다. CAN 1에서는 여러 ID의 메시지를 전송하고, CAN2에서는 특정 ID만 수신하도록 하였습니다. 위의 코드는, CAN2에서 ID : 3인 메시지만을 수신하도록 설정하였습니다.

ㄴ . Mak 및 Filter 기능을 사용했을 때, 통신 결과

```
ID : 3  
Data : 3 3 3 3 3 3 3 3
```

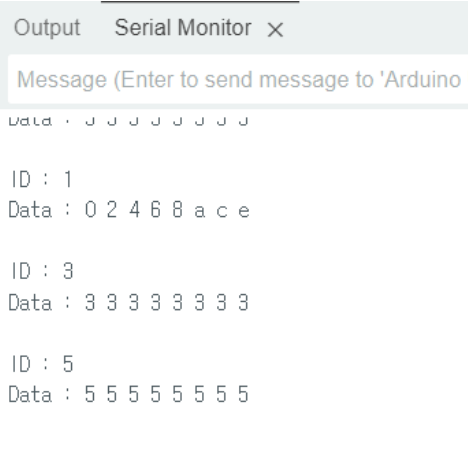
```
ID : 3  
Data : 3 3 3 3 3 3 3 3
```

```
ID : 3  
Data : 3 3 3 3 3 3 3 3
```

< CAN2에서의 출력 값 >

CAN1에서는 여러 ID의 메시지를 전송하지만, CAN2에서는 ID : 3의 메시지만 수신하 결과를 도출했습니다. 이는 Mask와 Filter를 설정해주는 코드를 추가했기 때문입니다.

ㄷ. Mask 및 Filter 기능을 사용하지 않았을 때, 통신 결과



Mask와 Filter 기능을 사용하지 않을 때에는, CAN1에서 보내는 여러 ID의 메시지가 모두 CAN2에 수신됩니다. 특정 ID의 메시지만 수신하는 Mask와 Filter 사용 프로세스와 달리, 모든 메시지가 수신된 후에 마이크로 컨트롤러가 모든 메시지의 ID를 검사해서 특정 ID가 있는 메시지를 처리하는 방법입니다.

전송 속도의 효율성 측면에서, 미리 특정 아이디만을 Filtering하여 메시지를 수신 받는 첫 번째 방법이 데이터를 더 효율적이게 주고받는 통신 기법이라고 생각합니다.

4) CAN 버스를 다중으로 연결하고, 다양한 센서와 모듈 활용하기

ㄱ. CAN 노드 구성

CAN1 (온습도센서)	CAN2 (신호등 LED)	CAN3 (LCD)	CAN4 (서보 모터)	CAN5 (시리얼 모니터)	CAN6 (실시간 차트)

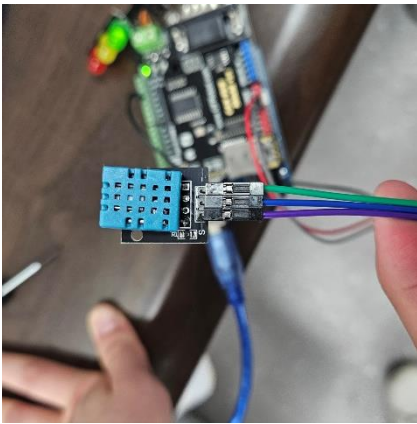
ㄴ. 실험 개요

- CAN1에서는 온습도 센서로 온도와 습도 값을 측정한 뒤, CAN2~6으로 송신합니다.
- CAN2에서는 수신한 습도 데이터를 세 범위로 나누어 신호등으로 표현합니다.

- CAN3에서는 수신한 온도와 습도 데이터를 LCD 화면에 나타냅니다.
- CAN4에서는 수신한 온도 값이 25도 이상이면서 습도 값이 60% 이상이면 서보 모터가 180도 돌아갑니다.
- CAN5에서는 수신한 온도와 습도 데이터를 시리얼 모니터에서 출력합니다.
- CAN6에서는 수신한 온도와 습도의 데이터 변화를 시리얼 플로터로 시각화 합니다.

