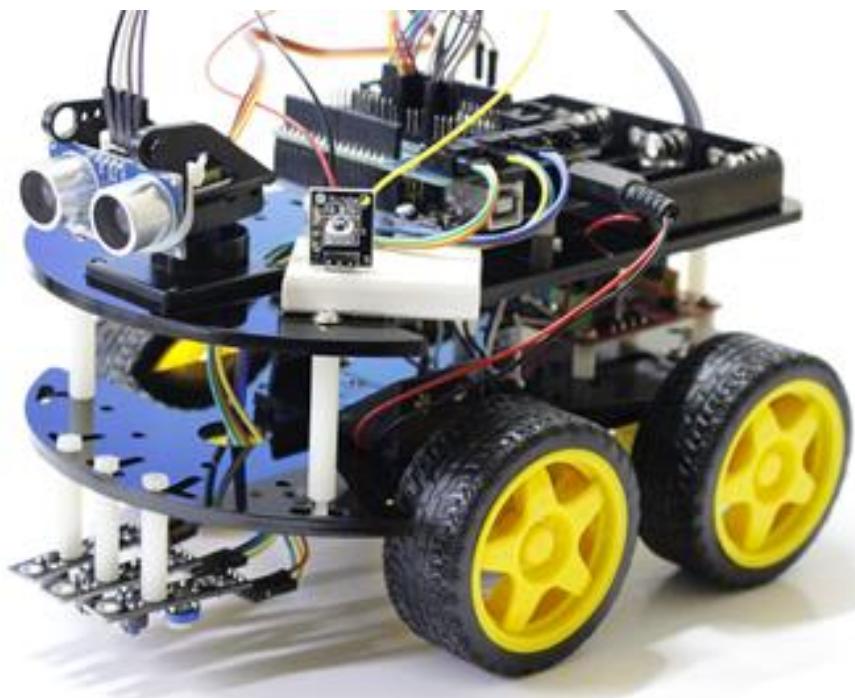


# 아두이노 스마트 로봇 자동차 키트

Make Obstacles Avoiding with Arduino SMART Robot Car



<http://www.gameplusedu.com>  
igameplus.co.,ltd All Rights Reserved.

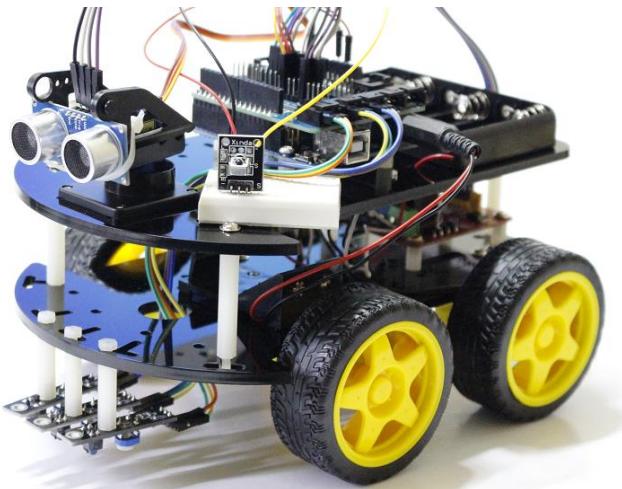
# **Arduino Robot**

## **Smart 4WD Car Kit V3**

**아두이노 로봇**  
**스마트 자동차 4WD 키트 버전 3**

**문서 버전 6.0**

<http://www.gameplusedu.com>  
<http://www.igameplus.com>



## 목차

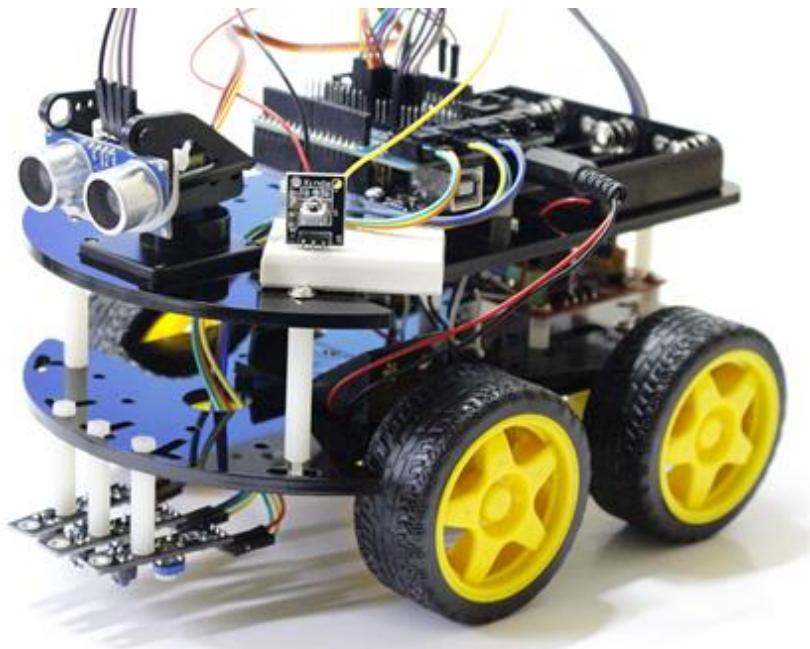
1	스마트 로봇 자동차 키트 소개.....	6
2	스마트 로봇 자동차 기초 기술 요구 사항 .....	7
3	Package included.....	8
4	스마트 로봇 자동차 조립 .....	9
4.1	자동차 하부 조립.....	10
4.1.1	모터와 하판 조립 .....	11
4.1.2	L293N 제어보드와 모터 케이블 연결.....	14
4.2	자동차 상부 조립.....	18
4.2.1	아두이노 우노, 센서 실드 V4, 배터리 박스 .....	19
4.2.2	배터리-제어보드-센서실드 전원 연결 .....	22
4.2.3	제어보드-센서실드 제어포트 .....	24
4.2.4	블루투스 통신 .....	26
4.2.5	스마트폰 블루투스 페어링(Pairing) .....	28
4.3	자동차 주행 .....	29
4.3.1	DC 모터 주행 기본 코드 .....	29
4.3.2	블루투스 제어 주행 코드.....	32
4.3.3	안드로이드 앱 .....	44
5	DC 모터.....	46
5.1	DC 모터 설명 .....	46
5.2	DC 모터 전원 연결 케이블 연결 시 주의 사항 .....	48
5.3	DC 모터 리드선 연결 시 리드선 주의 사항.....	49
5.4	DC 모터 전원 연결선 정리.....	49
6	L298N 듀얼 모터 제어 드라이버 .....	51
6.1	H-Bridge 회로 .....	52
6.2	L298N 모터 제어 보드.....	54
6.3	5V_EN 포트 사용 설명.....	57
6.4	아두이노 우노 R3 & L298N 제어 포트 연결 .....	58
6.5	자동차 속도 및 방향 조절 .....	60
7	초음파 센서와 PAN/TILT 거치대 조립.....	63
7.1	팬, 틸트 거치대 조립 .....	64
7.2	거치대에 서보 모터 결합 .....	66

7.3	초음파 센서 장착 .....	69
8	초음파 센서 프로그래밍 .....	70
8.1	초음파 센서 작동 원리 .....	71
8.2	초음파 센서 연결하기 .....	73
8.3	초음파 센서를 이용한 거리 측정 구현.....	74
8.3.1	예제코드 1 – 스케치 NewPing 라이브러리 사용 .....	74
8.3.2	예제코드 2 – 디렉트 Trig, Echo 사용 .....	75
9	서보 모터 제어 .....	77
9.1	서보모터 연결하기 .....	80
9.2	서보 모터의 중앙 기준 방향 설정 .....	81
9.3	조금 더 안전한 서보모터 제어 .....	85
9.4	서보모터 & 초음파 거리 측정 센서 코드 결합 .....	87
10	Object Avoidance Driving(장애물 감지 운행).....	94
10.1	장애물 감지 방향 전환 .....	94
10.2	편의 기능 구현 .....	96
10.3	운행 모드 동작 QUEUE 개념을 적용해 봅니다 .....	97
10.4	초음파 센서 방향으로 차량 회전 .....	97
10.5	장애물 감지 거리 측정 운행 구현 .....	98
10.6	타이머를 이용한 주기적 초음파 거리 측정 .....	121
11	리모트 컨트롤 운행 제어 .....	123
11.1	IR 신호 수신 .....	123
11.2	리모트 컨트롤러 .....	124
11.3	IR 리모트 라이브러리 .....	125
11.4	IR 리모트 라이브러리 사용시 주의점 .....	125
11.4.1	해결 방법 .....	127
11.5	IR 리모트 센서 연결 .....	128
11.6	리모트 컨트롤러 운행 예제 코드 .....	133
11.7	리모트 수신 모듈 배치(미니브레드보드 사용) .....	142
11.8	센서 실드와 뒤풍 케이블 사용하는 경우 .....	143
11.9	리모트 컨트롤러 운행 코드에 속도제어 추가 .....	144
12	라인 추적 운행 (Line Tracking) .....	155
12.1	라인 추적 운행 소개 .....	155
12.2	라인 추적 운행 준비 .....	155

12.2.1	트랙 라인 설치 .....	155
12.3	라인 트레이서 모듈 기초 정보.....	157
12.3.1	라인센서(TCRT5000)모듈 인식 거리 .....	157
12.4	라인 트레이서 모듈 장착 .....	159
12.5	라인 추적 운행 기초 원리 .....	161
12.6	아두이노 우노 R3 제어 보드와 연결.....	162
12.7	라인 추적 운행 코드.....	165
13	스마트폰과 4WD 스마트 자동차 연동.....	174
13.1	블루투스란.....	174
13.2	아두이노와 스마트폰의 블루투스 통신.....	175
13.3	블루투스 HC-06 슬레이브 모듈'.....	175
13.4	아두이노와 블루투스 모듈 연결.....	177
13.4.1	아두이노 하드웨어 시리얼 포트.....	178
13.4.2	아두이노 소프트웨어 시리얼 통신 .....	180
13.4.3	하드웨어, 소프트웨어 시리얼 통신의 차이점 .....	180
13.5	블루투스 시리얼 데이터 연동 기초 코드.....	181
13.6	아두이노 블루투스 시리얼 통신 예제코드.....	182
13.7	스마트폰과의 페어링.....	183
13.8	안드로이드 앱 설치 .....	184
13.8.1	블루투스 제어 주행 코드.....	187

## 1 스마트 로봇 자동차 키트 소개

아두이노를 활용한 다기능 스마트 로봇 자동차 키트입니다.



아두이노 우노 R3 마이크로 컨트롤러 보드를 사용하여 DC 모터를 사용하는 자동차 프레임과 L298N 모터 제어 보드로 전/후/좌/우 조종 할 수 있습니다.

아두이노 프로그래밍으로 자동차 운행 기능과 키트에서 제공하는 주변 하드웨어 장치를 추가하여 아래와 같은 구현이 가능합니다.

- 초음파 센서에 의한 거리 감지, 장애물 감지 운행
- 서보 모터를 사용한 거리 측정 센서 방향 설정.
- 바닥 라인 트랙 유도선 운행
- 리모트 컨트롤러 제어 운행
- 블루투스 모듈을 장착 스마트폰으로 제어 운행.

4WD 새시 프레임에는 사용되는 모든 부품 고정 훌더가 있으므로 보다 더 견고하고 확실하게 조립하여 사용 할 수 있습니다. 또한 지속적으로 업데이트 되는 예제 코드와 부품 등을 추가하여 더 많은 기능을 구현 할 수 있습니다.

추가로 적외선 거리센서, 블루투스, 지그비, 카메라 등의 모듈들을 부착하여 여러 가지 용도로 사용 가능 합니다.

처음 아두이노를 접하는 경우도 어렵지 않게 원리를 이해하고 조립 할 수 있도록 설명 하였으나 이해되지 않거나 수정할 부분이 있는 경우 본 도서의 기술지원 게시판, 이-메일 등으로 연락 주시기 바랍니다.

## 2 스마트 로봇 자동차 기초 기술 요구 사항

기초 이상의 C/C++ 프로그래밍 능력이 요구됩니다. 스마트 로봇 자동 운행에는 하나의 모듈이 아닌 여러 종류의 하드웨어 모듈을 통합하여 사용합니다. DC 모터 제어, 초음파 센서 거리 측정, 라인 트레이서, 리모트 컨트롤러 수신 코드 구현 및 블루투스 모듈을 사용한 시리얼 통신에 대한 적극적인 이해가 필요합니다. 스마트 로봇 차량 운행을 하기 위해서는 여러 개의 하드웨어 모듈을 제어하기 위한 C/C++ 프로그래밍 능력이 요구됩니다.

- 1) 아두이노 스케치 IDE 기본 사용 기술 습득
- 2) 아두이노 스케치 IDE 라이브러리 적용 방법
- 3) 아두이노 라이브러리 적용 및 사용에 대한 기본 이해
- 4) 4WD 조립에 필요한 부품 연결 및 하드웨어 구성 이해
- 5) 기초 C/C++ 프로그래밍 능력
- 6) 초음파 센서, 서보모터, DC 모터 제어, H-BRIDGE 제어보드 컨트롤
- 7) 모듈 연결 및 응용
- 8) 원격 통신에 대한 이해
- 9) 시리얼 통신에 대한 기초 이해
- 10) 시스템에 소요되는 각종 부품들의 조합 및 연계 코드 구현

### 3 PACKAGE INCLUDED

1 x Arduino UNO328 controller board  
1 x Arduino sensor board V4  
4 x Geared motor  
4 x Tire  
4 x Motor fixing bolt and nuts  
1 x 100 x 213 x 3~5mm Acrylic glass plate  
1 x 100 x 213 x 3~5mm Acrylic glass plate  
1 x L298N motor indicator driver  
1 x Holder kit  
1 x Steering gear  
1 x Ultrasonic Sensor Module  
3 x Line inductive module  
1 x Infrared receiver module  
1 x Remote controller  
1 x Mini breadboard  
1 x Battery holder (6 x AA / not included battery)  
1 x 9v Battery holder with DC Jack  
40 x DuPont Female to Female line  
10 x DuPont Male to Male line  
1 x USB cable B-Type  
11 x Acrylic Copper pillar (3 x 35mm / 2 x 20mm / 6 x 6mm)  
1 x Screws kit  
1 x 9V Battery  
6 x AA Battery  
1x Bluetooth HC-06 Slave Module  
1x HC-11 RF UART Module  
1x Arduino Mega2560 R3

\* 제품에 따라 구성품목이 다를 수 있음.

## 4 스마트 로봇 자동차 조립

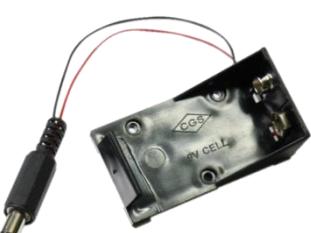
자동차 차체 구성은 아크릴 프레임 상/하 2 개, 4 개의 타이어 & DC 모터, 각종 나사 너트, 배터리 홀더 등이 있습니다. 차체 새시 아크릴에 많은 홀더가 있습니다. 본 키트의 차체 새시는 아두이노 보드, 센서 거치대, L298N 제어 보드, 배터리 상자 등의 고정 홀더들이 있습니다.

아두이노 보드, L298N 제어 보드는 홀더에 6 각 아크릴로 고정대를 만들어 고정시키게 됩니다. 볼트&너트로 고정 합니다.



## 4.1 자동차 하부 조립

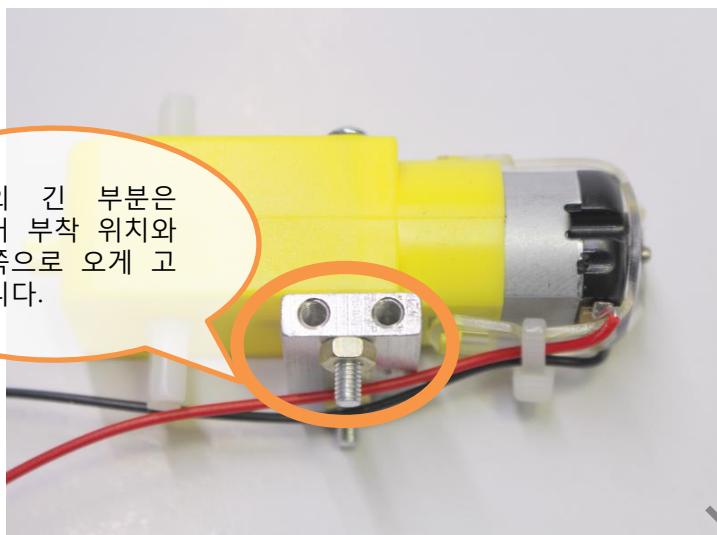
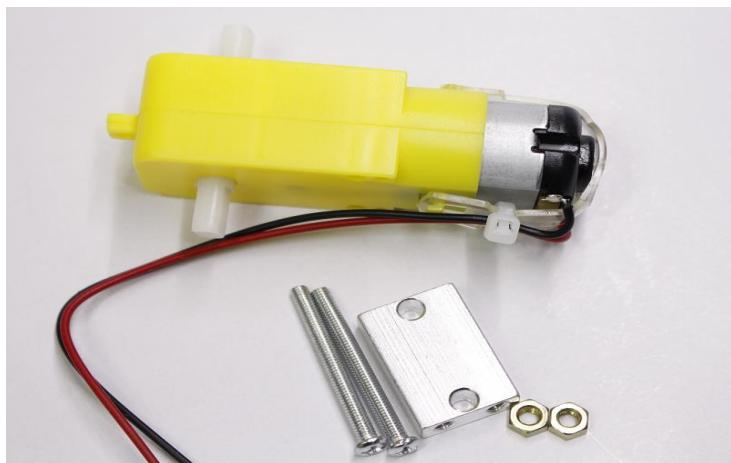
자동차 하부는 DC 모터, 바퀴, L298N 모터 제어보드, 차체 새시는 아크릴 유광 검은색 또는 업그레이드에 의해 검은색 / 빨간색 색상 등으로 되어 있습니다

블랙 새시 하판	DC 모터 고정 볼트/너트	DC 모터 x 4
		
타이어 x 4	L298N 모터 제어보드	제어보드 고정 볼트/너트
		
DC 9 볼트 배터리 박스	상판 하판 고정 볼트/너트	
		

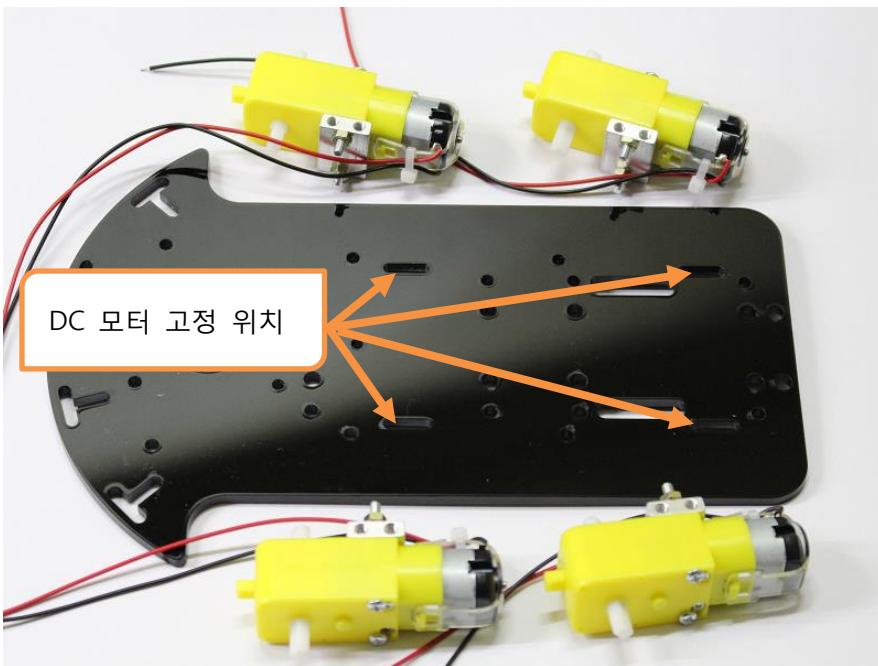
자동차 하부 부품

상/하 쇄시가 있어 조립하는 순서는 하판부터 조립을 하게 됩니다. 편의에 따라 최종 조립 순서는 임의로 변경하여도 무방합니다. 아크릴 차체 쇄시는 굽힘 방지를 위해 투명 비닐 스티커 부착 상태입니다. 제거하고 사용하면 됩니다 상/하 쇄시는 모양이 거의 동일한데, 훌더의 위치가 용도에 따라 다릅니다.