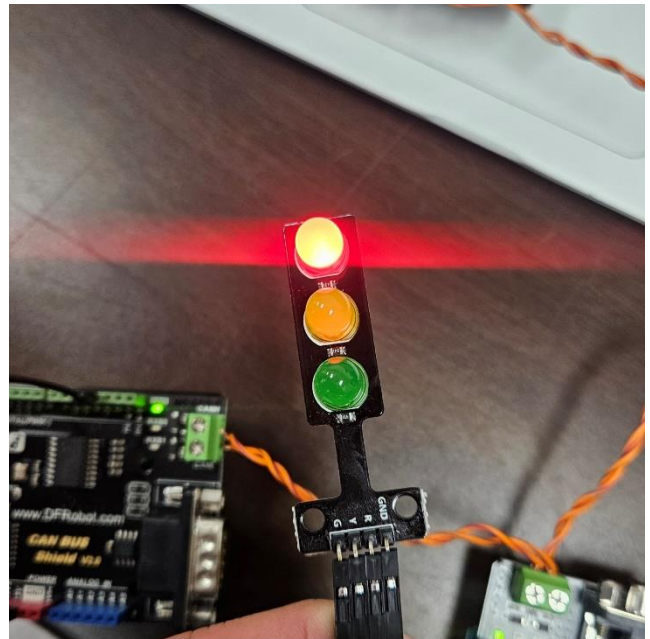
ㄷ. 실험 결과

- CAN1 : 온습도 값 측정



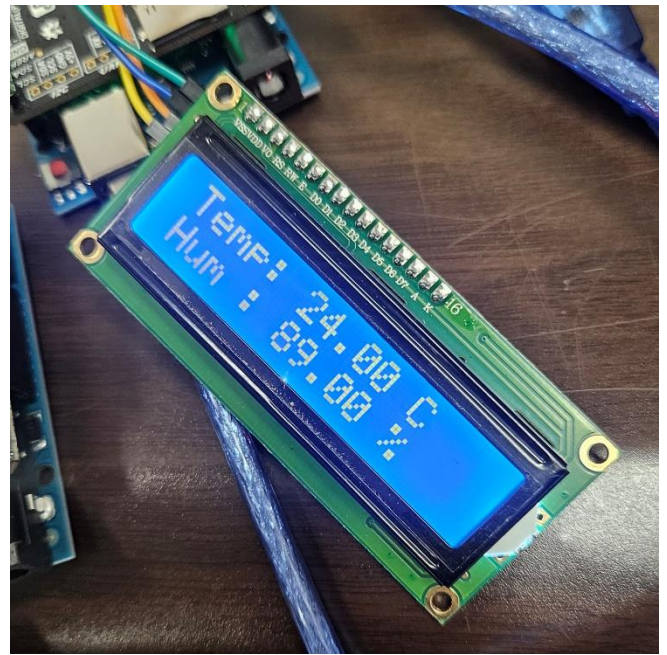
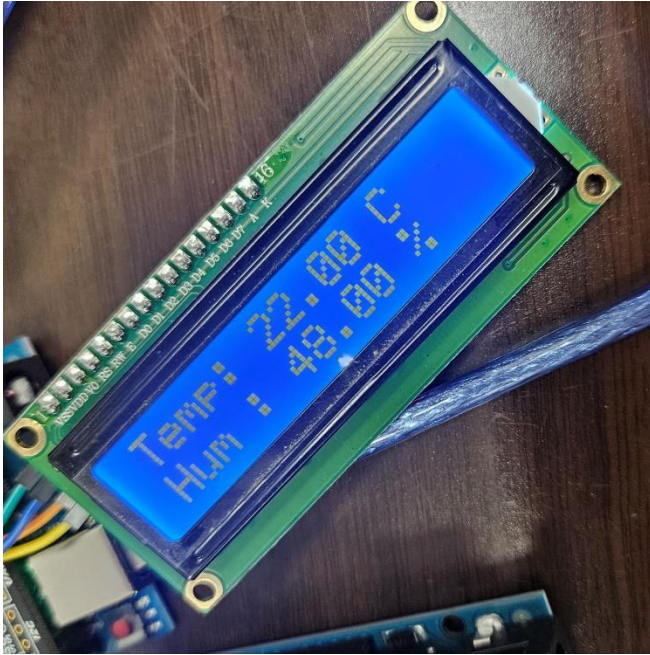
< 온습도 센서로 측정 >

- CAN2 : 신호등 LED



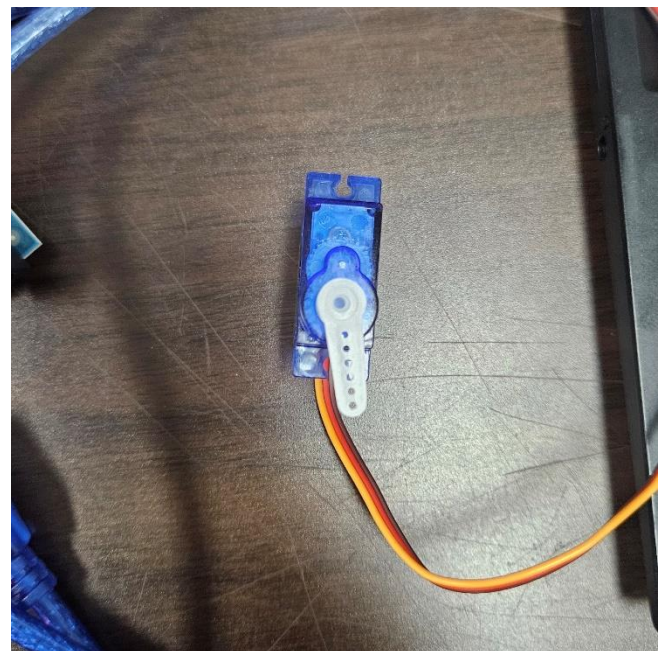
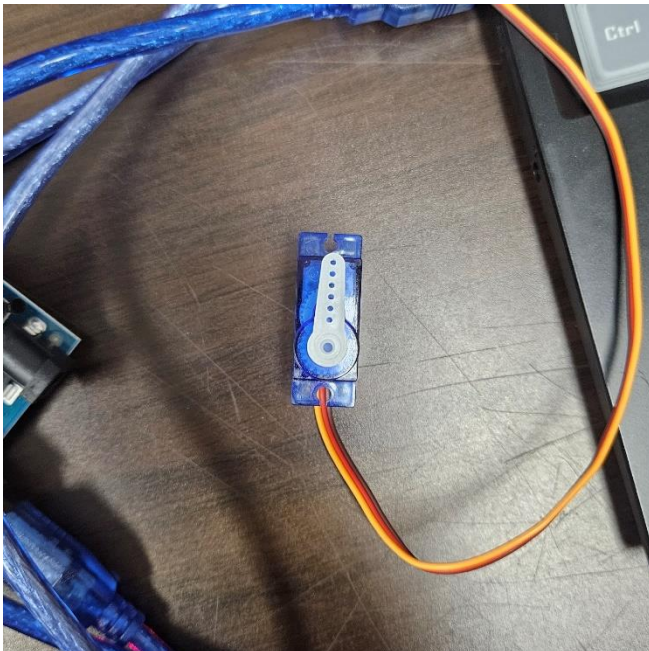
온습도 측정 값이 달라짐에 따라 실시간으로 불이 들어오는 신호등 LED의 색이 바뀜을 확인하였고, 이를 통해 CAN1에서 보낸 데이터가 CAN2에서 잘 수신됨을 알 수 있었습니다.

- CAN3 : LCD



온습도 측정 값이 달라짐에 따라 실시간으로 LCD에 표시되는 온도 값과 습도 값이 바뀜을 확인하였고, 이를 통해 CAN1에서 보낸 데이터가 CAN3에서 잘 수신됨을 알 수 있었습니다.