#### 4.1.1 모터와 하판 조립

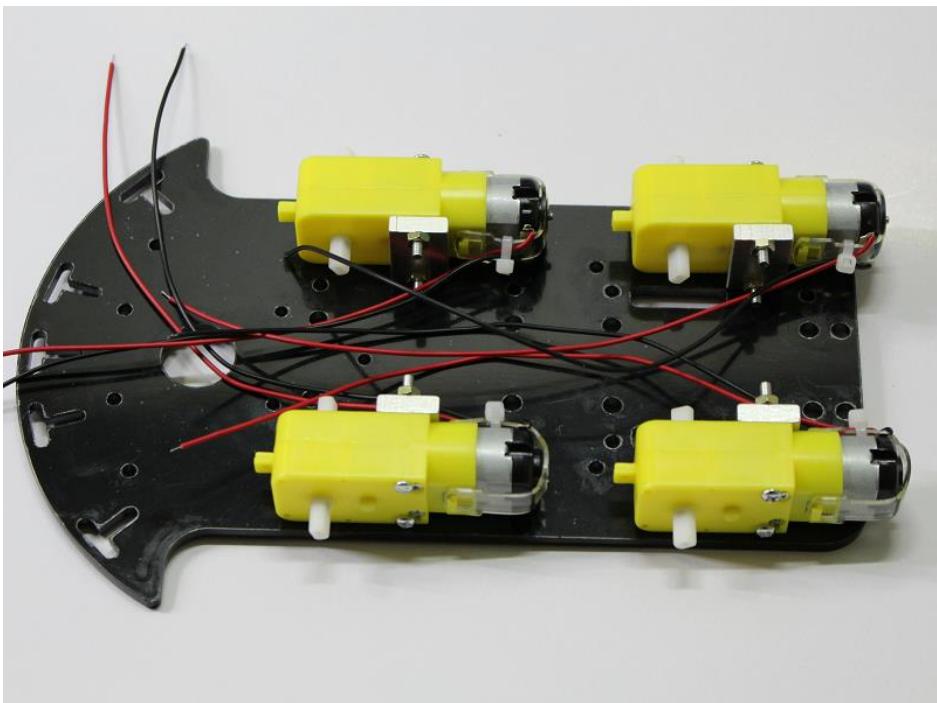


볼트와 너트, 기어모터 고정부품, DC 기어모터입니다. 고정판 틀과 볼트/너트를 연결하여 고정합니다.



DC 모터를 위 그림처럼 배치한 다음 하판 구멍에 맞춰서 볼트로 고정시킵니다.



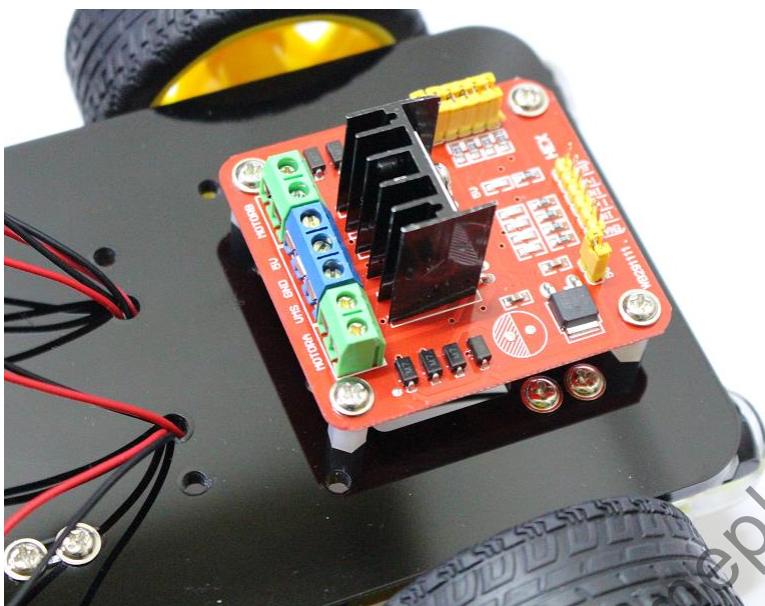
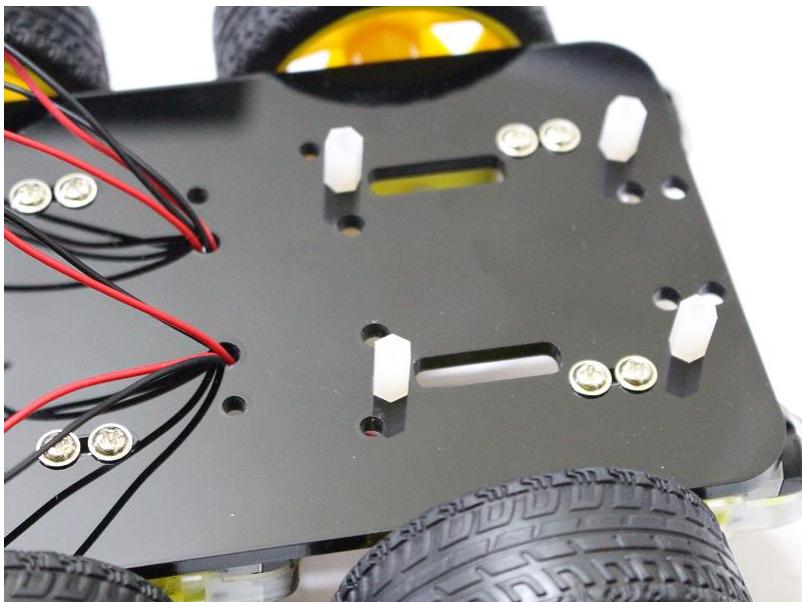


하판에 DC 모터 고정이 끝나면 타이어를 DC 모터 샤프트의 홈 모양에 맞춰 끼워서 바퀴를 조립합니다.

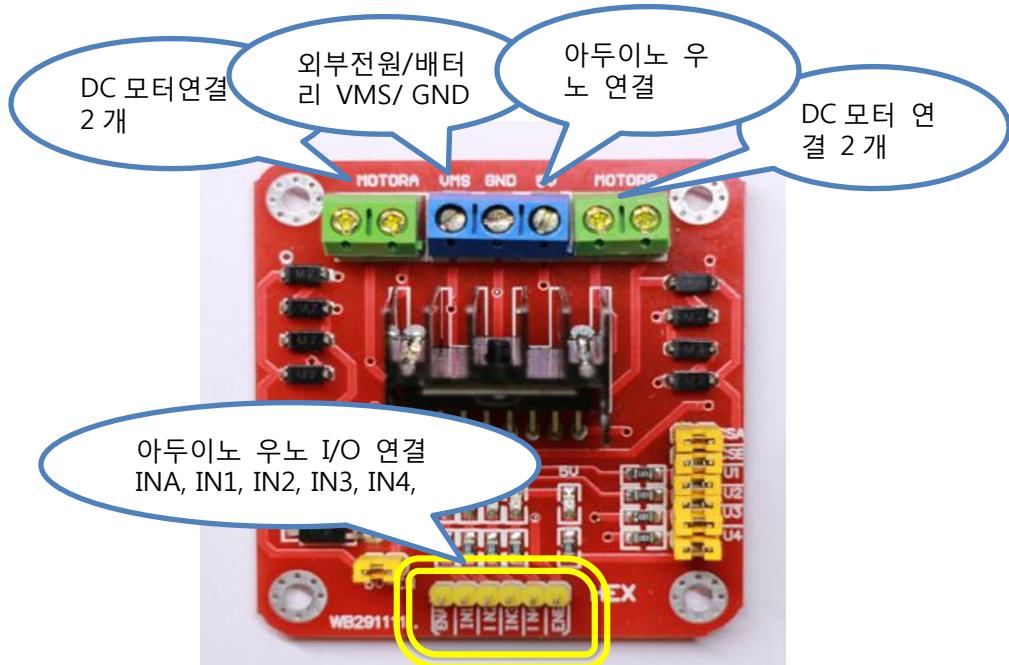


#### 4.1.2 L293N 제어보드와 모터 케이블 연결

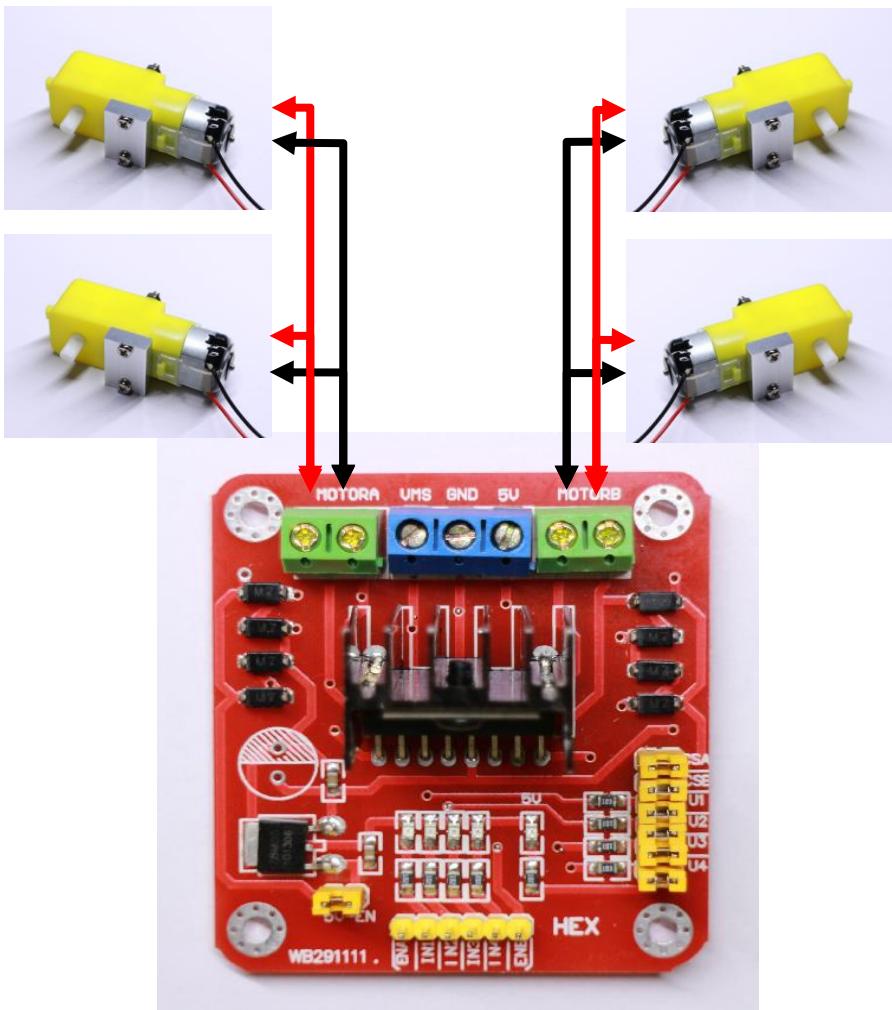
모터 케이블을 같은 방향 모터끼리 구멍을 통해서 위쪽으로 뽑아냅니다. 다음은 L293N 모터제어보드 고정 볼트/너트를 조립하고 L293N 제어보드를 장착합니다.



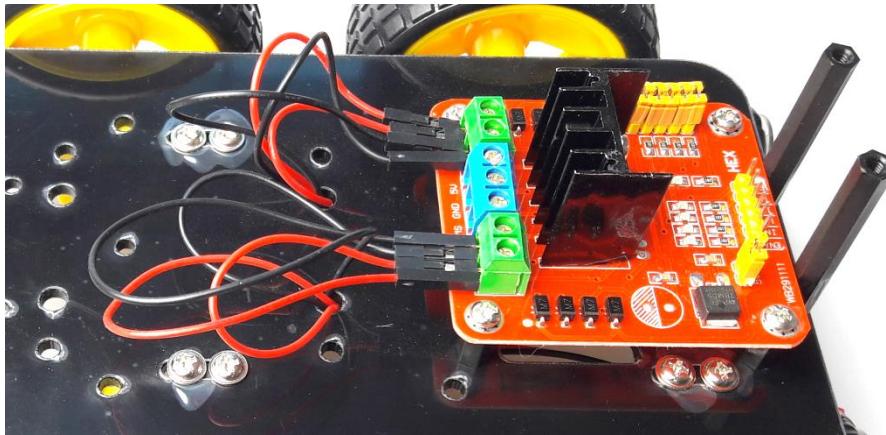
L293N 모터 드라이버 모듈은 DC 모터 방향, 속도를 제어하기 위해 사용하며 한 모듈로 2 개의 모터를 제어할 수 있습니다. 연결방법은 아래와 같습니다.



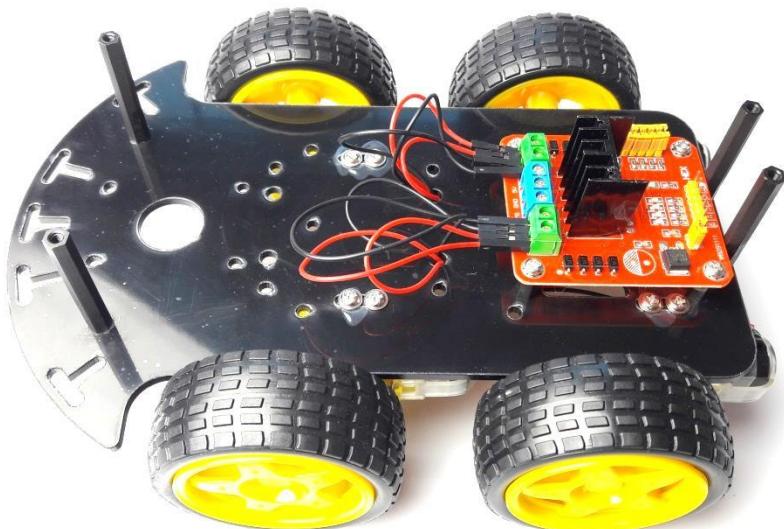
DC 모터는 L298N 제어 보드의 Motor A, Motor B 포트에 각각 2 개씩 연결됩니다. 검정, 빨강 전원 연결선과 DC 모터의 연결 부위 상, 하 연결하면 됩니다. DC 모터에 연결된 빨간색, 검은색 위치를 유의해서 통일되게 L298N 제어 보드에 연결해줍니다.



VMS, GND, 5V 모터와 아두이노 쉴드에 전원 공급을 위해 사용합니다. ENA, IN1, IN2(MotorA), IN3, IN4, ENB(MotorB) 핀은 아두이노와 연결하여 모터를 제어하는데 사용합니다.

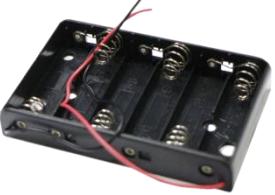


L293N 제어보드에서 MotorA, MotorB 포트 위에 있는 나사를 약간 푼 다음 모터 케이블을 해당 위치에 꼽고 나사를 조여서 케이블이 빠지지 않도록 고정시킵니다. 너무 강하게 조이면 포트부분이 손상될 수 있으니 주의해서 작업합니다.



## 4.2 자동차 상부 조립

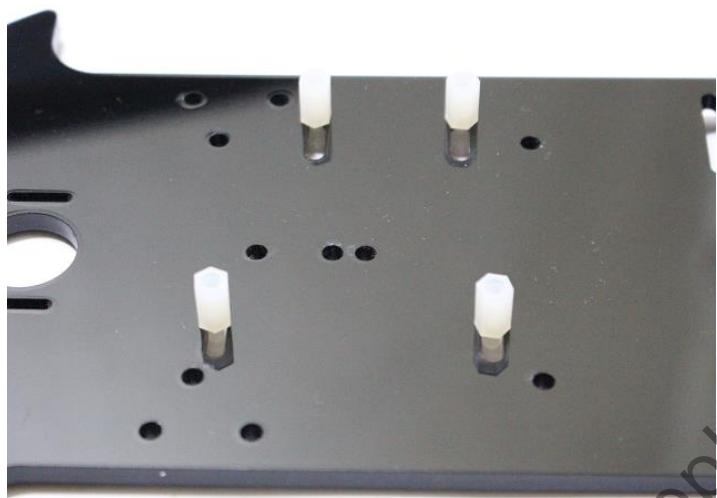
자동차 상부는 블랙 새시 상판, 아두이노 우노보드, 우노 센서실드 V4, 6xAA 배터리 박스(또는 리포 배터리), 블루투스 모듈, 미니 브레드보드로 구성됩니다.

블랙 새시 상판	아두이노 우노 R3	센서 실드 V4
		
6 X AA 배터리 박스	2 Cell 7.4V 리포 배터리	우노보드 고정 볼트/너트
		
블루투스 HC-06	미니 브레드 보드	
		

자동차 상부 부품

#### 4.2.1 아두이노 우노, 센서 실드 V4, 배터리 박스

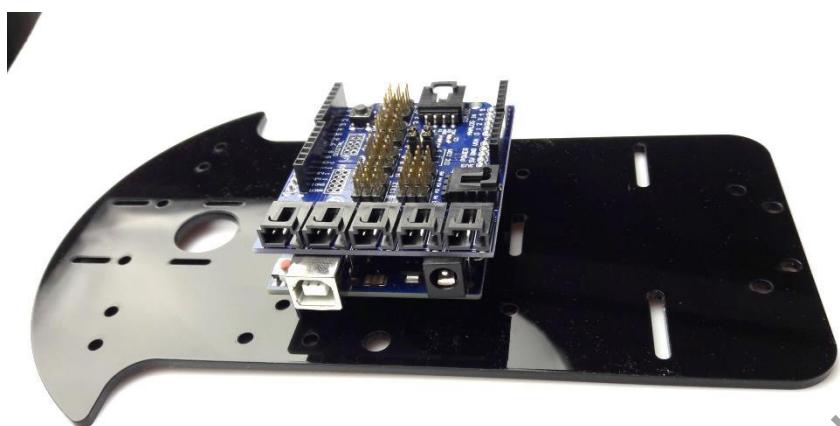
상판에 장착할 부품은 아두이노 우노보드, 센서실드, 배터리 박스입니다. 상판에서 우노보드 고정 볼트를 아래 사진을 참고하여 조립한 다음 아두이노 우노 보드를 장착합니다.





센서실드는 아두이노의 기본 핀을 확장하여 각종 주변 모듈들을 쉽게 우노보드에 연결할 수 있도록 도와주는 실드입니다. 우노보드의 각 핀별로 GND, VCC, Signal 를 추가로 장착하여 브레드보드 없이도 주변 모듈을 직접 연결할 수 있습니다. 스마트 로봇 자동차는 L293N 모터 제어보드 핀들을 센서실드와 연결하여 주행을 제어합니다.

G(GND), V(VCC), S(Signal) : S 핀 배열은 아두이노 핀 배열과 동일합니다.





스마트 로봇 자동차는 하판에 9V 배터리와 상판에 6xAA 배터리 또는 리포배터리를 장착할 수 있으며 상판 배터리만으로도 충분히 주행이 가능합니다. 리포배터리는 재충전이 가능하므로 지속적 사용할 경우 비용을 절감할 수 있습니다. 리포 배터리는 양면 테이프를 이용하여 상판에 고정시킵니다.

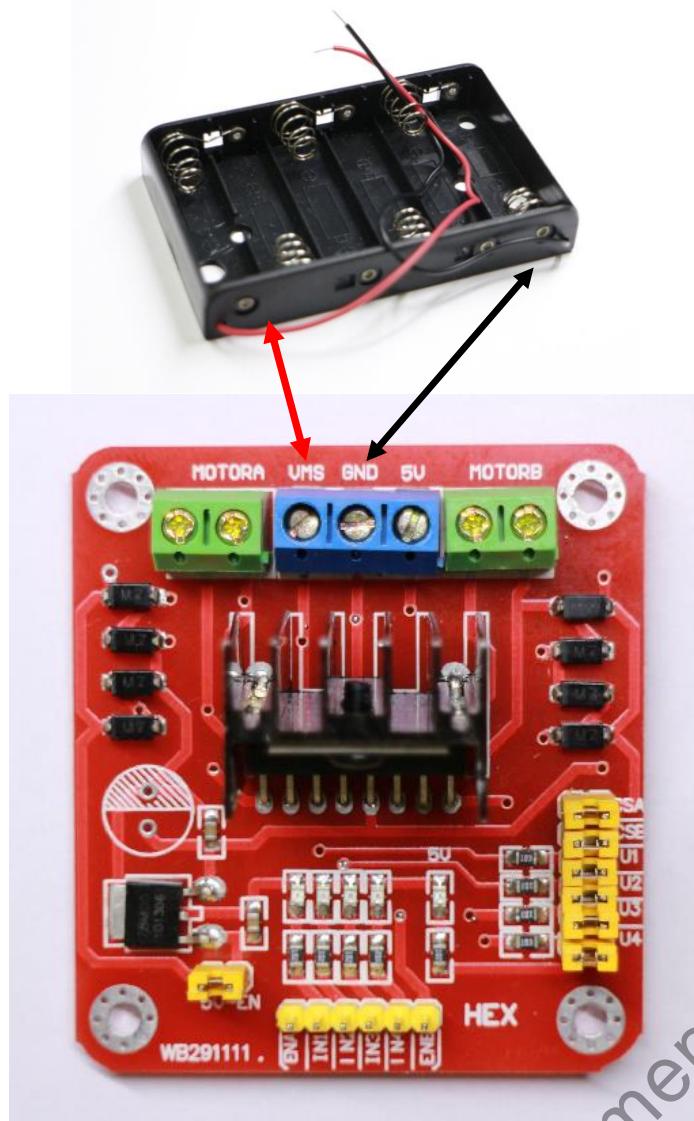
배터리 사용시 '+', '-' 양극이 합선되지 않도록 주의해야 합니다. 합선되면 화재가 발생할 수 있습니다.



#### 4.2.2 배터리-제어보드-센서실드 전원 연결

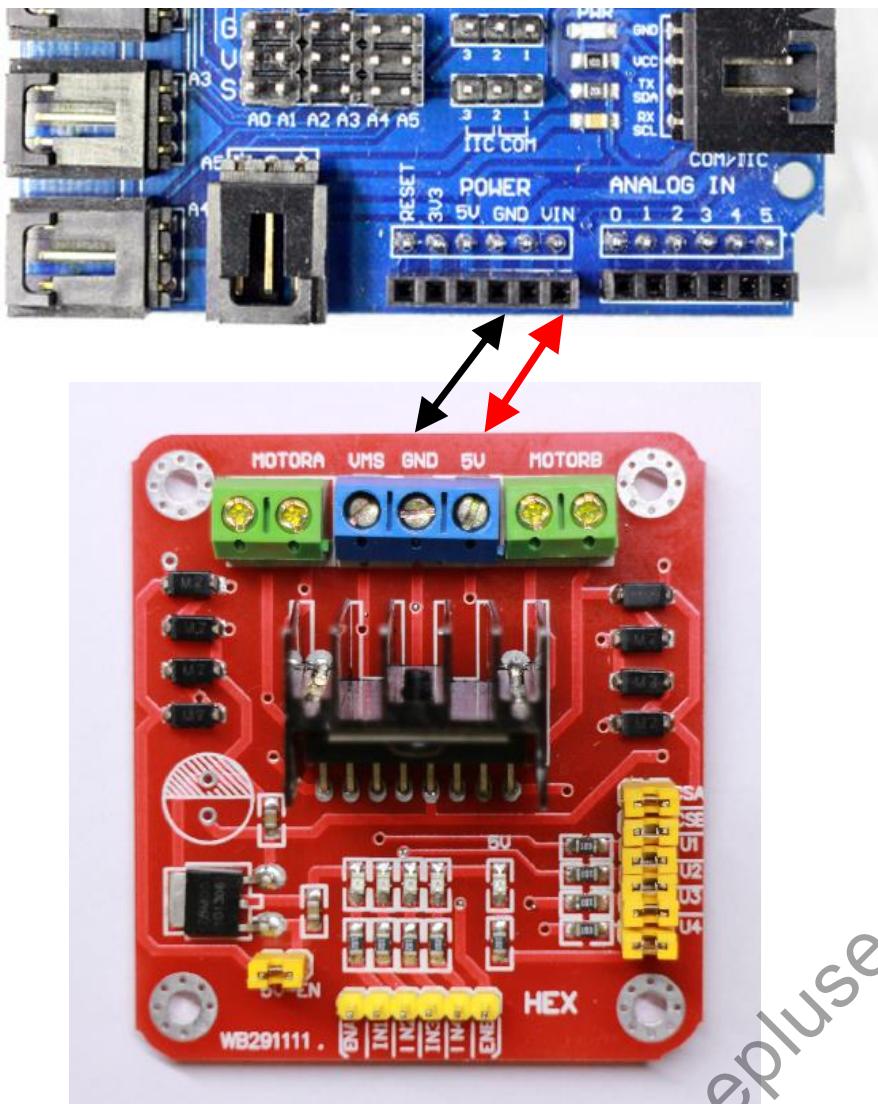
스마트 로봇 자동차의 전원은 배터리 -> 모터제어보드 -> 센서 실드 순으로 연결합니다. 6V 이상 9V 내의 전원이 공급되어야 주행과 제어가 가능합니다.

배터리의 양극(+) 케이블을 VMS, 음극(-)케이블을 GND에 연결하여 제어보드에 전원을 공급합니다. 상판 홀더를 통해서 케이블을 하판 제어보드에 연결합니다.



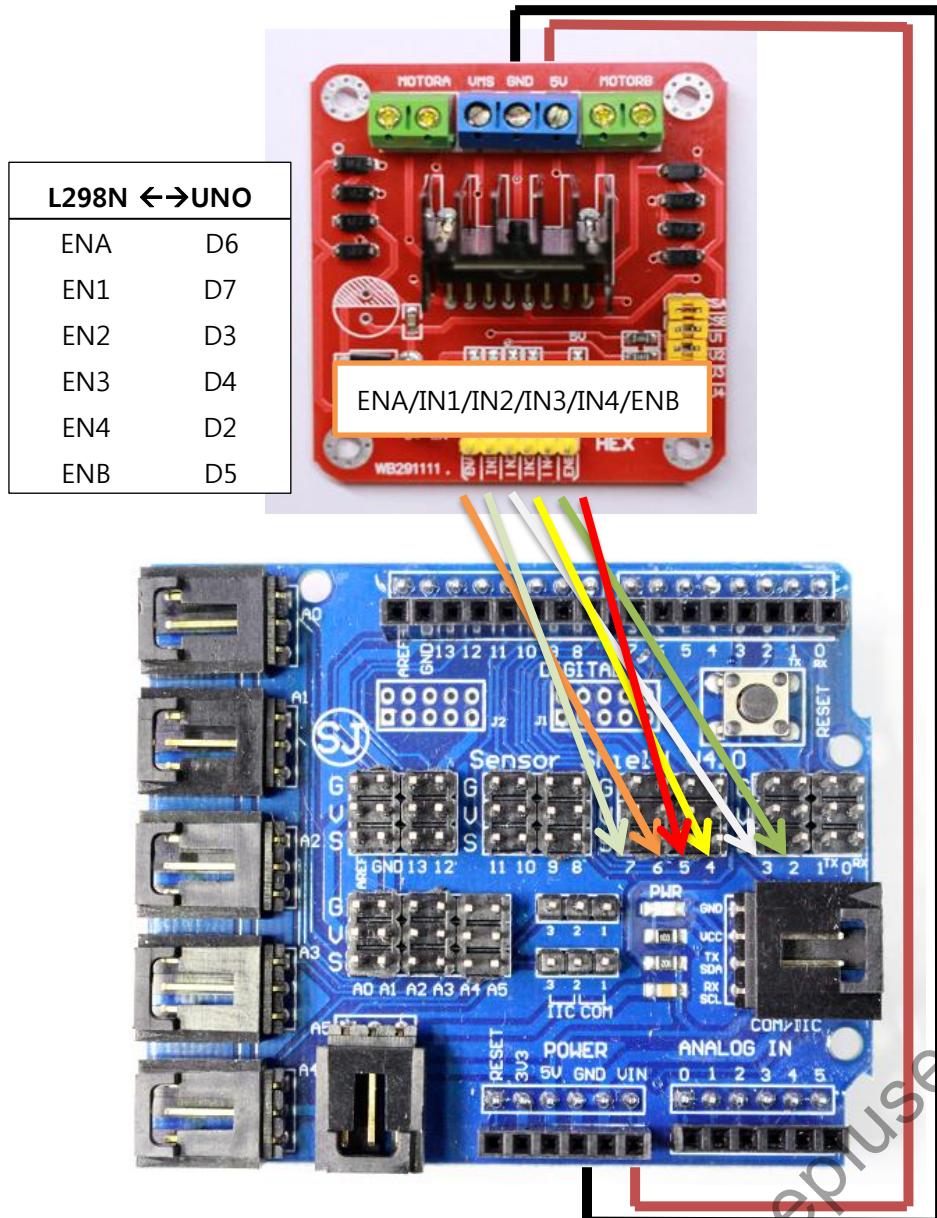
6xAA 배터리 박스를 설치한 경우 박스에 연결된 적색 케이블은 '+', 검정색 케이블은 '-' 입니다. 리포 배터리를 이용하는 경우는 M-M 듀퐁 케이블로 제어보드와 배터리를 연결하는데 반드시 **양극, 음극이 바뀌지 않도록 주의해야 합니다.**

다음은 제어보드와 센서실드를 연결하여 아두이노 보드/센서실드에 전원을 공급합니다. M-M 듀퐁케이블로 GND <-> GND, 5V <-> VIN 을 아래 사진과 같이 연결합니다.

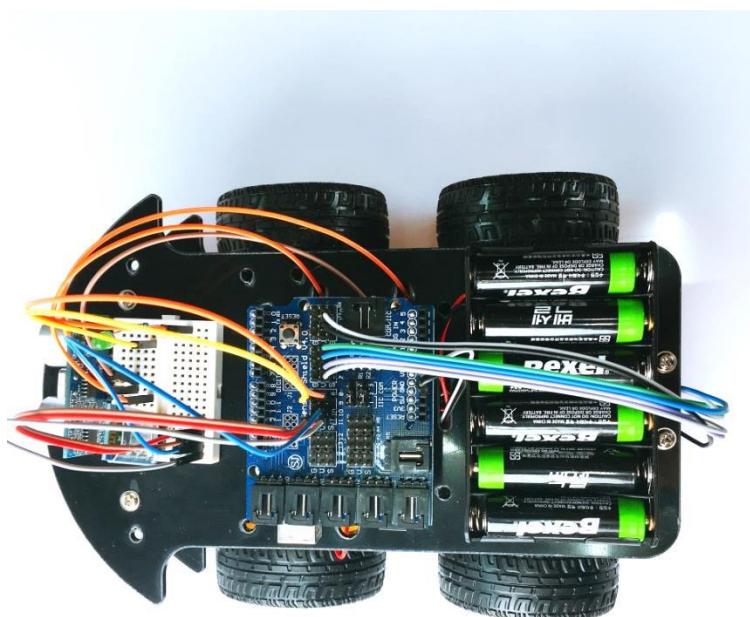
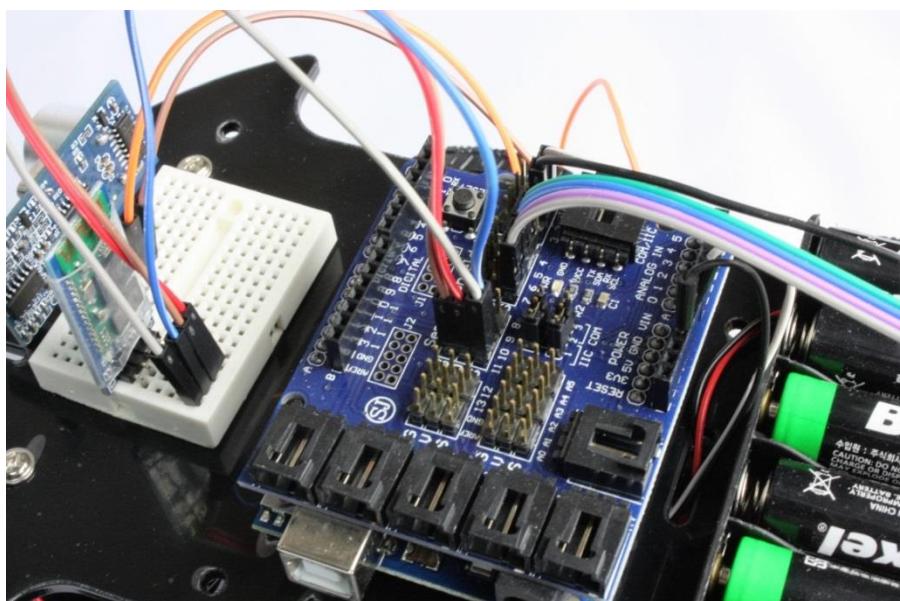


### 4.2.3 제어보드-센서실드 제어포트

자동차의 주행방향, 속도를 제어하기 위해서 모터제어보드의 포트를 센서실드와 연결해야 합니다. 아래 테이블을 참고하여 정확하게 케이블을 연결합니다.



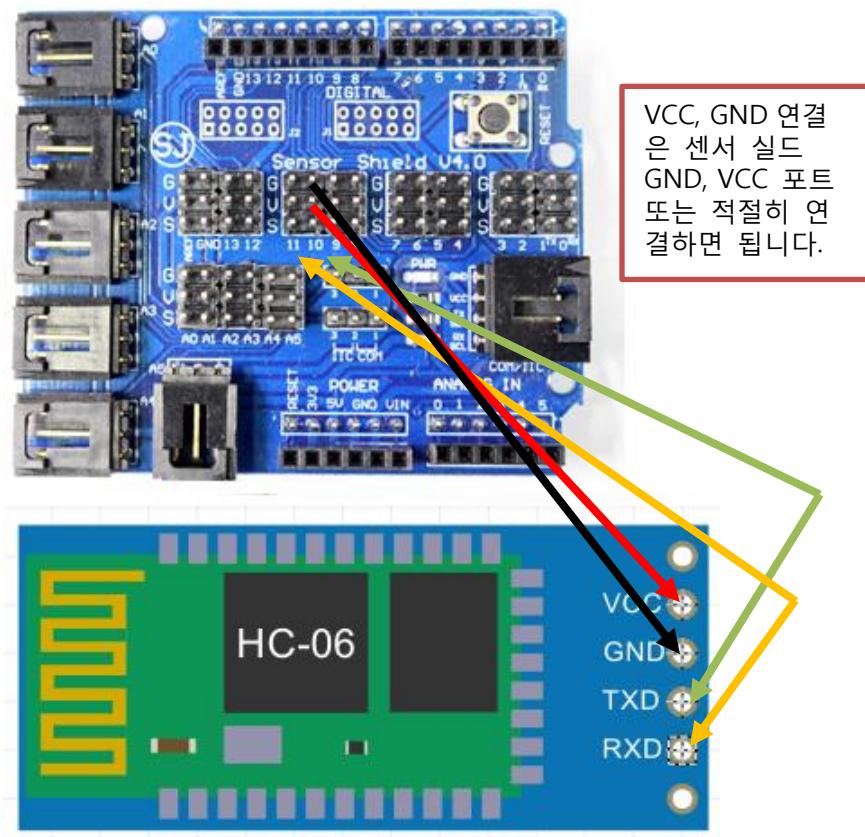
ENA, ENB 는 PWM 지원포트로 연결하여 속도를 조절합니다.



#### 4.2.4 블루투스 통신

스마트폰으로 자동차를 제어하기 위해 블루투스(Bluetooth) 통신을 이용합니다. 블루투스란 근거리 무선 통신기술로 대부분의 휴대폰이 지원하고 있습니다. 스마트 로봇 자동차는 HC-06 블루투스 모듈을 이용하여 스마트폰과 통신을 합니다.

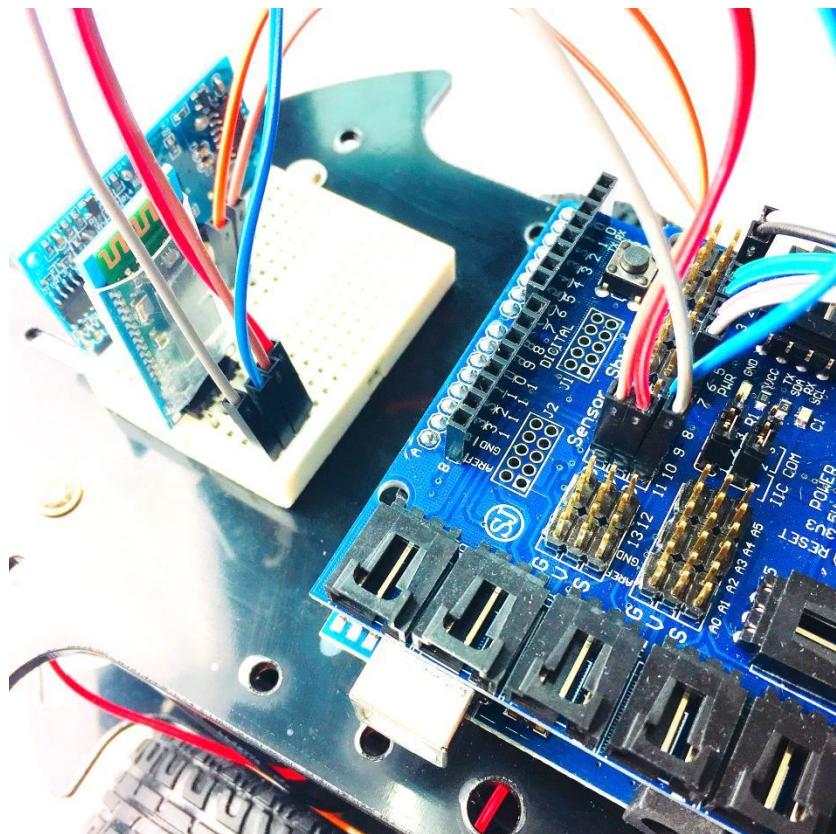
스마트 로봇 자동차는 실드 10(RX), 11(TX) 포트를 블루투스 통신에 사용합니다.



HC-06 블루투스 모듈	아두이노/실드
VCC	5V/V(10)
GND	GND/G(10)
TXD	10(RX)
RXD	11(TX)

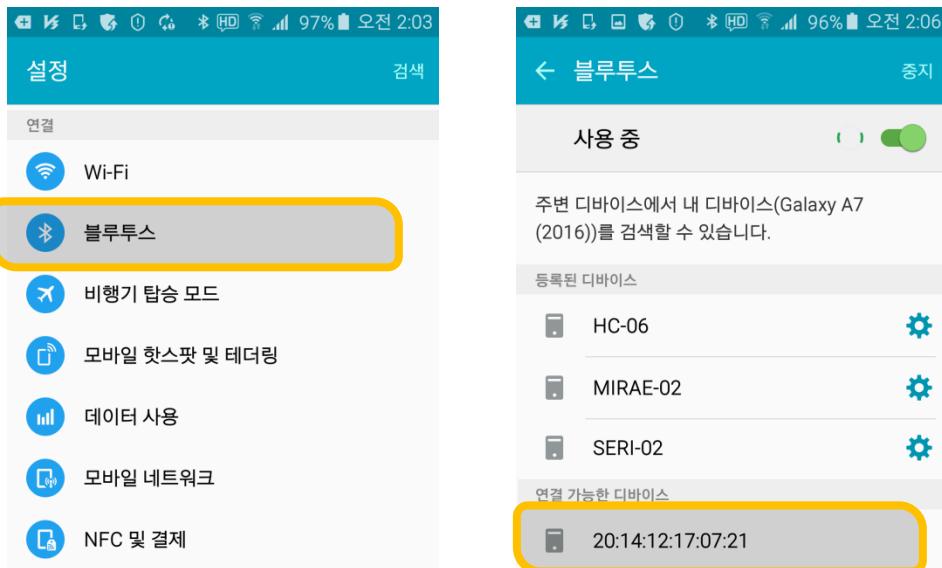
미니 브레드보드를 자동차 상판위에 고정시키고 블루투스 모듈 HC-06 을 설치합니다. 블루투스 모듈에 표시되어 있는 송/수신(TX/RX) 포트를 아두이노 실드에 점퍼케이블로 위에 기술한대로 연결합니다. TXD -> RX(10), RXD <- TX(11).

아두이노 실드의 송/수신 포트는 변경이 가능한데 소스코드도 수정해야 합니다.



#### 4.2.5 스마트폰 블루투스 페어링(Pairing)

스마트폰에서 자동차의 블루투스 모듈에 접속하려면 먼저 블루투스 페어링을 해야 합니다. 스마트 로봇 자동차에 전원을 넣고 블루투스 모듈에 빨간 불이 들어오는지 확인합니다. 정상적으로 블루투스 모듈이 설치되면 빨간불이 깜박입니다. 그 다음은 자동차 주행을 조종할 스마트폰에서 [설정] > [블루투스]로 이동하여 설치한 블루투스 모듈을 등록합니다.



블루투스 디바이스를 등록하면 HC-06으로 표시됩니다. 같은 공간에 있는 모든 블루투스 디바이스가 동일한 이름을 사용하여 나중에 자신의 디바이스 식별이 어렵습니다. HC-06을 자신만의 이름으로 변경하세요.

## 4.3 자동차 주행

자동차 조립을 끝내고 실제 주행에 필요한 펨웨어와 스마트 폰 앱으로 로봇 자동차를 테스트 해 봅니다.