- CAN4 : 서보 모터



온도가 25도 이상이면서 습도가 60% 이상으로 측정될 때 서보 모터가 180도 회전(오른쪽 노트북이 기준선)하도록 설계했고, 실제로도 동작함을 확인했습니다. 이를 통해, CAN1에서 실시간으로 변하는 데이터가 CAN4에서 잘 수신됨을 알 수 있었습니다.

- CAN5 : 시리얼 모니터

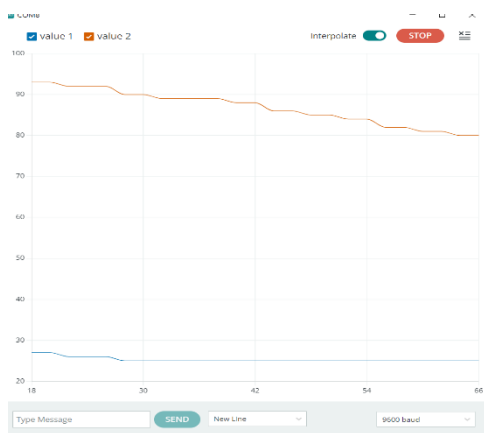
```

Message (Enter to send message to 'Arduino I
25.00 82.00
Received Temp: 25.00 C, Hum: 82.00 %
25.00 82.00
Received Temp: 25.00 C, Hum: 81.00 %
25.00 81.00
Received Temp: 25.00 C, Hum: 81.00 %
25.00 81.00
Received Temp: 25.00 C, Hum: 80.00 %
25.00 80.00
Received Temp: 25.00 C, Hum: 80.00 %
25.00 80.00

```

=> 시리얼 모니터를 이용해 CAN5에서 데이터 수신 확인

- CAN6 : 시리얼 플로터

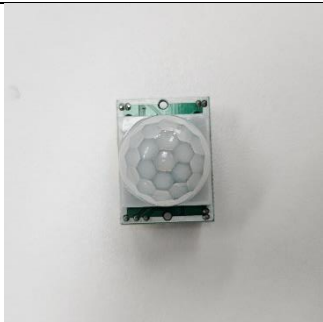
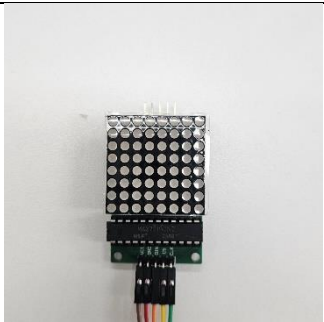
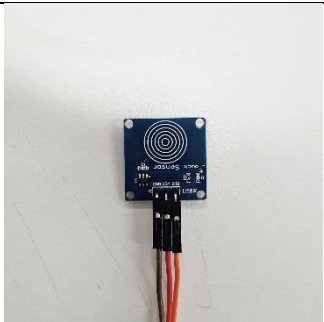