### 4.3.1 DC 모터 주행 기본 코드

전진, 후진, 좌회전, 우회전 코드입니다. 변수 direction 을 원하는 방향 값으로 수정하고 아두이노로 업로드 한 다음 작동 상태를 확인 하세요. 정상작동이 되지 않을 경우는 우선 모터제어 보드와 실드의 포트가 정확히 연결됐는지 점검하기 바랍니다.

```
/*
 * 1: 정방향
 * 2: 좌회전
 * 3: 우회전
 * 4: 후진
 * 0: 정지
 */
int direction = 1; //
int speed = 200; // 최대 속도의 78 % for testing.

//
// 주의: ENA, ENB 는 PWM 지원 포트에 연결한다.
//
#define ENA    6
#define EN1    7
#define EN2    3
#####
#define EN3    4
#define EN4    2
#define ENB    5
```

```
// 부팅 후 1회 실행되는 함수. 초기화 함수. Setup()
void setup()
{
    pinMode(ENA, OUTPUT); // ENA
    pinMode(EN1, OUTPUT); // EN1
    pinMode(EN2, OUTPUT); // EN2

    pinMode(ENB, OUTPUT); // ENB
    pinMode(EN3, OUTPUT); // EN3
    pinMode(EN4, OUTPUT); // EN4

}

// 계속 실행되는 함수. loop()
void loop()
{

    if(direction == 1) // 전진
    {
        digitalWrite(EN1, HIGH);
        digitalWrite(EN2, LOW);
        analogWrite(ENA, speed);

        digitalWrite(EN3, HIGH);
        digitalWrite(EN4, LOW);
        analogWrite(ENB, speed);
    }
    else if(direction == 4) // 후진.
    {
        digitalWrite(EN1, LOW);
        digitalWrite(EN2, HIGH);
    }
}
```

```
analogWrite(ENA, speed);

digitalWrite(EN3, LOW);
digitalWrite(EN4, HIGH);
analogWrite(ENB, speed);

}

else if(direction == 2) // 좌회전
{
    digitalWrite(EN1, LOW);
    digitalWrite(EN2, HIGH);
    analogWrite(ENA, speed);

    digitalWrite(EN3, HIGH);
    digitalWrite(EN4, LOW);
    analogWrite(ENB, speed);

}
else if(direction == 3) // 우회전
{
    digitalWrite(EN1, HIGH);
    digitalWrite(EN2, LOW);
    analogWrite(ENA, speed);

    digitalWrite(EN3, LOW);
    digitalWrite(EN4, HIGH);
    analogWrite(ENB, speed);

}
else if(direction == 0) // 정지.
{
    analogWrite(ENA, 0);
    analogWrite(ENB, 0);
}
}
```

### 4.3.2 블루투스 제어 주행 코드

스마트폰 앱과 블루투스 통신을 통하여 로봇 자동차를 조종하는 주행 코드입니다. 다음 스케치를 컴파일하고 아두이노 우노 보드로 업로드 합니다.

```
/*
 * Smart Robot Car V3
 * - Bluetooth(HC-06) control version
 * - Android app provided
 */

#include <SoftwareSerial.h>

///////////
// <BT>      <UNO>
// TX <----> RX
// RX <----> TX
///////////
SoftwareSerial btSerial(10, 11); // RX, TX(UNO)

///////////
// Note: ENA and ENB must be connected to PWD supported pins
//
#define ENA 6 // PWD
#define EN1 7
#define EN2 3

#define EN3 4
#define EN4 2
#define ENB 5 // PWD
```

```
//////////  
// Ultrasonic sensor  
//////////  
int TRIG_pin = 12; // 센서 Trig 핀, D12  
int ECHO_pin = 13; // 센서 Echo 핀, D13  
  
#define blinkLED 8 // for crash warning  
  
//////////  
// Car direction  
//  
#define CAR_DIR_FW 0 // forward  
#define CAR_DIR_BK 1 // backward  
#define CAR_DIR_LT 2 // left turn  
#define CAR_DIR_RT 3 // right turn  
#define CAR_DIR_ST 4 // stop  
  
//////////  
// Default direction and speed  
//  
int g_carDirection = CAR_DIR_ST;  
int g_carSpeed = 230; // 60% of max speed for testing  
  
//////////  
// Note : confirm HIGH/LOW for correct movement  
//  
void car_forward()  
{  
    digitalWrite(EN1, HIGH);  
    digitalWrite(EN2, LOW);  
    analogWrite(ENA, g_carSpeed);  
  
    digitalWrite(EN3, HIGH);
```

```
digitalWrite(EN4, LOW);
analogWrite(ENB, g_carSpeed);
}

void car_backward()
{
    digitalWrite(EN1, LOW);
    digitalWrite(EN2, HIGH);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, LOW);
    digitalWrite(EN4, HIGH);
    analogWrite(ENB, g_carSpeed);
}

void car_left()
{
    digitalWrite(EN1, LOW);
    digitalWrite(EN2, HIGH);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, HIGH);
    digitalWrite(EN4, LOW);
    analogWrite(ENB, g_carSpeed);
}

void car_right()
{
    digitalWrite(EN1, HIGH);
    digitalWrite(EN2, LOW);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, LOW);
}
```

```
digitalWrite(EN4, HIGH);
analogWrite(ENB, g_carSpeed);
}

void car_stop()
{
    analogWrite(ENA, 0);
    analogWrite(ENB, 0);
}

///////////////////////////////
// Execute car moving
//
void update_Car()
{
    switch ( g_carDirection ) {
        case CAR_DIR_FW:
            car_forward();
            break;
        case CAR_DIR_BK:
            car_backward();
            break;
        case CAR_DIR_LT:
            car_left();
            break;
        case CAR_DIR_RT:
            car_right();
            break;
        case CAR_DIR_ST:
            car_stop();
            break;
        default :
            ;
    }
}
```

```
    }

    return;
}

///////////////////////////////
// Class - Serial Protocol
//
class _CommProtocol
{
private:
    unsigned char protocolPool[28];
    int bufPoint;

public:
    _CommProtocol()
    {

    }

    void addPool(unsigned char cByte)
    {
        if (bufPoint < 28)
        {
            if (bufPoint == 0 and cByte != 0x0c)
                return; // invalid code

            protocolPool[bufPoint++]=cByte;
            //Serial.print("bufPoint -> ");
            //Serial.println(bufPoint);
        }
    }

    void clearPool()
```

```

{
    bufPoint = 0;
    memset(protocolPool, 0x00, 28);
    Serial.println("clearPool");
}

bool isValidPool()
{
    if (bufPoint >= 28)
    {
        //Serial.print("protocol length : ");

        if (protocolPool[0] == 0x0c && protocolPool[14] == 0x0c)
        {
            //Serial.println(protocolPool.length());
            return true;
        }
        else
        {
            clearPool();
            Serial.println("isValidPool 28 OVER");
        }
    }
    return false;
}

unsigned char getMotorLValue()
{
    unsigned char szProto[14];
    memcpy(szProto, protocolPool, 14);
    if (szProto[0] == 0x0C &&
        szProto[1] == 0x00 &&
        szProto[2] == 0x80 &&

```

```

        szProto[3] == 0x04 &&
        szProto[4] == 0x02)
    {
        unsigned char l = szProto[5];// -0x32;
        return l;
    }
    return 0x00;
}

unsigned char getMotorRValue()
{
    unsigned char szProto[14];
    memcpy(szProto, &protocolPool[14], 14);
    if (szProto[0] == 0x0C &&
        szProto[1] == 0x00 &&
        szProto[2] == 0x80 &&
        szProto[3] == 0x04 &&
        szProto[4] == 0x01)
    {
        unsigned char l = szProto[5];// -0x32;
        return l;
    }
    return 0x00;
}
}; // class(_CommProtocol)

///////////////////////////////
// Create an instance of class(_CommProtocol)
//
_CommProtocol SerialCommData;

///////////////////////////////
// Parse the serial input value and convert to MOVE command

```

```

//  

void process_SerialCommModule()  

{  

    if (SerialCommData.isValidPool())  

    {  

        char motorLR[2];  
  

        motorLR[0] = (char)SerialCommData.getMotorLValue();  

        motorLR[1] = (char)SerialCommData.getMotorRValue();  

        SerialCommData.clearPool();  
  

        //  

        Serial.print("Left [");  

        Serial.print(motorLR[0],DEC);  

        Serial.print("] Right [");  

        Serial.print(motorLR[1],DEC);  

        Serial.println("]");  

        //  
  

        char szCmdValue = '5';  

        // set MOVE commands  

        if (motorLR[0] == 0 && motorLR[1] == 0) { // (0,0) stop  

            szCmdValue = '5';  

        }  

        else  

        {  

            int nSpeed;  

            nSpeed = max(abs(motorLR[0]), abs(motorLR[1]));  
  

            // Set direction  

            if (motorLR[0] > 0 && motorLR[1] > 0) // (+,+) forward  

            {  

                szCmdValue = '2';

```

```

        g_carSpeed = 255.0f * ((float)nSpeed / 100.0f);
    }
    else if (motorLR[0] < 0 && motorLR[1] < 0) // (-,-) backward
    {
        szCmdValue = '8';
        g_carSpeed = 255.0f * ((float)nSpeed / 100.0f);
    }
    else if (motorLR[0] < 0 && motorLR[1] > 0) // (-,+) left turn
    {
        szCmdValue = '4';
        g_carSpeed = 255.0f * ((float)((float)nSpeed*1.66f) / 100.0f);
    }
    else if (motorLR[0] > 0 && motorLR[1] < 0) // (+,-) right turn
    {
        szCmdValue = '6';
        g_carSpeed = 255.0f * ((float)((float)nSpeed*1.66f) / 100.0f);
    }
}
//
Serial.print("speed ");
Serial.print(g_carSpeed);
Serial.print(" ");
Serial.println(szCmdValue);
//
// Set the direction and speed with command
controlByCommand(szCmdValue);
}

///////////
// Hint : Command codes come from keypad numbers
//

```

```
void controlByCommand(char doCommand)
{
    switch ( doCommand ) {
        case '+': // speed up
            g_carSpeed += 20;
            g_carSpeed = min(g_carSpeed, 255);
            break;
        case '-': // speed down
            g_carSpeed -= 20;
            g_carSpeed = max(g_carSpeed, 75);
            break;
        case '2': // forward
            g_carDirection = CAR_DIR_FW;
            break;
        case '5': // stop
            g_carDirection = CAR_DIR_ST;
            break;
        case '8': // backward
            g_carDirection = CAR_DIR_BK;
            break;
        case '4': // left
            g_carDirection = CAR_DIR_LT;
            break;
        case '6': // right
            g_carDirection = CAR_DIR_RT;
            break;
        default :
            ;
    }
    return;
}
```

```
//////////
```

```
// Setup to run once
//
void setup()
{
    Serial.begin(9600);      // PC serial monitor debugging
    btSerial.begin(9600);    // bluetooth serial connection

    //init car control board
    pinMode(ENA, OUTPUT);  // ENA
    pinMode(EN1, OUTPUT);  // EN1
    pinMode(EN2, OUTPUT);  // EN2

    pinMode(ENB, OUTPUT);  // ENB
    pinMode(EN3, OUTPUT);  // EN3
    pinMode(EN4, OUTPUT);  // EN4

    pinMode(blinkLED, OUTPUT); // for crash check
    pinMode(TRIG_pin, OUTPUT);
    pinMode(ECHO_pin, INPUT);

    //
    Serial.print("direction value ");
    Serial.println(g_carDirection);
    Serial.print("speed pwm value ");
    Serial.print(g_carSpeed);
    Serial.println("");
    //
}

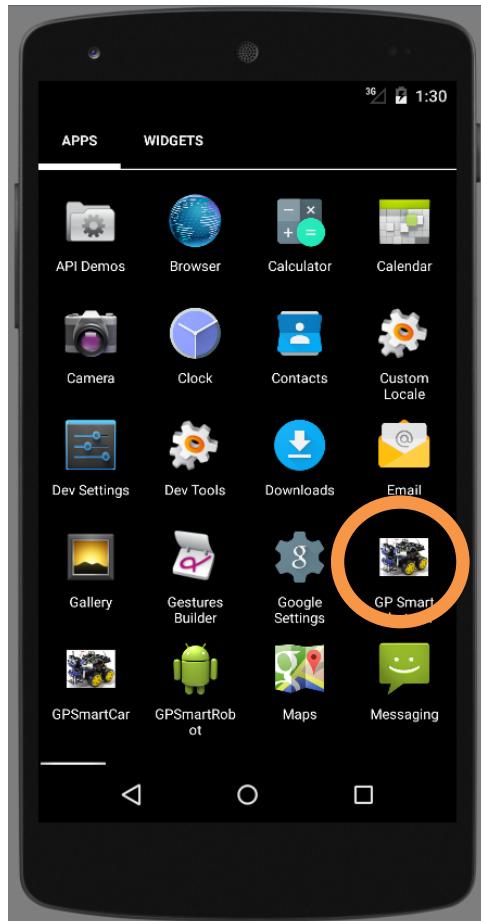
///////////////////////////////
// main code to run repeatedly
//
void loop()
```

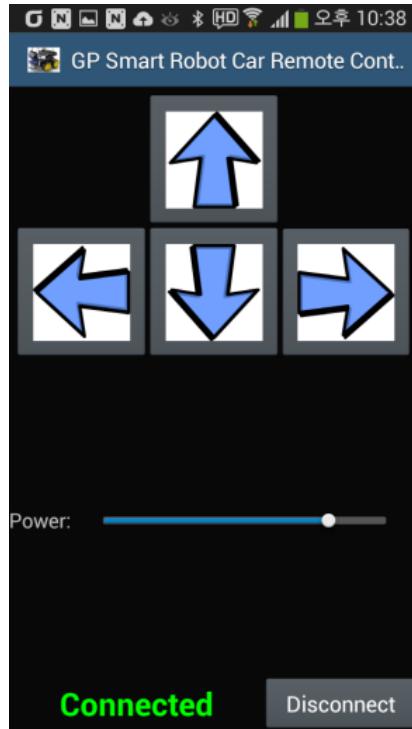
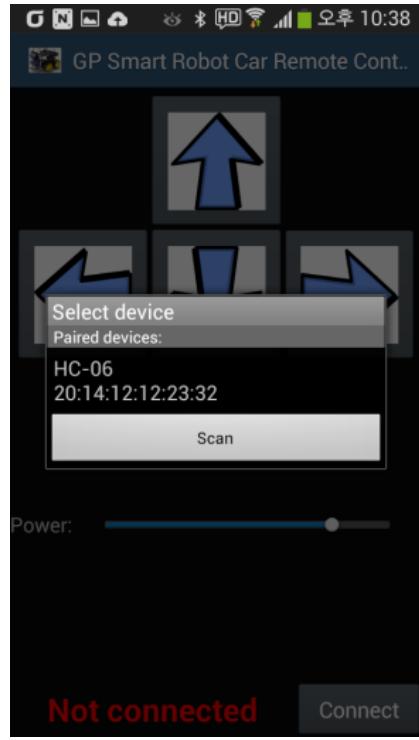
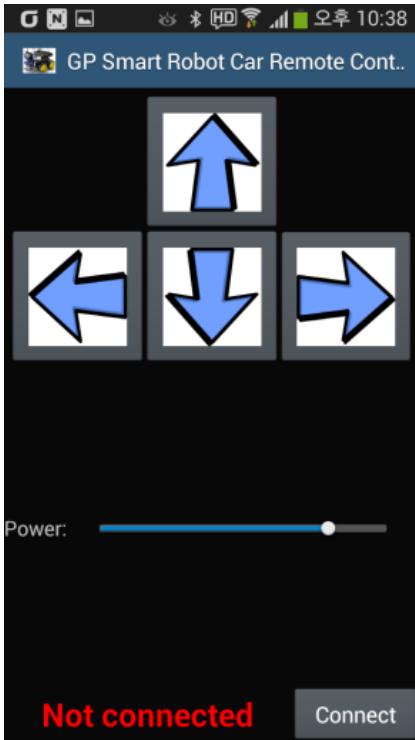
```
{  
  
    if (btSerial.available()) {  
  
        unsigned char cByte;  
  
        cByte = btSerial.read();  
  
        SerialCommData.addPool(cByte); // store the serial input to Buffer  
  
        process_SerialCommModule(); // parse and change the input value  
                                // to MOVE command  
        update_Car();          // execute car MOVE  
  
    }  
  
}
```

### 4.3.3 안드로이드 앱

로봇 자동차 주행 제어에 안드로이드 앱을 이용할 수 있습니다. 게임플러스에듀 사이트([www.gameplusedu.com](http://www.gameplusedu.com)) 게시판에서 앱을 다운받아 설치하세요.

[http://www.gameplusedu.com/pds/gpshop/arduino/robotics/kit\\_example/car\\_4wd/android\\_apk/GPSmartCarRemoteManager.apk](http://www.gameplusedu.com/pds/gpshop/arduino/robotics/kit_example/car_4wd/android_apk/GPSmartCarRemoteManager.apk)





## 5 DC 모터

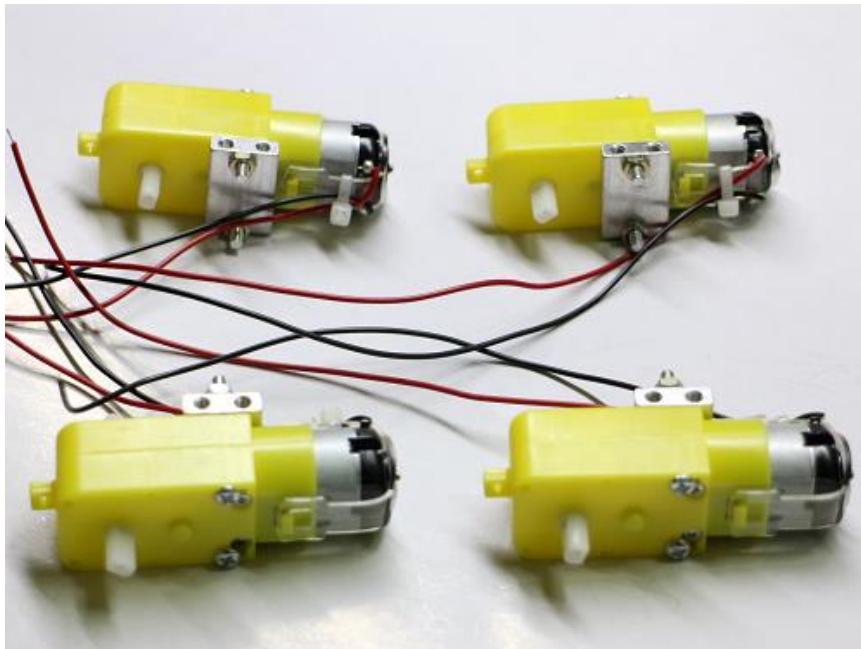
DC 모터 샤프트에는 타이어가 물려 있습니다. 운행하기 위해서는 모터 회전 및 제어를 해야 합니다. DC 모터만 제대로 동작 시키면 자동차 관련 로봇은 쉽게 제작 할 수 있습니다.

### 5.1 DC 모터 설명

본 키트의 DC 모터 작동은 L298N Dual H-Bridge 드라이버 모듈로 제어 합니다.

DC 모터는 (+) (-) 연결해 주면 회전하는 부품입니다. DC 모터는 (+)(-) 연결 극성 변경, 전압 조절에 의해 회전 방향 및 속도 변경할 수 있습니다.

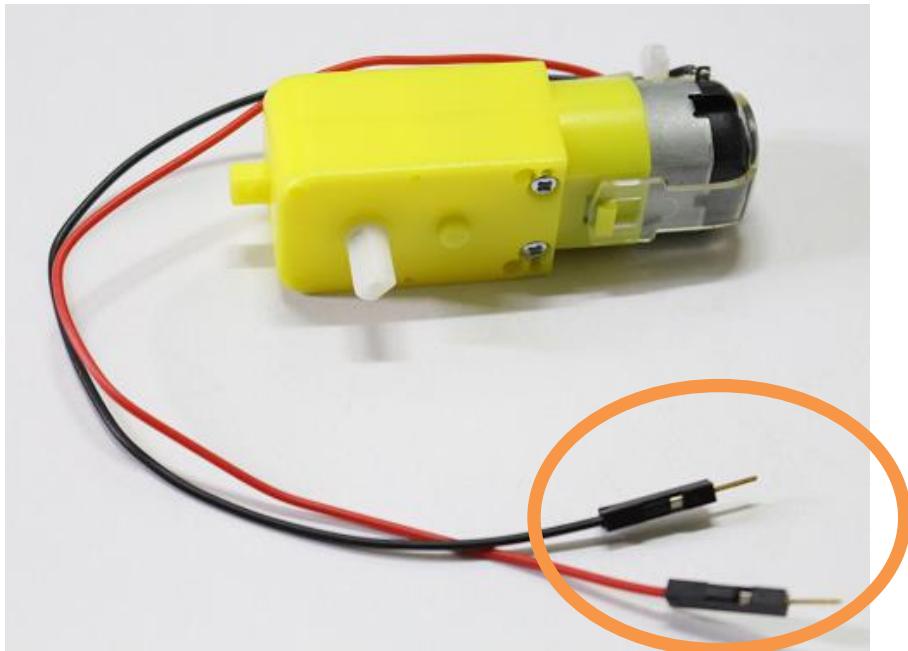
키트에 사용되는 DC 기어 모터입니다.



3V ~ 6V 사용 DC 모터입니다. 권장 가동 전압은 4~5V 정도입니다.

DC 모터의 전원 연결선은 2.54 피치 헤더 핀 형태입니다.

L298N 모터 제어 보드, 브레드보드 등에 연결하기 용이한 형태입니다.



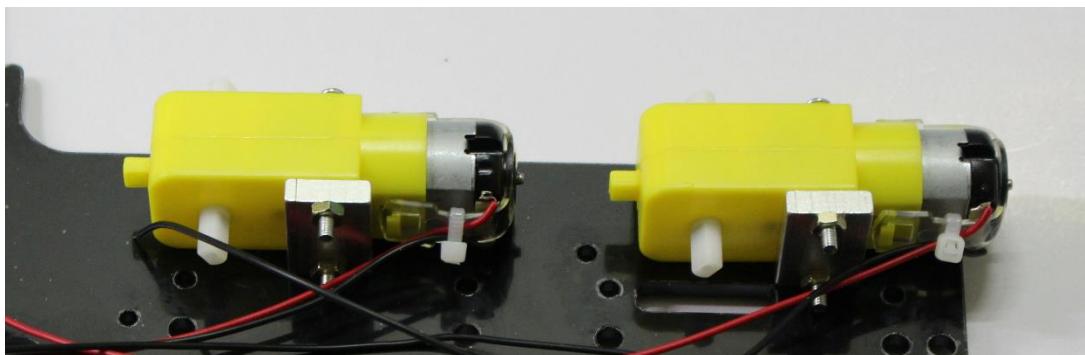
working voltage	DC 3V	DC 5V	DC 6V
working current	100mA	100mA	120mA
reduced ratio	48:1		
non-load	100rpm	190rpm	240rpm
wheel diameter	615 px		
non-speed	20m/min	39m/min	48m/min
weight	50g		
Size	70mm*22mm*18mm		
noise	< 65db		

전압(V)의 크기에 따라 회전 속도의 변경이 가능합니다. 회전 속도 변경으로 운행 속도 조절이 가능합니다. 물론 배터리와 DC 모터의 최대 RPM 을 기준으로 최고속도이고 그 이하로 속도 조절이 가능합니다.

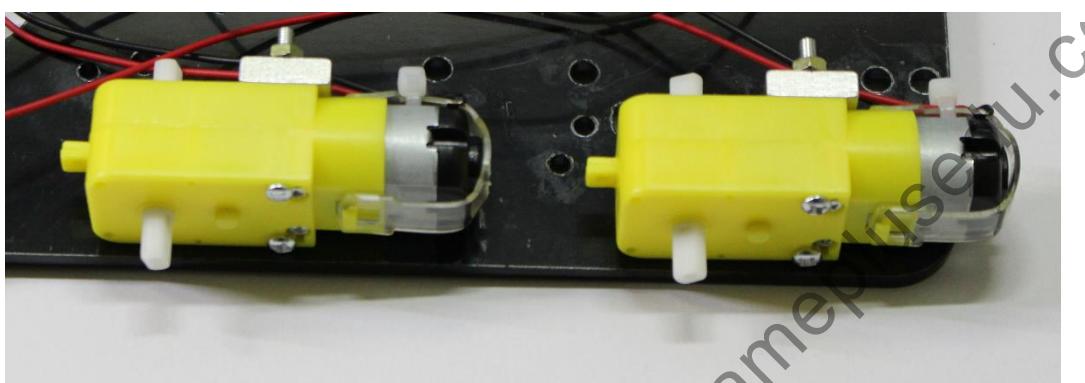
## 5.2 DC 모터 전원 연결 케이블 연결 시 주의 사항

DC 모터의 2 개 전원 연결선은 극성을 구분하지 않습니다. 무조건 가동 전압 충족 되면 돌아가는 부품입니다. 전원 연결선의 (+)(-)의 방향에 따라 회전 방향이 반전됩니다. 키트에 포함된 DC 모터의 입고 버전에 따라 검은색, 빨간색 전원 연결선의 위치가 다른 경우가 종종 있습니다.

그리고 DC 모터 연결된 검은색/빨간색(또는 검은색/흰색) 전선의 납땜 연결이 끊어질 수 있습니다. 끊어짐 방지를 위한 대비책으로 케이블 타이로 묶었지만, 조립 과정 중에 끊어질 수 있습니다. 약간의 주의가 필요합니다. 만약 끊어진 경우 인두기가 있다면 다시 연결하여 사용하면 됩니다. 또는 적절한 조치(절연 테이프로 임시로 붙여서 묶어주는 식)를 취하시기 바랍니다. 아래의 이미지처럼 빨간색 전원 연결선은 위쪽에 있습니다.



반대편 DC 모터는 부착 후 검은색 전원선이 위쪽으로 있게 됩니다.

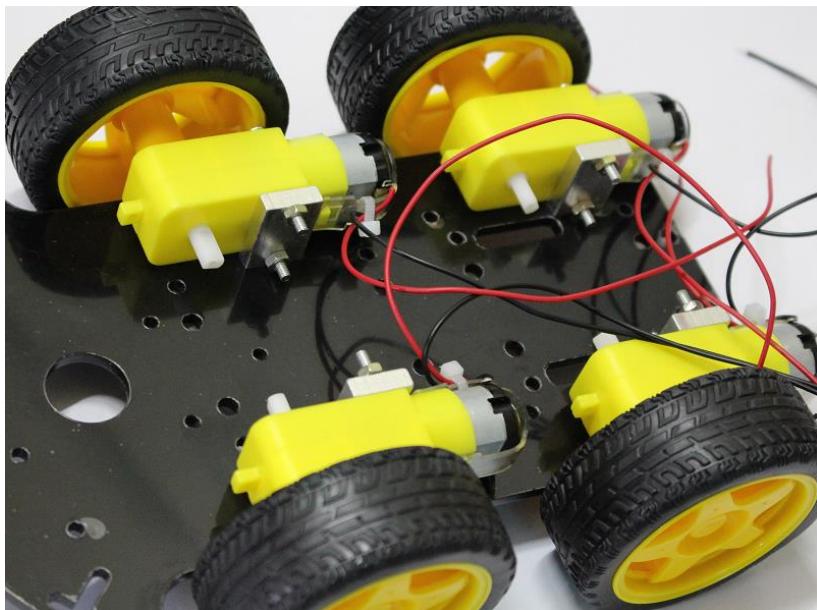


### 5.3 DC 모터 리드선 연결 시 리드선 주의 사항

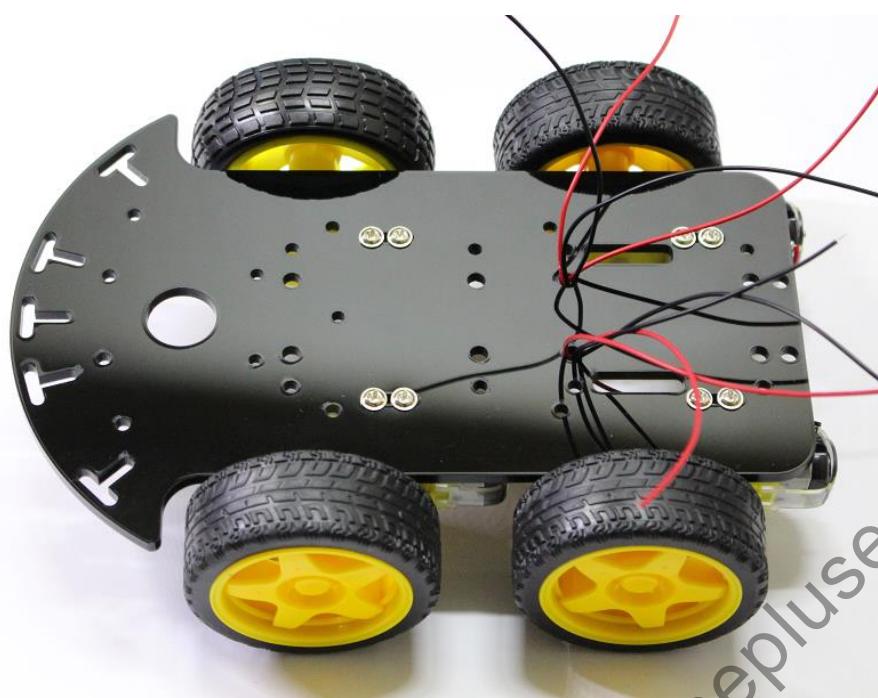
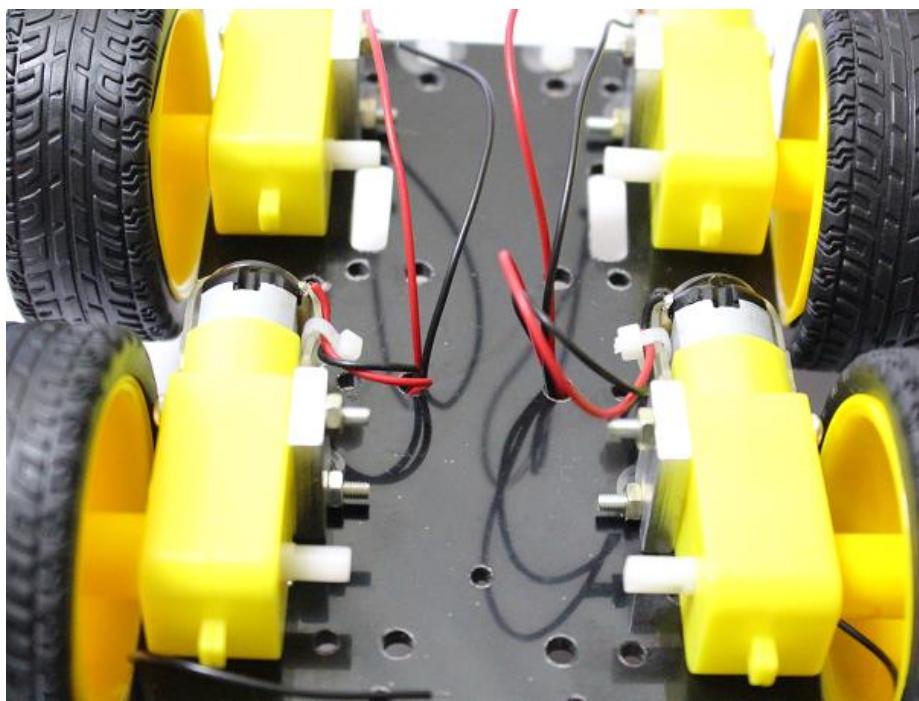
부품 입고 순서 및 사정에 의해 DC 모터 리드선의 색상이 다를 경우에는 납땜이 가능하다면 전선을 분리, 일괄적으로 배선 순서를 다시 납땜 해주어도 무방합니다. 그냥 사용하는 경우에는 L298N 제어 보드의 DC 모터 전원 연결 배선할 때 리드선의 연결된 위치를 염두에 두시고 배선하기 바랍니다.

### 5.4 DC 모터 전원 연결선 정리

하판 샐시 프레임에 DC 모터 & 타이어 고정 후 그림입니다. 뒷면으로 전선들을 훌더 사이로 넣어 줍니다.



전선들을 위로 올려 주는 훌더 위치 선택은 적절히 해줍니다.



## 6 L298N 듀얼 모터 제어 드라이버

DC 모터의 회전 및 속도를 제어하기 위해 사용하는 모듈입니다. L298N은 모터 제어에 많이 사용됩니다. 모터 포트의 극성을 변경할 수 있으며 전압 조절 포트로 입력 전압에 의해 모터 출력 포트로 전압 조절이 가능합니다.

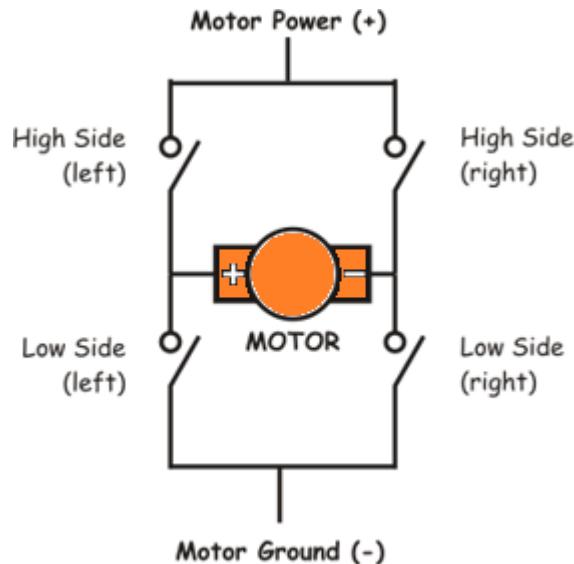


2 개의 DC 모터 포트가 있으며 최대 2A 까지 지원 합니다. 전원 입력은 VMS 포트에 **5 V** 부터 **35 V** 까지 지원 되어 많은 종류의 DC 모터를 구동 시킬 수 있습니다. 권장 전압은 5V 부터 12V~24V 까지입니다.

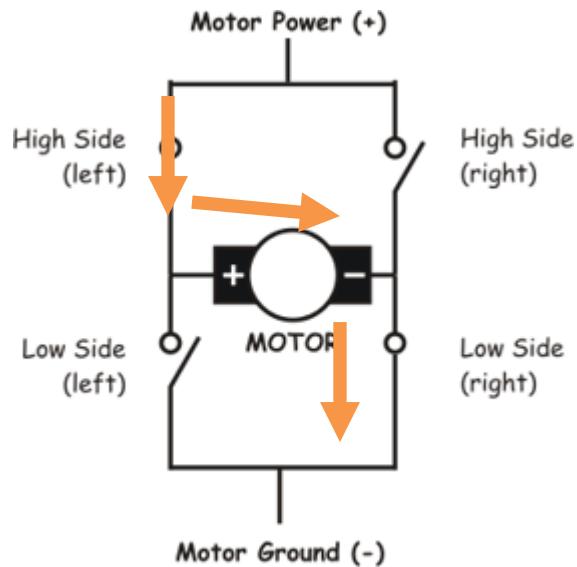
L298N 듀얼 H-Bridge 모듈은 2 개의 모터를 제어 하는데 사용합니다. 정방향/역 방향 변경 가능한 DC 모터 전원 연결 포트가 2 개 있습니다. DC 모터는 전원 연결시 (+)(-) 극성 구분이 없습니다. 연결된 (+)(-)를 반대로 연결하면 회전 방향이 역방향이 됩니다.

## 6.1 H-BRIDGE 회로

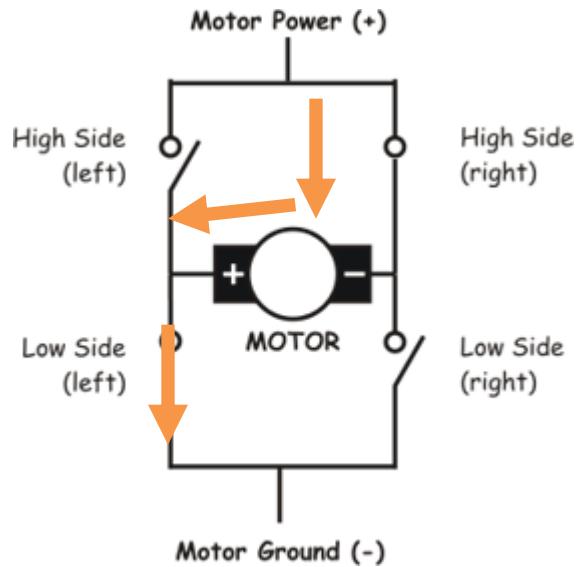
H-BRIDGE 전류의 흐름을 정방향/역방향 변경해 주는 회로의 정식 명칭이기도 합니다. 알파벳 H 문자와 비슷합니다. DC 모터 연결 후 전류의 흐름을 바꾸어 주면 모터의 회전 방향이 바뀌게 됩니다.



H-Bridge 스위치의 중앙에는 DC 모터 같은 (+)(-) 극성으로 연결되어 있어야 됩니다. 양쪽에 위치한 스위치 열고/닫기 조합에 의해 전류의 흐름 방향이 반전 됩니다. 물론 L298N IC 내부에서는 위의 스위치의 기능을 트랜지스터(TR NPN, PNP)가 대신하고 있습니다.



DC 모터 방향 변환 양극에서 음극으로 흐를 경우

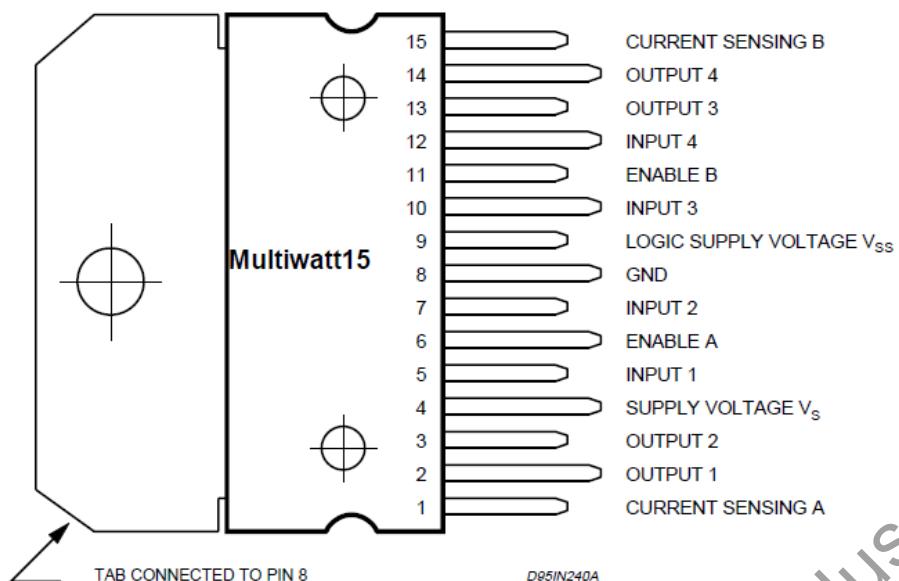


DC 모터 방향 변환 음극에서 양극으로 흐를 경우

## 6.2 L298N 모터 제어 보드

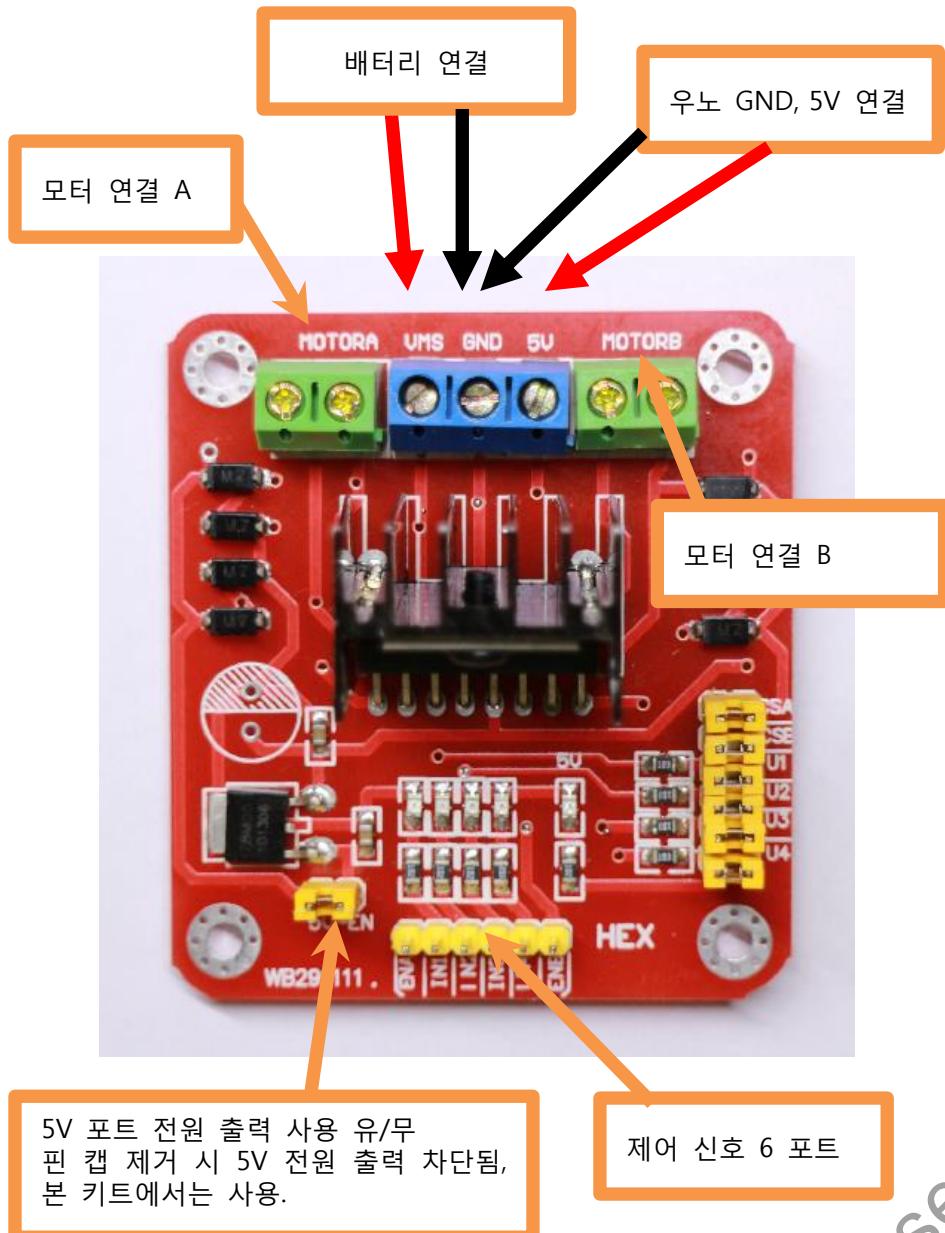
L298N 제어 보드 사양은 아래와 같습니다.