=> 시리얼 플로터를 이용해 CAN6에서 데이터 수신 확인

5) CAN 통신의 다른 기능 사용해보기

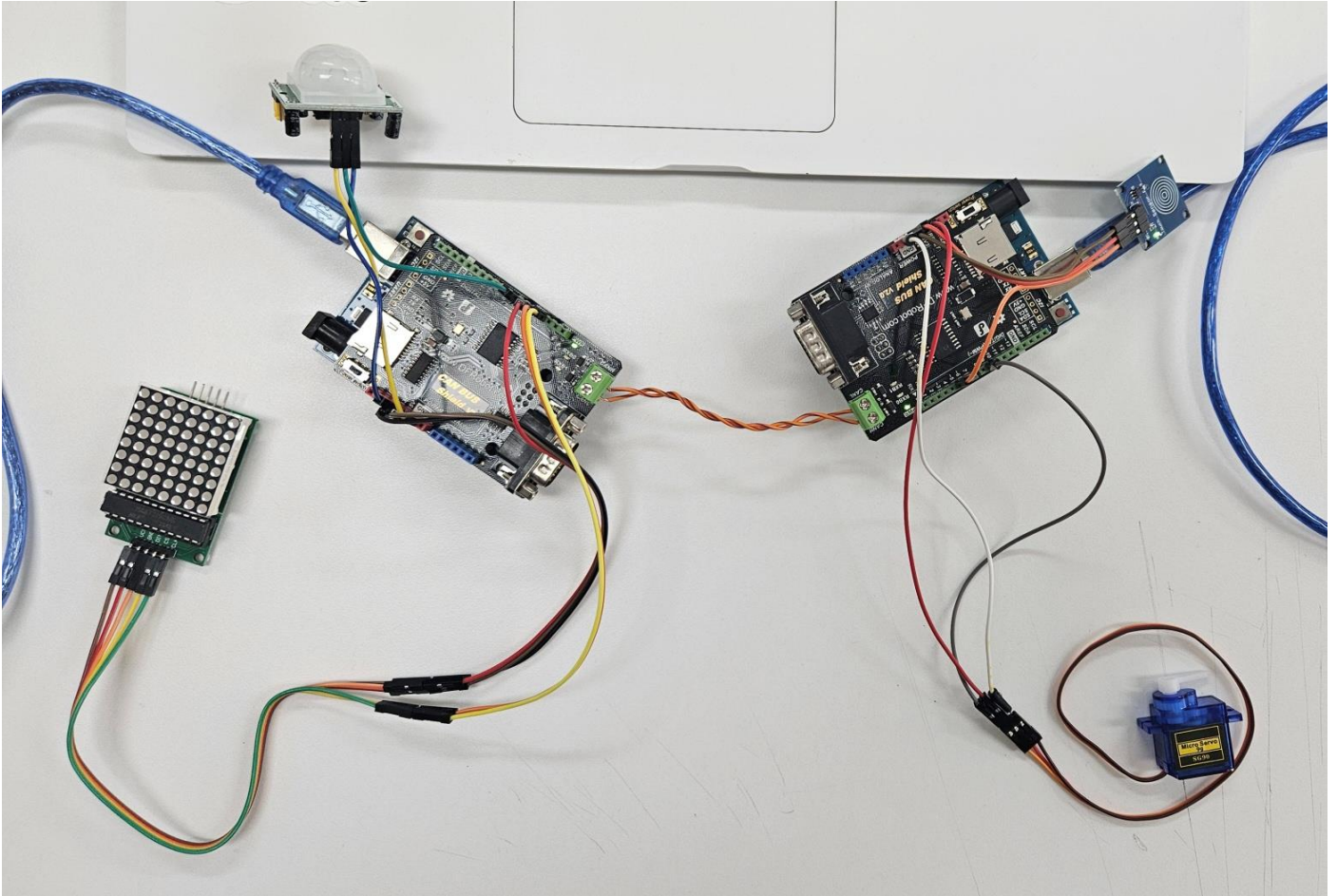
- 실험 주제 : 양방향 통신과 모듈 제어

ㄱ. 사용한 센서와 모듈

CAN1(센서)	CAN1(모듈)	CAN2(센서)	CAN2(모듈)
			
인체 감지 센서	8X8 매트릭스 LED	터치 센서	서보 모터

ㄴ. 실험 개요

- CAN1에서 인체 감지 센서를 통해 데이터를 측정하고 CAN2로 전송합니다.
- CAN2에서 터치 센서를 통해 데이터를 측정하고 CAN1로 전송합니다.
- CAN1에서 터치 감지 신호를 수신하면 8X8 매트릭스 LED에 불이 들어오도록 설계합니다.
- CAN2에서 인체 감지 신호를 수신하면 서보 모터가 와이퍼처럼 동작하도록 설계합니다.