- 드라이버 IC: L298N DUAL H-BRIDGE 드라이버 칩
- 드라이브 부분의 공급 전압 VMS: +5 V ~ 35 V-
- 드라이브 부분의 최대 전류: 2A / bridge
- 시그널 공급 전압 VSS: 4.5-5V
- 시그널 작동 전류 범위: 0 ~ 36mA
- 입력 제어 신호 전압 범위: H: 4.5~ 5.5V / L: 0V
- 최대 소비 전력: 20W
- 보관 온도: -25 °C까지 130 °C
- 드라이버 보드 크기: 55mm \* 60mm \* 30mm
- 드라이버 보드 무게: 33g
- 기타 기능: 방향 제어 표시 LED, 전원 표시 LED



L298N 부품 핀 설명

L298N 제어 보드의 포트 설명 그림입니다.



제어 포트 설명

그림에서 상단의 나사가 달려 있는 포트 중 가운데 3 개의 포트 (파란색 터미널 블럭)는 전원 입력 부분입니다. 왼쪽 상/하 모터 / 전원 연결 부분 Motor A, Motor B 있습니다.

**Motor A:** DC 모터 연결 포트.

**Motor B:** DC 모터 연결 포트.

**VMS:** 배터리 / 외부 전원 입력 포트.

**GND:** 그라운드 포트입니다. 아두이노 / 배터리 외부 전원 연결 공통 사용.

**5V:** 5V 시그널 가동 전원 연결 포트입니다.

**5V EN:** VMS 에서 받은 전원으로 레귤레이터에서 정제된 5V 출력하기 위해 사용하는 포트입니다. 우노 R3 (센서 실드 적층 된 상태) 보드 전원을 별도로 입력하여 주므로 핀 캡 제거(열기) 후 사용하기 권장합니다. 물론 핀 캡 덮어 둔 상태로 사용해도 무방합니다. 나노 V3, 프로 미니 보드 등을 사용할 경우 역 전류가 흐르게 될 수도 있습니다.

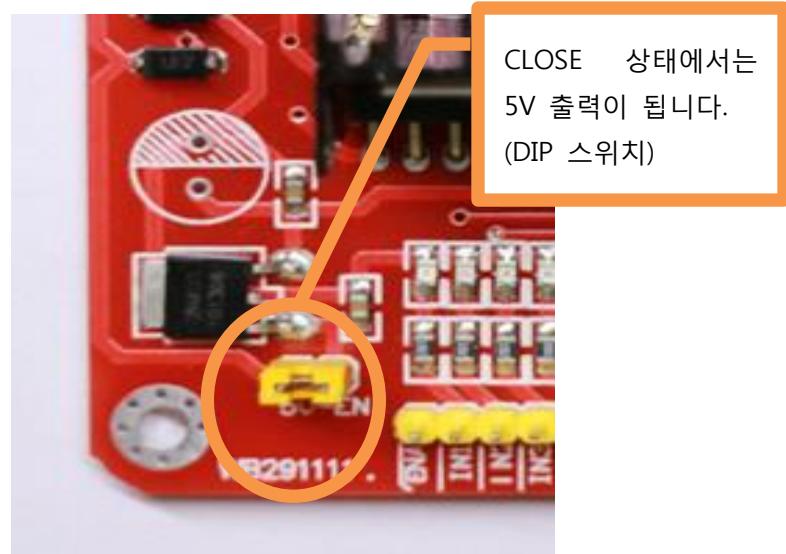
만약, 5V\_EN 을 사용(CLOSE 상태)하는 상태인 경우 L298N 의 5V 포트와 우노 R3 (센서 실드 V4)의 VIN 포트에 연결하고 우노 R3 에는 별도의 전원 입력(9V 배터리)을 하지 않아도 됩니다.

사용시 적절히 선택하시기 바랍니다.

아두이노 보드	모터 드라이버 모듈	배터리 / 외부 전원
5V	5V	
GND	GND	(-)
	VMS	(+)

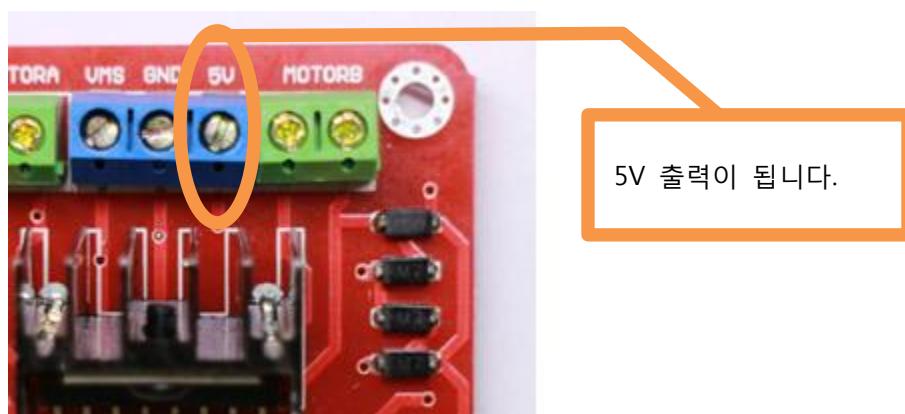
### 6.3 5V\_EN 포트 사용 설명

5V\_EN 포트의 스위치가 닫혀 있는 상태입니다.



5V\_EN 포트 위치

위와 같이 되어 있는 상태인 경우 5V 포트로 전압이 나오게 되어 있습니다.



반대로 오픈 상태이면 5V 포트로 전원 출력이 안됩니다.

본 키트에서는 5V\_EN 포트 닫혀진 상태로 사용하면 됩니다.

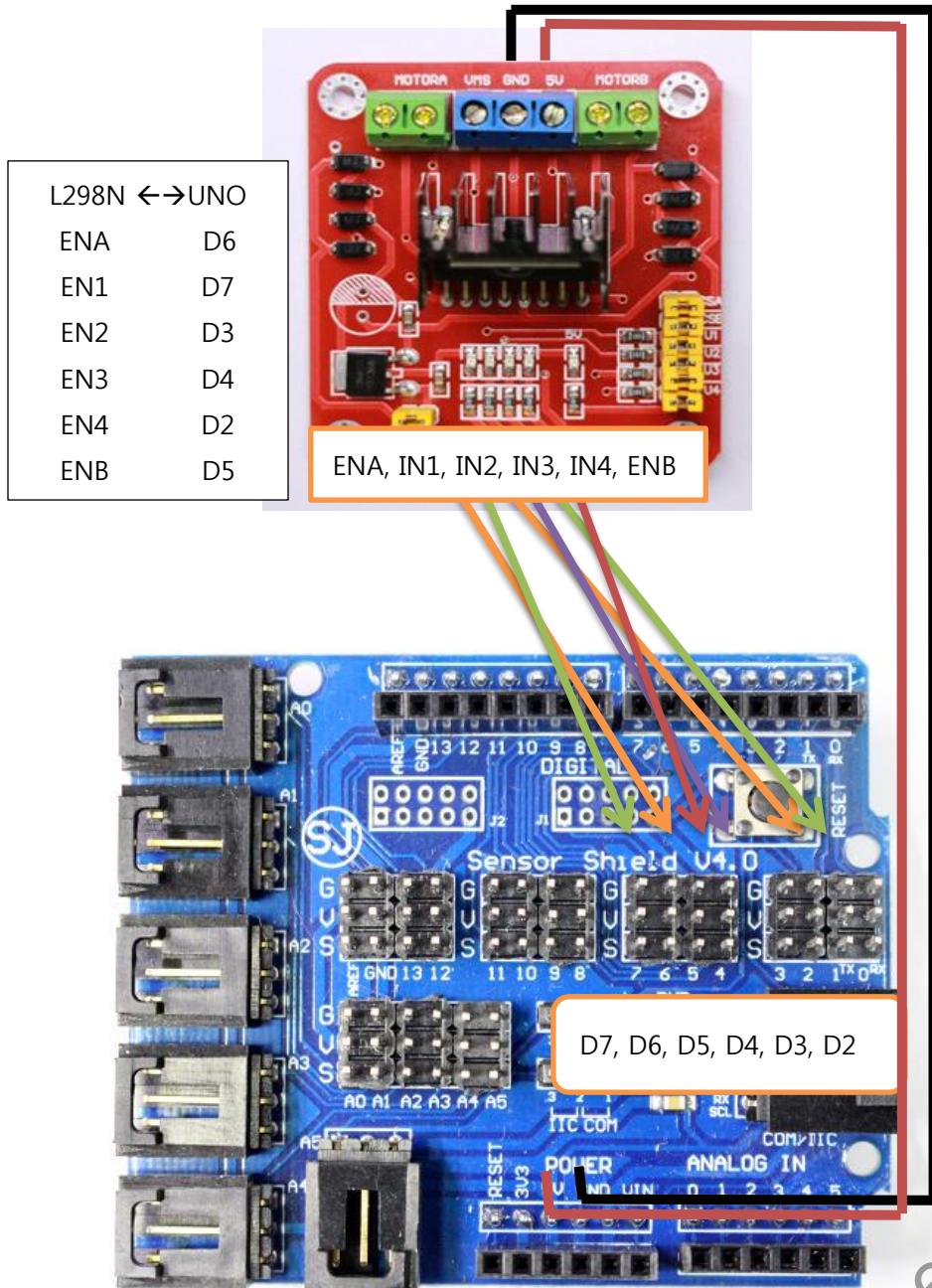
## 6.4 아두이노 우노 R3 & L298N 제어 포트 연결

운행 방향, 속도 제어하기 위해 L298N 제어 보드의 포트와 연결해야 합니다. ENA, IN1, IN2, (D3, D4, D5), ENB IN3, IN4, ENB (D9, D8, D7) 순서로 연결 합니다. 물론, 아두이노 보드와 연결 시 같은 성격의 해당 핀에 연결하여도 무방합니다.

ENA, ENB 는 PWM 지원 포트에 연결합니다.

L298N 모듈	아두이노 보드	비고
ENA	D6	PWM 포트
IN1	D7	디지털 포트
IN2	D3	디지털 포트
ENB	D5	PWM 포트
IN3	D4	디지털 포트
IN4	D2	디지털 포트

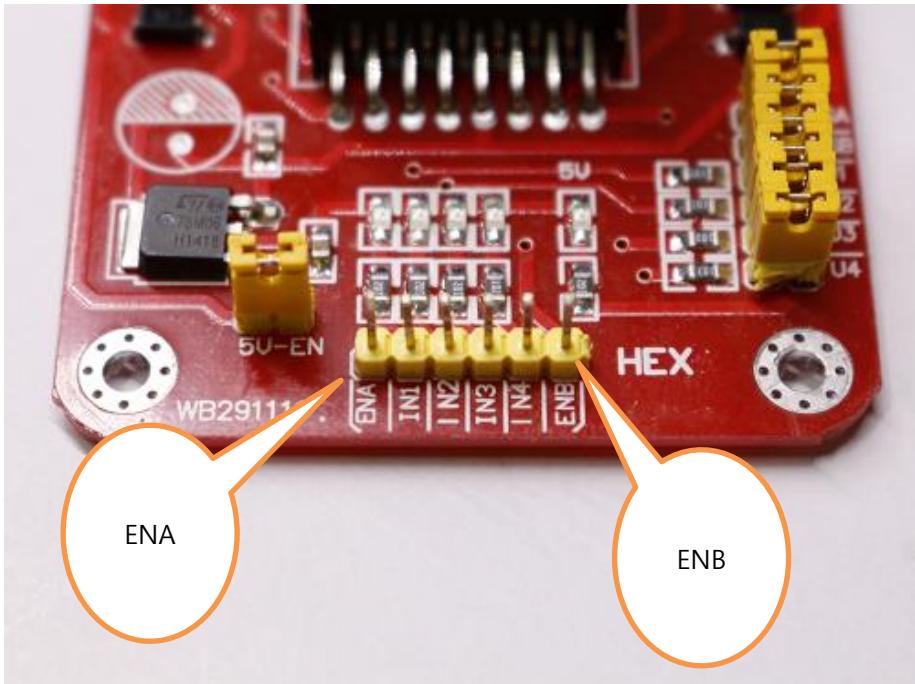
4WD 스마트 로봇 자동차를 만들기 위한 초음파 센서, 서보모터, 적외선 수신모듈, L298N 제어 보드와 연결됩니다. 연결하기 편한 GVS 포트를 사용하여도 되며, 기존 아두이노 핀 배열 형태의 Female 헤더도 지원됩니다.



L298N 센서 실드 연결 설명도

## 6.5 자동차 속도 및 방향 조절

L298N 제어 보드의 **ENA**, **ENB** 는 DC 모터의 속도 조절 및 출발/정지에 사용할 수 있는 포트입니다.



L298N ENA, ENB 포트

ENA, ENB 아두이노의 PWM 지원되는 핀(D3, 5, 6, 9, 10, 11)에 연결하고 나머지는 임의의 Digital 핀에 연결합니다. 연결하는 이유는 DC 모터의 속도 조절을 하기 위해서입니다. 아두이노의 PWM 기능의 포트는 `analogWrite` 함수를 사용하여 전압 조절이 가능합니다. 본 키트에 사용되는 DC 모터는 3V~6V 사용 가능 합니다. (모터의 권장 전압 범위 3.3V~5V)

위 그림에서 하단 가운데에 있는 ENA, IN1, IN2 (이상 모터 A 제어용), IN3, IN4, ENB (이상 모터 B 제어용) 핀들은 모두 아두이노 보드에 연결합니다. ENA 와 ENB는 모터 A, B 를 Enable & 전압의 세기를 설정 할 수 있는 포트입니다. ENA 포트는 IN1, IN2 연동 됩니다. ENB 포트는 IN3, IN4 연동 됩니다.

속도 조절은 `analogWrite` 함수를 사용하여, ENA, ENB 연결된 포트의 스케치 코드에서의 속도 조절. 위에 기술된 기본 운행 예제 코드 중에 아래와 같은 부분을 참조하여 비교해봅니다.

```
#define ENA 6
```

```
int speed=255; // 속도 설정.
```

```
analogWrite(ENA, speed);
```

`speed` 값은 0~255 범위의 값이 설정될 수 있습니다. 0 은 정지입니다. 0 은 0V 입니다. 255 는 최대 속도가 됩니다. 255 는 5V 입니다. 실제 조립 후 운행시 자동차의 속도는 배터리의 잔량 전압에 의해서도 많은 변동이 있으므로 최대 배터리 전압 상태를 감안하여 이해하도록 합니다.

보통 속도 변수의 값은 200 이상으로 놓고 사용해야 합니다.

아두이노에서 PWM 포트에 연결된 ENA, ENB 에 High 전압을 걸어 주고 IN1, IN2 입력을 조절해서 모터 방향을 제어할 수 있습니다. ENA, ENB 에 Low 걸어 주면 IN1, IN2 (IN3, IN4)에 관계없이 정지합니다. ENA, ENB 에 아두이노 PWM 핀(D3, 5, 6, 9, 10, 11)과 연결 하면 모터의 속도를 변경 시킬 수 있습니다.

아래 표를 참고 하시기 바랍니다.

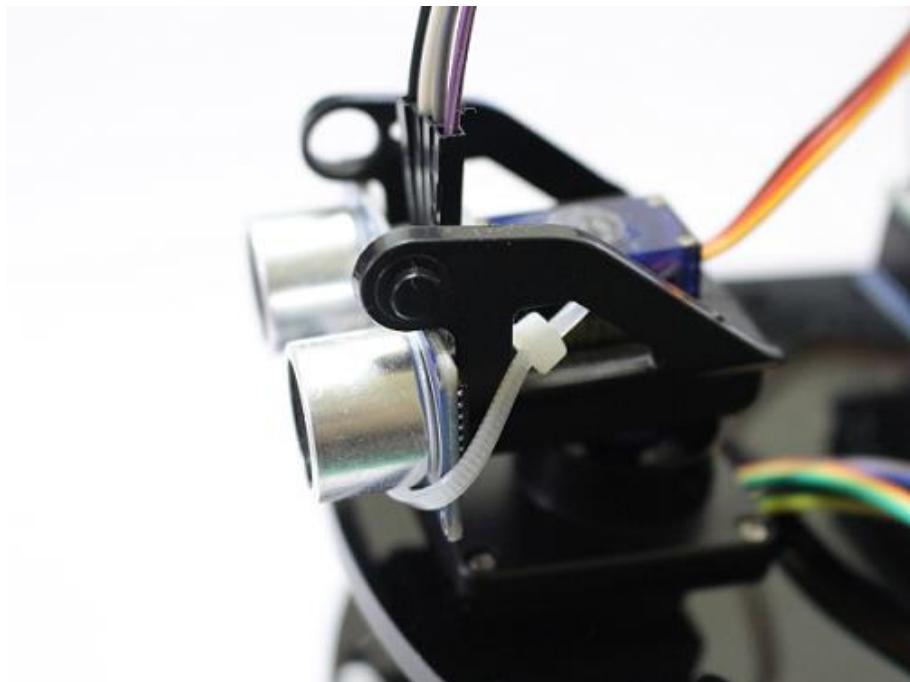
ENA (or ENB)	IN1 (or IN3)	IN2 (or IN4)	모터 A (or B)
High	High	Low	정방향 회전
High	Low	High	역방향 회전
High	High	High	정지
High	Low	Low	정지
Low	ANY	ANY	ANY

우회전은 원쪽 바퀴의 모터는 정방향 회전, 오른쪽 바퀴는 느리게 회전 또는 정지, 반대로 좌회전은 하면 되겠습니다. 차량 동체의 회전 속도는 ENA, ENB 의 값을 적절히 설정하면 됩니다..

가령, ENA, ENB 포트에 `analogWrite(pin 번호, 255);` 5V 전압으로 회전합니다. DC 모터의 RPM 이 높아지므로 속도가 증가하게 됩니다. `analogWrite` 매개 변수 255 를 5V 출력으로 가정할 경우 128 은 2.5V 입니다. ENA, ENB 포트에 `analogWrite(pin 번호, 128);` 255/2 이므로 2.5v 모터가 회전 안 하거나 회전을 하더라도 늦게 회전됩니다. 모터는 3V~6V 구동 전원입니다. `analogWrite( pin 번호, 200);`  $200/255 = 0.78$   $5V * 0.78 = 3.9V$  정도로 구동됩니다. ENA, ENB 값들은 4WD 운행의 방향 및 회전에 유용하게 사용됩니다.

## 7 초음파 센서와 PAN/TILT 거치대 조립

초음파 센서를 고정 시킬 수 있는 거치대입니다. 좌우/상하 회전 지원되는 고정 틀 입니다. 서보 모터 9g 2 개를 사용 할 경우 좌우/상하 조절 가능합니다. 본 키트에서는 서보 모터 1 개 사용 합니다. 방향 설정 참조 거리 측정 센서 값을 구하기 위해 좌우 회전만 필요로 합니다. 초음파 센서는 아래의 이미지처럼 얇은 케이블 타이로 결합, 고정 해줍니다.



초음파 센서 거치대 완성 모습

## 7.1 팬, 틸트 거치대 조립



팬/틸트 거치대 부품들

받침대와 모터 고정 부품과, 상단 받침대, 볼트 너트로 구성 되어 있습니다. 받침대와 서보 모터 톱니바퀴 고정 부품입니다.

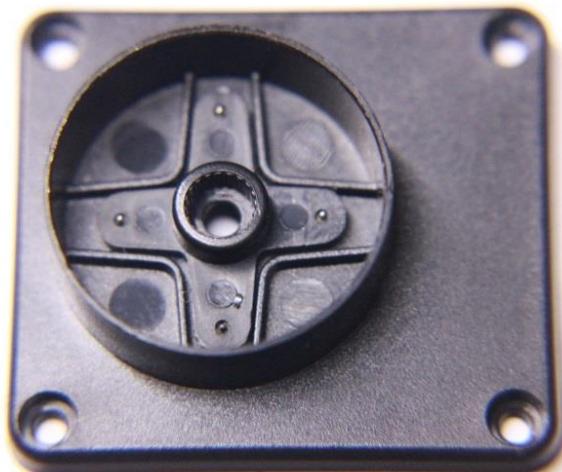


거치대 바닥 받침대

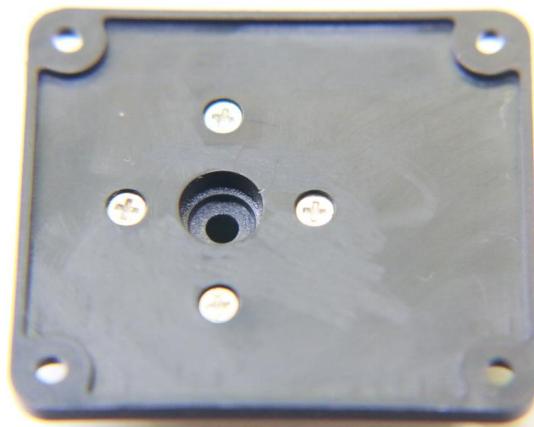


서보 모터 샤프트 허더

받침대와 서보 모터 텁니바퀴 고정된 상태의 이미지입니다. 약간 짧은 부분과 긴 부분을 유의하여 위치를 맞추면 정확하게 맞습니다. 4 개의 작은 나사로 하단부 또는 상단부에서 조여주면 됩니다.

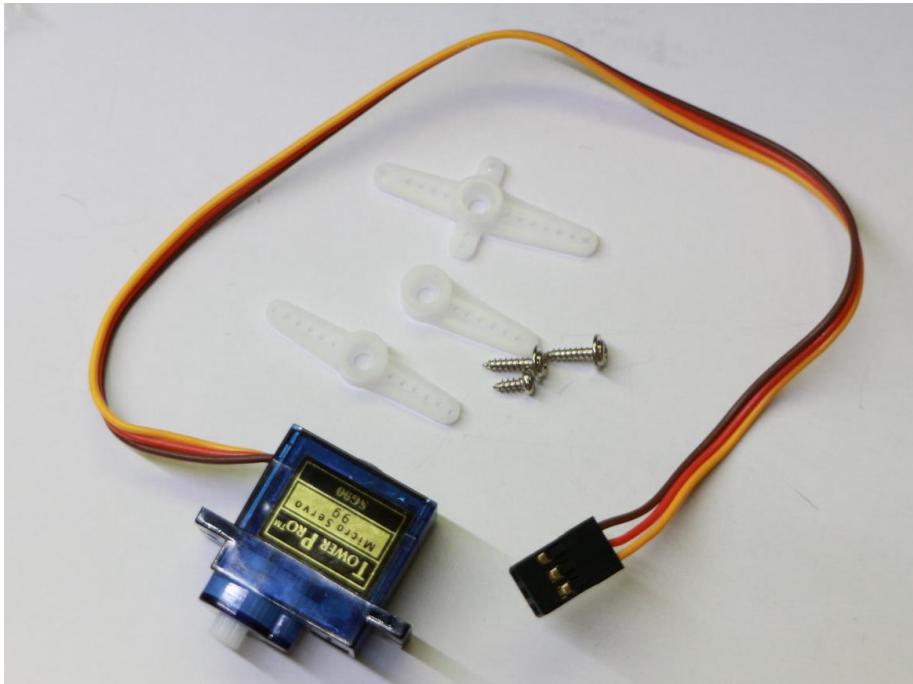


받침대에 서보 모터 샤프트 홀더 고정된 모습



침대에 서보 모터 샤프트 고정 하부 볼트

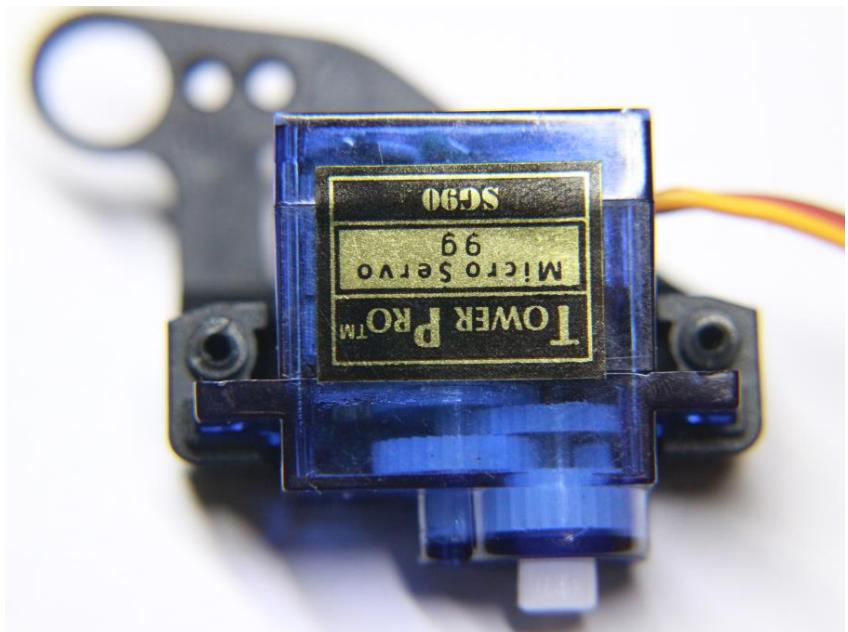
## 7.2 거치대에 서보 모터 결합



서보 모터 sg90 & 고정 부품들



서보모터 고정 틀 좌/우 방향



서보모터 고정 위치



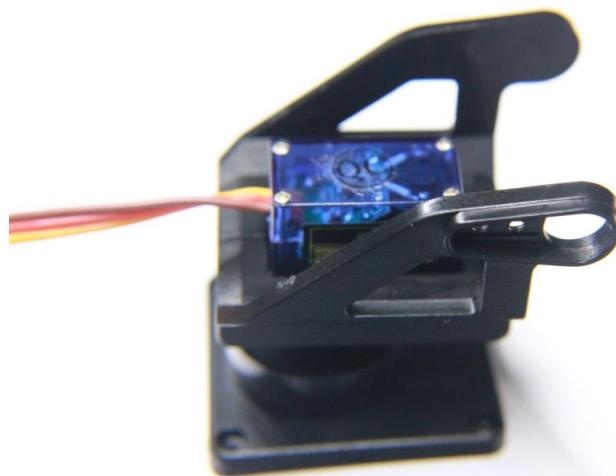
서보모터 고정 후 아래에서 본 모습

위의 이미지처럼 완성되면 볼트를 사용하여 고정합니다. 볼트를 사용하지 않아도 꽉 물려진 상태라 쉽게 분리 되지 않습니다.



샤프트 홀더 고정

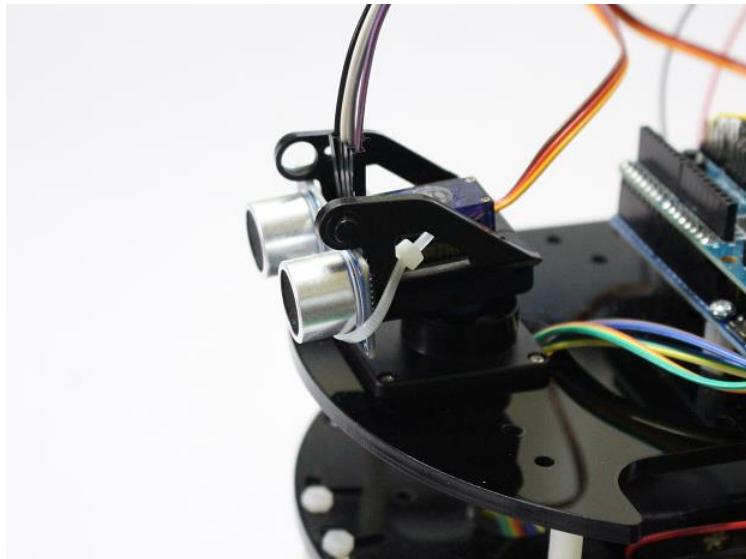
서보 모터의 톱니바퀴를 아래 받침 고정 틀에 얹어 주는 식으로 맞추어 줍니다.  
그럼 아래의 이미지처럼 장착되게 됩니다.



거치대 자체 완성된 모습

### 7.3 초음파 센서 장착

팬/틸트 거치대의 용도는 미니 카메라를 거치하는 용도의 제품이라 아래의 이미지를 참조하여 초음파 센서를 적절히 결합합니다.



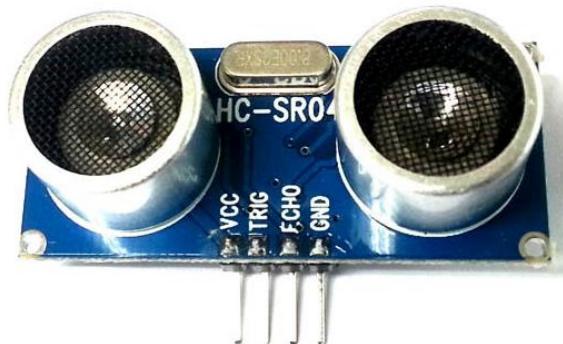
케이블 타이로 고정



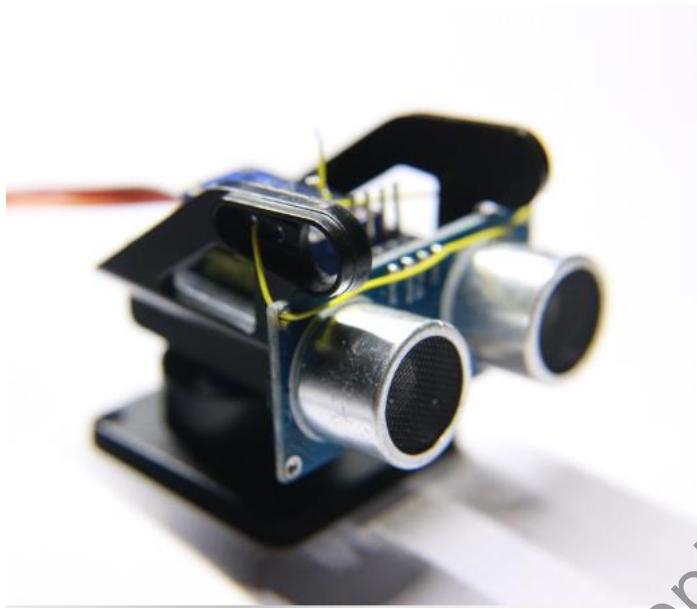
상단 프레임의 앞쪽의 4 부분, 초음파 센서 거치대 Holder 홀더 위치에 서보모터 팬/틸트 받침대를 장착합니다.

## 8 초음파 센서 프로그래밍

초음파 센서를 탑재하여 전방의 물체와의 거리 측정하여 장애물을 피해가기 위한 방향 전환 데이터 수집에 사용합니다.



초음파 센서 모듈

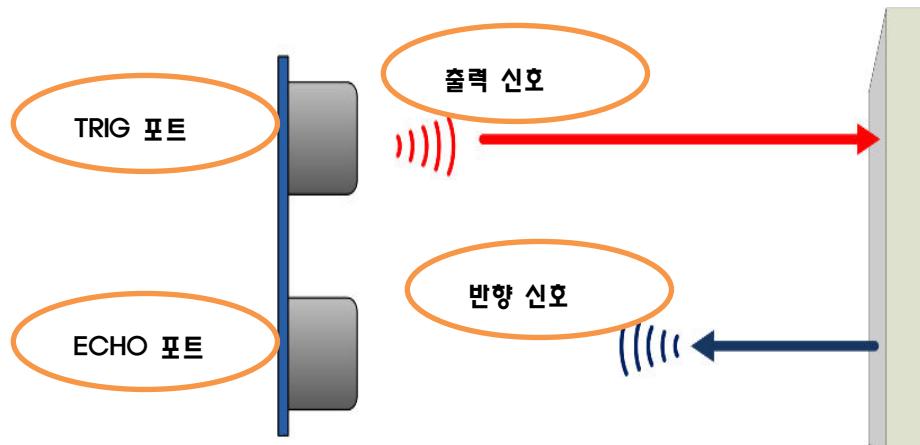


초음파 센서 모듈 거치대 사용 모습

## 8.1 초음파 센서 작동 원리

초음파 거리 측정 센서의 기본 원리는 음파를 쏘아서 반향 되어 수집되는 음파 까지의 시간차로 거리를 계산할 수 있습니다. 음속이 340m/s (1 초에 340 미터) 센서를 통해 응답이 오는 시간만 알면 초음파 센서 앞에 있는 사물까지의 거리를 쟈 수 있습니다.

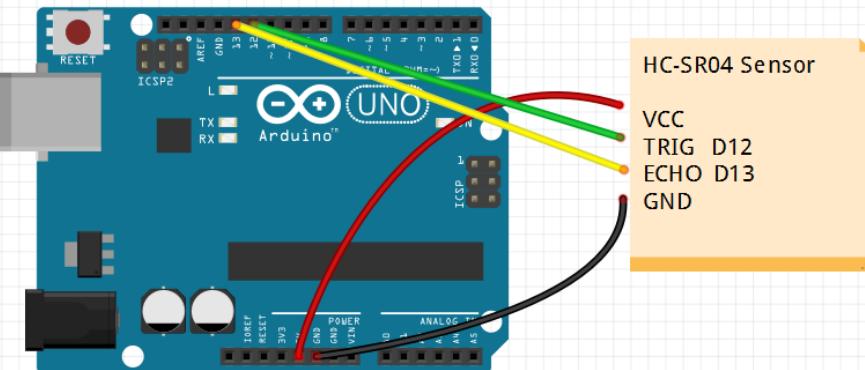
측정거리: 2cm ~ 5m (4m), 측정각도: 15 도, 측정 해상도: 3mm



초음파 센서 작동 원리

초음파 센서 모듈에는 4 개의 핀(포트)이 있습니다. VCC, Trig, Echo, GND 입니다.

초음파 센서 모듈	아두이노
VCC	5V
Trig	D12
Echo	D13
GND	GND



초음파 센서 연결 포트

아두이노에서 Trigger(트리거) 핀으로 HIGH 를 입력하면 초음파 모듈에서 40KHz 음파를 발사합니다. (10us 이상 HIGH 유지) 이때부터 Echo 핀은 High 상태가 되고, 음파가 되돌아와 수신되면 echo 핀이 다시 Low 상태가 됩니다. 이 간격에서 거리를 구하고 다시 2로 나누면 됩니다.

음파속도가 340m/s 이고 1cm 가는데 29us 가 걸립니다. 거리를 구하는 공식은

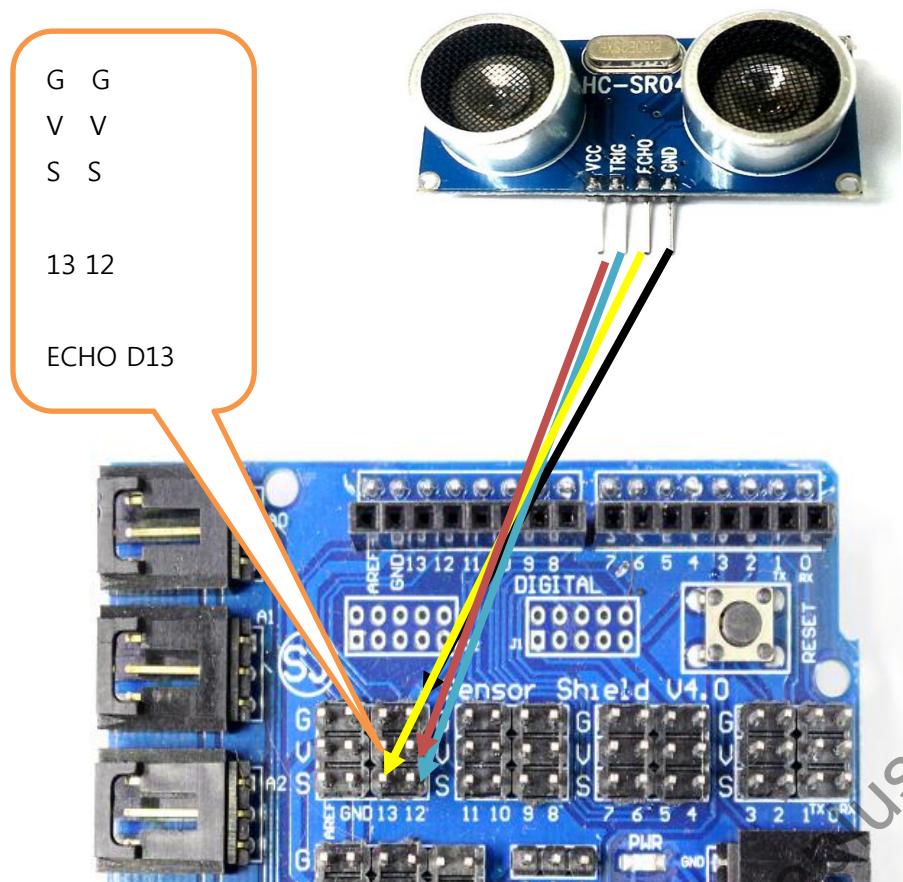
$$\text{Distance} = \text{time} / 29 / 2$$

HC-SR04 는 약 4 미터 정도의 거리까지 측정 가능합니다.

## 8.2 초음파 센서 연결하기

키트에 포함된 뒤풋 케이블 4 개를 떼어서 사용하도록 합니다. 또는 이미 가지고 있는 케이블 Female To Female 케이블을 사용하면 됩니다.

초음파 센서의 TRIG 포트는 D12 번, ECHO 포트는 D13 에 연결하도록 합니다. 센서 실드의 S 표시부분에 연결하여야 합니다. G, V, S 표시된 부분은 G: GND 는 (-) 그라운드, V: VCC (+) 전원, S: SIGNAL 신호 포트입니다. 초음파 센서의 GND 는 센서 실드의 G 표시 핀에 연결하도록 합니다. G, V 표시된 부분은 동일한 성격의 핀으로서 원하는 핀에 연결하면 됩니다.



초음파 센서 연결

## 8.3 초음파 센서를 이용한 거리 측정 구현

스케치 라이브러리 NewPing, 또는 디렉트로 연결하여 사용해도 무방합니다. 거리 측정을 위한 2 가지 코딩 방식으로 테스트 후 원하는 코드를 사용하여 구현하도록 합니다.

### 8.3.1 예제코드 1 – 스케치 NewPing 라이브러리 사용

NewPing 라이브러리를 사용하도록 합니다.

<http://playground.arduino.cc/Code/NewPing>에서 다운로드 합니다. NewPing 라이브러리 설치 후 NewPing 라이브러리에 포함된 기본 예제, 아래의 코드 업로드 하여 프린트 되는 거리값을 확인해 봅니다. 초음파 센서 앞쪽을 손으로 막아서 거리 값을 확인해 보도록 합니다.

코드: 4wd\_ex\_3

```
#include <NewPing.h>

#define TRIGGER_PIN 12
#define ECHO_PIN     13
#define MAX_DISTANCE 400 // 4 미터

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

void setup() {
    Serial.begin(9600);
}

void loop() {
    delay(50); //
    int uS = sonar.ping();
    // 측정된 거리.
```

```
    Serial.print("Ping: ");
    Serial.print(uS / US_ROUNDTRIP_CM);
    Serial.println("cm");
}
```

### 8.3.2 예제코드 2 – 디렉트 Trig, Echo 사용

NewPing 등의 라이브러리를 사용하지 않고 직접 구현해 봅니다.

초음파 센서 모듈의 Trig, Echo 포트에 대한 컨트롤을 코드에서 처리합니다.