㉔. 실험 결과

- CAN1과 CAN2 간의 데이터 전송이 잘 이루어지는지 시리얼 모니터로 확인

SensorN1.ino

```
1 #include <df_can.h>
2 #include <SPI.h>
3 #include "LedControl.h"
4
5 const int SPI_CS_PIN = 10;
6 MCP23017 CAN(SPI_CS_PIN);
7 const int humanSensorPin = 7; // 인체 감지 센서 핀
8 LedControl lc = LedControl(5, 4, 3, 1); // Pins: 5 = Data, 4 = Clock, 3 = CS
9
10 void setup() {
11   Serial.begin(9600);
12   pinMode(humanSensorPin, INPUT);
13
14   lc.shutdown(0, false); // 절전 모드 해제
15   lc.setIntensity(0, 8); // 밝기 설정 (0~15)
16   lc.clearDisplay(0);
17   CAN.init();
18
19   if (CAN.begin(CAN_1000KBPS) == CAN_OK) {
20     Serial.println("CAN BUS Shield init ok!");
21   } else {
22     Serial.println("CAN BUS Shield init failed!");
23   }
24 }
```

Output Serial Monitor x

Message (Enter to send message to 'Arduino Uno' on 'COM7')

New Line 9600 baud

Received touch signal from CAN2. Activating 8x8 matrix...
Message received with ID: 1, First byte: 84
Received touch signal from CAN2. Activating 8x8 matrix...
Message received with ID: 1, First byte: 84
Received touch signal from CAN2. Activating 8x8 matrix...
Motion detected! Sending message to CAN2...
Motion detected! Sending message to CAN2...
Message received with ID: 1, First byte: 84
Received touch signal from CAN2. Activating 8x8 matrix...
Motion detected! Sending message to CAN2...
Motion detected! Sending message to CAN2...

SensorN2.ino

```
1 #include <df_can.h>
2 #include <SPI.h>
3 #include <Servo.h>
4
5 const int SPI_CS_PIN = 10;
6 MCP23017 CAN(SPI_CS_PIN);
7 Servo servoMotor;
8 const int touchSensorPin = 5; // 터치 센서 핀
9 const int servoPin = 9; // 서보 모터 핀
10
11 void setup() {
12   Serial.begin(9600);
13   pinMode(touchSensorPin, INPUT);
14   servoMotor.attach(servoPin);
15   CAN.init();
16   if (CAN_OK == CAN.begin(CAN_1000KBPS)) {
17     Serial.println("CAN BUS Shield init ok!");
18   } else {
19     Serial.println("CAN BUS Shield init failed!");
20     while (1);
21   }
22 }
```

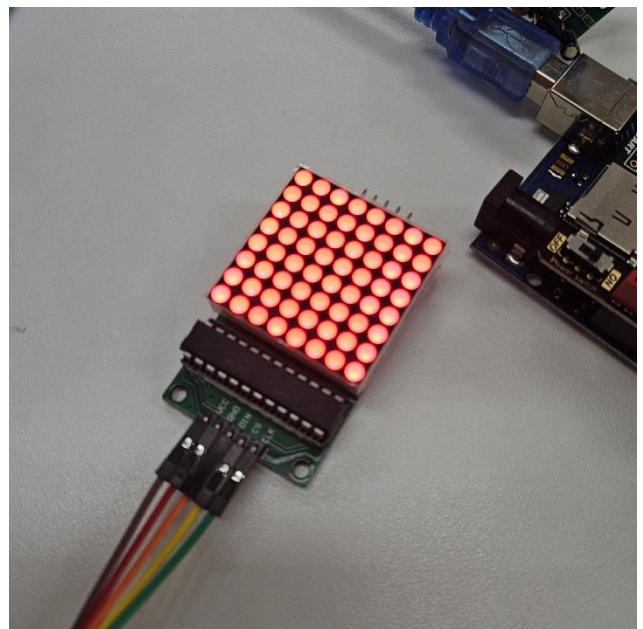
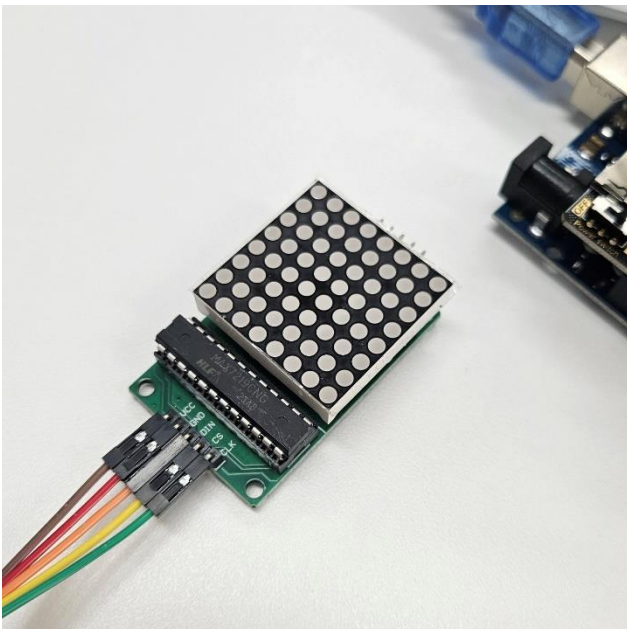
Output Serial Monitor x

Message (Enter to send message to 'Arduino Uno' on 'COM8')

New Line

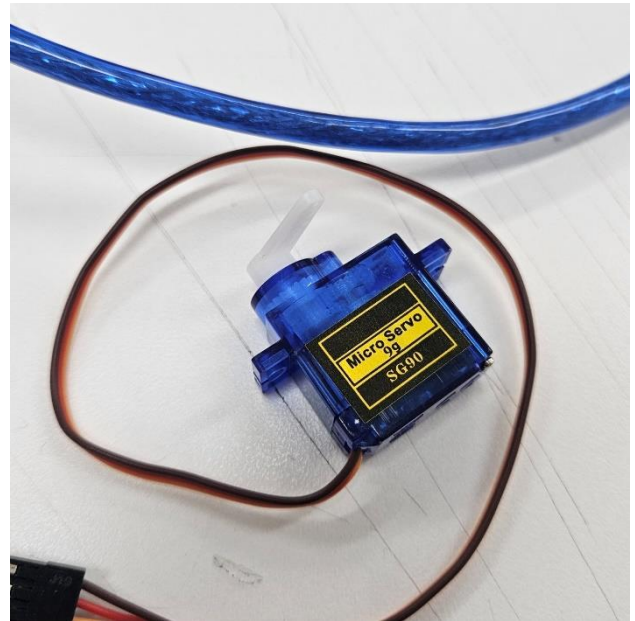
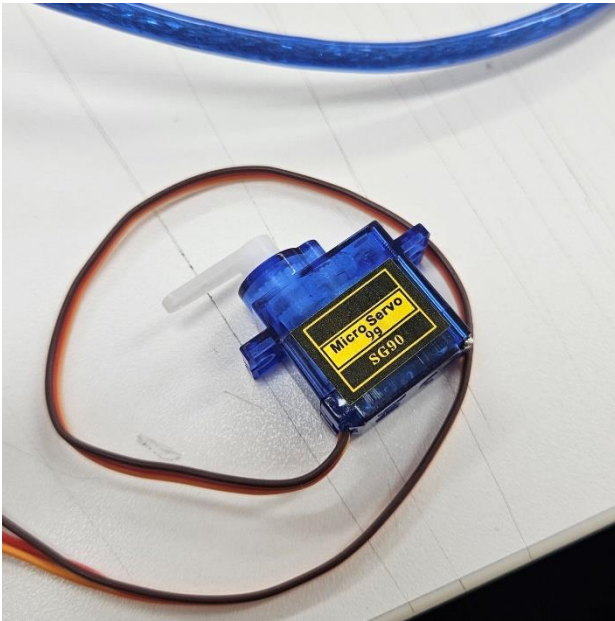
Received motion signal from CAN1. Activating servo motor...
Received motion signal from CAN1. Activating servo motor...
Received motion signal from CAN1. Activating servo motor...
Touch detected! Sending message to CAN1...
Touch detected! Sending message to CAN1...
Touch detected! Sending message to CAN1...
Received motion signal from CAN1. Activating servo motor...
Touch detected! Sending message to CAN1...
Received motion signal from CAN1. Activating servo motor...
Received motion signal from CAN1. Activating servo motor...
Received motion signal from CAN1. Activating servo motor...

- 8X8 매트릭스 LED 제어



CAN2의 터치 센서에 손가락을 대지 않았을 때에는, 왼쪽 사진처럼 LED에 불이 들어오지 않지만, 터치 센서에 손가락을 대면 CAN1에 데이터를 전송(시리얼 모니터로 데이터 오는지 확인)하여 8X8 matrix LED에 불이 들어오는 것을 확인할 수 있었습니다.

- 서보 모터 제어



CAN1의 인체 감지 센서 근처에 가지 않을 경우 서보 모터가 0도에 가만히 있지만, 인체 감지 센서 근처에 손을 대면 센서가 감지해 CAN2에 데이터를 전송(시리얼 모니터로 데이터가 오는지 확인) 합니다. 서보 모터는 0도에서 180도까지 와이퍼와 같이 반복적으로 동작하도록 설계했고 잘 동작함을 확인할 수 있었습니다.

ㄹ. 결과 분석

CAN은 ID간에 우선 순위가 존재하지만 매우 빠른 시간안에 데이터가 전송되므로, 사람의 눈에는 양방향 통신이 가능한 것처럼 보입니다. 이를 이용해 CAN1과 CAN2에서 서로 다른 센서로 데이터를 측정한 뒤에 서로에게 보내, 다양한 모듈을 제어하는 실험을 설계했습니다. 모듈을 동작함에 있어서 미세하게 딜레이가 발생했지만, 전반적으로 실험을 설계 할 때 예상했던 결과대로 동작함을 확인했습니다.

이를 통해, 실제 자동차에 사용되는 CAN 통신이 얼마나 효율적인 통신 방법인지 확인할 수 있었습니다. 자동차에서는 수많은 센서를 통해 실시간으로 데이터를 측정하고, 측정 데이터를 바탕으로 모듈을 제어해야 합니다. 이런 상황에서 CAN 구조 기반의 통신을 적용한다면, 빠르고 정확하게 차체의 많은 ECU들을 제어할 수 있을 것이라고 생각합니다.

3. 고찰

이번 CAN 버스를 이용한 실험을 진행하면서, 이론으로만 알고 있던 CAN 통신의 원리와 동작 구조를 직접 눈으로 확인해 볼 수 있었습니다.

처음 실험에서 오실로스코프를 통해 통신 신호의 파형을 분석할 때, 입력 값이 0인 경우의 CAN_H와 CAN_L의 전압 차에서 이론 값(2V)과 큰 오차가 발생했습니다. CAN끼리의 연결에는 문제가 없다고 판단하였고, 그렇다면 오실로스코프 장비와 프로브의 노후화가 문제라 판단하였습니다. 자리를 옮겨 실험실에서 제일 최신 오실로스코프 측정기가 있는 자리에서 실험을 진행하였고, 프로브 또한 4개를 준비했습니다. 새로운 환경에서 파형을 다시 출력했을 때에는 오차가 크게 감소해 무의미한 범위에 속할 정도였습니다. 미세한 노이즈가 여전히 검출되었지만, 무시할만한 수준이었습니다.

실제 자동차에 쓰이는 CAN 통신에 대한 실습을 진행하면서, 현재 UROP에서 진행하고 있는 배달 로봇 자율 주행 프로젝트에도 CAN 통신을 접목해보면 좋겠다고 생각했습니다. 차체의 휠을 움직이는 모터 드라이브에 있는 CAN 통신 선을 활용해, 카메라로 인식할 수 없는 사각 지대의 장애물을 초음파 센서로 인식하고 메인 CPU에 전송해 장애물을 인식했을 때, 모터 제어를 통해 장애물을 회피하는 방법으로 사용 해야겠다고 생각했습니다. 이를 위해, 통신 속도를 더 빠르게 높이고 CPU에서 이를 처리해 모터를 제어하는 프로그램을 설계할 것입니다.