코드: 4wd\_ex\_4

```
#define TRIGGER_PIN 12
#define ECHO_PIN     13

void setup()
{
    Serial.begin(9600);
    pinMode(TRIGGER_PIN, OUTPUT); // 센서 Trig 핀, D12y
    pinMode(ECHO_PIN, INPUT); // 센서 Echo 핀, D13
}

void loop()
{
    long duration, cm;
    digitalWrite(TRIGGER_PIN, HIGH); // 센서에 Trig 신호 입력
    delayMicroseconds(10); // 10us 정도 유지
    digitalWrite(TRIGGER_PIN, LOW); // Trig 신호 off
    duration = pulseIn(ECHO_PIN, HIGH); // Echo pin: HIGH->Low 간격을 측정
    cm = microsecondsToCentimeters(duration); // 거리(cm)로 변환
    Serial.print(cm);
    Serial.print("cm");
```

```
Serial.println(0);
delay(300); // 0.3 초 대기 후 다시 측정
}

long microsecondsToInches(long microseconds)
{
    // According to Parallax's datasheet for the PING()), there are
    // 73.746 microseconds per inch (i.e. sound travels at 1130 feet per
    // second). This gives the distance travelled by the ping, outbound
    // and return, so we divide by 2 to get the distance of the obstacle.
    // See: http://www.parallax.com/dl/docs/prod/acc/28015-PING-v1.3.pdf

    // 시간에 대한 값을 인치로 변환.
    return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds)
{
    // The speed of sound is 340 m/s or 29 microseconds per centimeter.
    // The ping travels out and back, so to find the distance of the
    // object we take half of the distance travelled.
    // 시간에 대한 값을 센티미터로 변환
    return microseconds / 29 / 2;
}
```

## 9 서보 모터 제어

서보 모터 회전하는 코드를 구현해봅니다. 서보 모터는 회전량을 조절하기 위한 용도로 사용됩니다. 보통 0 도~180 도까지 회전합니다. 일반적인 서보 모터는 전원 공급을 위한 GND, VCC 입력 포트와 회전 조절을 위한 신호 포트 1 개. 모두 3 개의 포트가 필요합니다.



Servo Motor sg90

### 서보모터 기술 사양:

- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree (0.1 초에 60 도 회전 속도)
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10  $\mu$ s
- Temperature range: 0 °C – 55 °C

서보 모터 모듈	Arduino Uno
Black wire	GND
Red wire	5V
Blue or Yellow wire	D9 or other PWM port

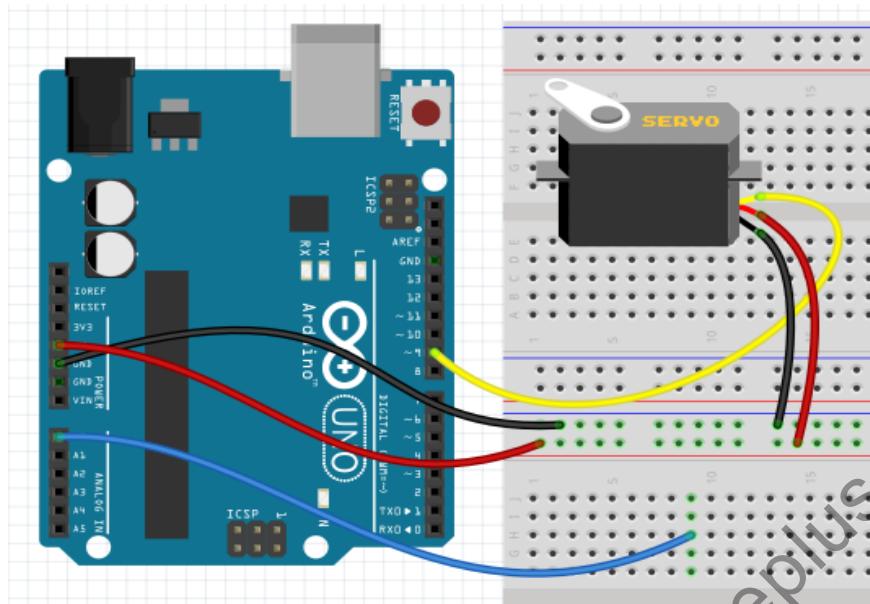
서보모터의 Black (or Brown )연결선은 그라운드에 연결합니다. 서보모터의 Red (또는 Light Red) 연결선은 5V 입력되게 합니다. 서보모터의 Blue or Yellow 연결선은 우도 R3 보드에 9 번 포트에 연결합니다. 서보모터 제어는 스케치 IDE 에 기본 포함된 Servo 모터 라이브러리를 사용하도록 합니다. Servo 모터 라이브러리를 사용하기 위해서는 스케치 코드에서 헤더파일을 선언하면 됩니다.

```
#include <Servo.h> // 서보 모터 라이브러리 사용 선언
```

아두이노 사이트에 서보 모터 예제 및 설명이 있습니다.

1 번째 예제는 <http://arduino.cc/en/Tutorial/Sweep>

예제는 0 ~ 180 도 회전 후 다시 0 위치로 복구 되는 코드입니다.



브레드보드 연결 회로도

```
// Sweep
// by BARRAGAN <http://barraganstudio.com>
// This example code is in the public domain.

#include <Servo.h>

Servo myservo; // create servo object to control a servo
                // a maximum of eight servo objects can be created

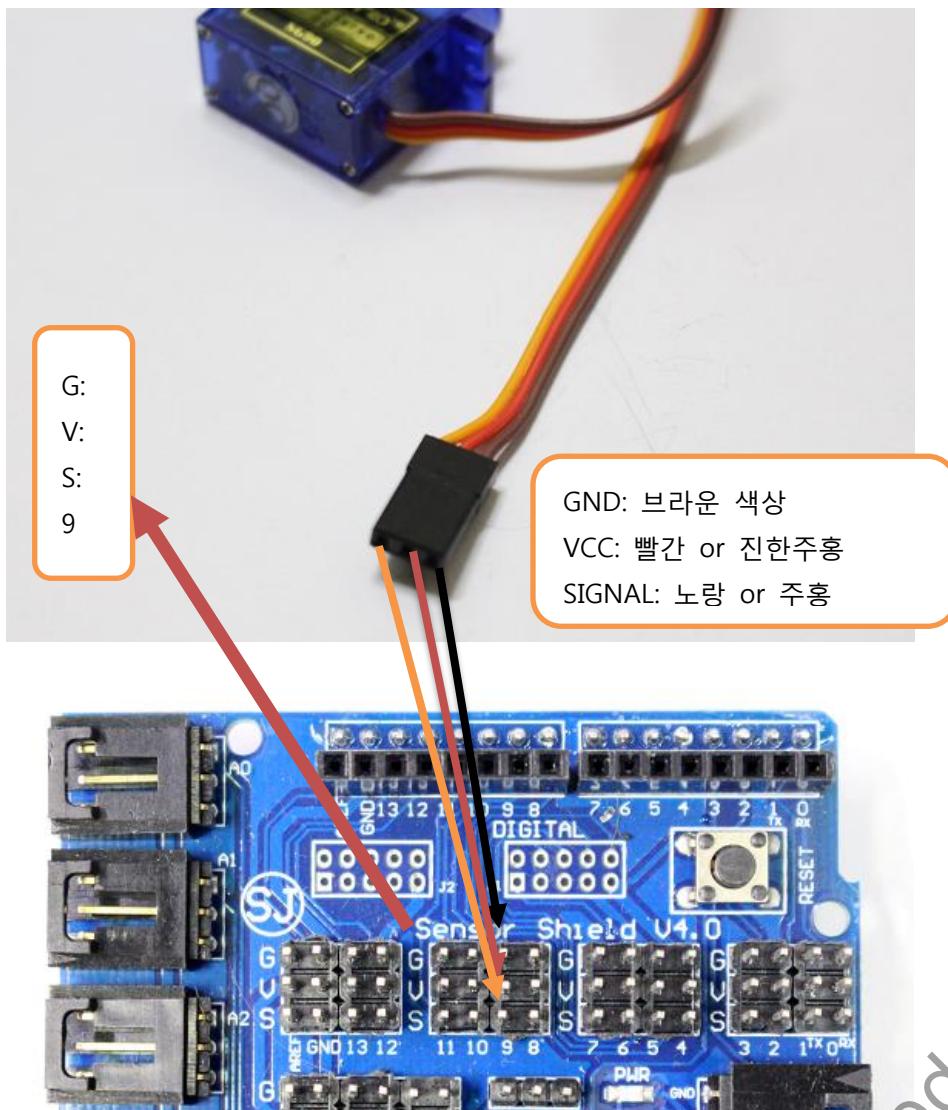
int pos = 0;    // variable to store the servo position

void setup()
{
    myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop()
{
    for(pos = 0; pos < 180; pos += 1) // goes from 0 degrees to 180 degrees
    {
        myservo.write(pos);          // tell servo to go to position in 'pos'
        delay(15);                  // waits 15ms for the servo to reach the
position
    }
    for(pos = 180; pos>=0; pos-=1) // goes from 180 degrees to 0 degrees
    {
        myservo.write(pos);          // tell servo to go to position in variable 'pos'
        delay(15);                  // waits 15ms for the servo to reach the position
    }
}
```

## 9.1 서보모터 연결하기

서보모터 SG90 모듈을 아두이노 센서 쉴드에 연결시 아래의 그림을 참조합니다.



## 9.2 서보 모터의 중앙 기준 방향 설정

서보모터의 중앙기준 방향 설정을 해야 측정할 각도의 범위를 정할 수 있습니다. 서보 모터는 3 방향 상태에서 정지합니다. 서보모터는 3 방향의 각각 설정 후 초음파 센서로부터의 거리를 구합니다. 서보 모터는 전방 장애물과의 거리를 측정하기 위해 120 도 3 등분 범위로 3 방향 왼쪽/중앙/오른쪽 회전합니다.  $120 / 3 = 40$  도, 120 도를 3 등분 하면 40 도가 됩니다. 또는 중앙 기준으로 40 도 왼쪽, 중앙 기준으로 40 도 오른쪽으로 이해하고 서보모터를 회전하면 됩니다.

간단한 코드 업로드를 하여 서보모터 초기화 및 기준이 되는 중앙 회전 값을 지정합니다. 즉, 코드 업로드를 하여 서보모터를 중앙 회전 값으로 지정 및 고정시키고 차량 정면을 향하도록 합니다.

서보모터 sg90 모듈의 회전 값은 최소, 최대는 0 도부터 180 도입니다.

중앙값을 85 도 지정합니다. 서보모터 중앙 기준 값을 85 도로 지정한 이유는 서보모터와 거치대 기어 물리는 부분이 85 도 일 때 정확히 중앙을 향하게 됩니다. 서보모터마다 약간의 오차가 있을 수 있으므로 아래의 코드를 참조하여 수정하면 됩니다.

코드: 4wd\_ex\_5

```
#include <Servo.h>      // 서보 모터 사용하기 위한 헤더파일 선언.

Servo myservo;

void setup()
{
    // 아두이노 우노의 9 번 포트에 연결합니다.
    myservo.attach(9);
    // 중앙 방향으로 회전 합니다.
    myservo.write(85); // or 89
}
```

```
void loop()
{
    // 빈 함수입니다.
}
```

위의 코드 업로드 후 기어 위치를 위의 설명을 참조하여 거치대의 서보모터가 전방 중앙으로 맞추어 줍니다.

서보모터 중앙 회전 값 고정 상태에서 서보모터를 끼워놓은 상태에서 강제로 돌리지 말고 손으로 빼서 다시 끼워서 기어 위치를 맞추어야 합니다. sg90 서보모터의 전원이 들어간 상태에서 강제로 돌리면 서보모터의 기어가 불량 될수 있습니다.

위에서 사용한 중앙 방향 설정 코드를 기초로 구현 해봅니다.

아래의 코드는 중앙 85 도 기준으로 왼쪽 40 도, 오른쪽 40 도 서보모터 회전하는 코드입니다. 각각의 구분 회전 단계마다 3 초 잠시 정지를 시킵니다. 물론 중앙 방향 기준 좌/우 40 도는 절대적인 기준값이 아니므로 테스트 및 환경에 따라 적절히 조절 해주면 됩니다.

코드: 4wd\_ex\_6

```
#include <Servo.h>    // 서보 모터 사용하기 위한 헤더파일 선언.

Servo myservo;

void setup()
{
    Serial.begin(9600);

    // 아두이노 우노의 9 번 포트에 연결합니다.
```

```
myservo.attach(9);

// 중앙 방향으로 회전만 합니다.
// 시스템 기동 시 중앙 방향으로 향하게 합니다.
myservo.write(85); // or 84
delay(3000); // 3 초 지연.

}

void loop()
{
    Serial.println("right");
    // 오른쪽 40 도 방향.
    myservo.write(85-40);
    // 3 초 동안 잠시 정지.
    delay(3000);

    Serial.println("center");
    // 중앙 85 - 0 .
    myservo.write(85-0);
    // 3 초 동안 잠시 정지.
    delay(3000);

    // 왼쪽 85 + 40
    Serial.println("left");
    myservo.write(85+40);

    // 3 초 동안 잠시 정지.
    delay(3000);
}
```

서보모터 회전 전용 함수를 구현하면 아래와 같습니다.

코드: 4wd\_ex\_7

```
#define SERVO_DIR_CENTER 0
#define SERVO_DIR_LEFT 1
#define SERVO_DIR_RIGHT 2

//
// int iDir : 방향 설정 변수
//
int setRotateServo(int iDir)
{
    switch ( iDir )
    {
        case SERVO_DIR_CENTER:
            myservo.write(85-0);
            delay()
            break;
        case SERVO_DIR_LEFT:
            myservo.write(85+40);
            break;
        case SERVO_DIR_RIGHT:
            myservo.write(85-40);
            break;
        default:
            // 지정된 함수 인자가 아닐 경우 처리.
            String errMsg="called int setRotateServo(int ";
            errMsg += iDir;
            errMsg += ")";
            Serial.println(errMsg);
            break;
    }
}
```

### 9.3 조금 더 안전한 서보모터 제어

최소 요구 작동 시간 내에 myservo.write() 함수를 호출 하는 경우 약간의 오차 또는 오류가 발생될 수 있습니다. 서보모터 sg90 모델의 스펙을 서보모터의 회전 속도에 대한 소요 시간이 나와 있습니다. Operating speed: 0.1 s/60 degree → 60 도에 0.1 초 회전 합니다. 반대로 말하면 0.1 초에 60 도 회전합니다.

왼쪽에서 오른쪽으로 120 도 회전 하는 경우 0.2 초의 최소 잠시 정지 시간이 필요합니다. 또는 필요에 의해 더 많은 회전을 하였다면 더 많은 소모시간이 요구 됩니다. 물론, 4WD 운행 코드 구현 시 아래의 예제코드는 필요하지 않을 수 있습니다.

보다 더 세밀한 프로젝트를 위해 적용시켜 봅니다. 코드 및 구현 개념에 따라 여러 가지 방법이 있습니다. 현재 각도 설정에 대한 변수를 정의하여, 현재 서보 모터의 각도를 저장하고 변수를 참조하여 함수를 구현할 수 있습니다. 또는 지원되는 라이브러리를 참조하여 세부 함수를 구현할 수도 있습니다.

아두이노 스케치의 Servo 클래스 라이브러리에는 현재 서보 모터의 회전 각도를 구할 수 있는 함수가 있습니다. Servo 클래스의 read() 함수를 사용하여 현재 서보 모터의 각도를 구할 수 있습니다.

```
int nCurrDegree = myservo.read();
```

그럼 현재 각도에 대한 이동해야 할 각도를 구해 60 도에 0.1 초 기준으로 지연 시간을 계산해줍니다. 조건은 무조건 60 도 미만일 경우에도 0.1 초 지연시간을 주기로 합니다.

```
// nDegree : 회전할 각도  
// 반환값: delay 함수에 들어가는 1 초 1000 비율로 환산.
```

```
int checkServoDelayByDegree(int nDegree)  
{  
    nDegree=nDegree%360; //
```

```
int nCurrDegree = myservo.read(); // 현재 서보모터의 각도
float fDegree = fabs(nCurrDegree - nDegree); // 절대값.
fDegree = fDegree / 60.0f;
fDegree=max(1.0f,fDegree);
//fDegree=min(10.0f,fDegree);
return (int)(fDegree*100.0f);
}
```

## 9.4 서보모터 & 초음파 거리 측정 센서 코드 결합

현재까지 구현된 서보 모터 제어 코드와 초음파 거리 측정 센서 코드를 적용 합니다. 서보모터 회전 함수와 초음파 센서 작동 코드를 결합 합니다. C/C++ 구현 기술은 여러 방법이 있지만 작동 함수 위주로 구현합니다. 위에서 기술된 초음파 센서 거리 측정 라이브러리 NewPing 을 사용하도록 합니다. 스케치 초음파 센서 라이브러리 NewPing 사용해봅니다.

예제 코드는 지정된 3 방향의 거리값을 구해 가장 먼 거리의 방향을 가리키는 메시지를 시리얼 포트로 출력합니다. 최종 결과를 얻기 위해 여러 가지 함수가 추가됩니다. 아래의 코드는 지정된 방향으로 서보모터 회전 후 해당 방향에서의 거리 측정된 값을 구하여 적절한 진행 방향을 선정합니다

코드: 4wd\_ex\_8

```
#include <NewPing.h> // for Ultrasonic sensor.  
#include <Servo.h> // for servo motor.  
  
//  
// for Ultrasonic module  
//  
#define TRIGGER_PIN 12 // Trigger Pin  
#define ECHO_PIN 13 // Echo Pin  
#define MAX_DISTANCE 250 // distance max value. 2.5m  
  
//  
// Direction for servo motor  
//  
#define SERVO_DIR_CENTER 1  
#define SERVO_DIR_LEFT 2  
#define SERVO_DIR_RIGHT 3
```

```

#define SERVO_DEGREE_LEFT
#define SERVO_DEGREE_CENTER
#define SERVO_DEGREE_RIGHT

//  

#define DIR_DISC_CENTER    1
#define DIR_DISC_LEFT      2
#define DIR_DISC_RIGHT     3

// global instance ultrasonic sensor.  

NewPing sonar_distance(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

// global instance servo motor.  

Servo myservo; // create servo object to control a servo  

                // a maximum of eight servo objects can be created

void setup() {  

    Serial.begin(9600);  

    initSystem();  

}

//  

// get value current distance from HC-SR04 ( ultrasonic )  

//  

int getCurrDistance()  

{  

    delay(60);  

    int uS = sonar_distance.ping();  

    return uS / US_ROUNDTRIP_CM;  

}

```

```

// prevent NO_ECHO, prevent 0 cm.
//
int getSafeDistance()
{
    const int nRetry=3; // modify count if you want...
    int nDist=NO_ECHO;
    //
    // simple code.
    // while(nDist==getCurrDistance() != NO_ECHO) ; // while when NO_ECHO
    // for debug message.
    do
    {
        nDist=getCurrDistance();
        if(nDist==NO_ECHO)
        {
            Serial.println("HC-SR04 returned NO_ECHO");
        }
    }
    while(nDist==NO_ECHO);

    // now return safe value.
    if(nDist!=NO_ECHO)
        return nDist;
}

//
// 각 방향에 대한 거리 데이터 보존 구조체.
//
struct DistanceDir
{
    int left;
    int center;
}

```

```
int right;
//
int choosedDir;

int getMax()
{
    int retMax=center;
    if(center<left)
    {
        retMax=left;
    }
    if(retMax<right)
    {
        retMax=right;
    }
    return retMax;
}
//
int getProperDir()
{
    int chooseDir=DIR_DISC_CENTER;
    // get max distance.
    int compareDist=getMax();

    if(compareDist==left)
        chooseDir=DIR_DISC_LEFT;
    if(compareDist==right)
        chooseDir=DIR_DISC_RIGHT;
    //
    if(compareDist==center)
        chooseDir=DIR_DISC_CENTER;
    // return value.
    return chooseDir;
}
```

```

    }

}; // end of struct DistanceDir

struct DistanceDir g_DistanceDirValue;
//  

// measure distance  

//  

int measureDirDistance(int iDir)  

{
    // servo motor rotate  

    setRotateServo(iDir);  

    delay(11); //wait for servo.  

    int currDistance = getSafeDistance();  

    return currDistance;  

}

//  

// direction  

//  

int decisionDirection()  

{
    g_DistanceDirValue.left = measureDirDistance(SERVO_DIR_LEFT);  

    delay(700);  

    g_DistanceDirValue.right = measureDirDistance(SERVO_DIR_RIGHT);  

    delay(700);  

    g_DistanceDirValue.center = measureDirDistance(SERVO_DIR_CENTER);  

    delay(700);  

    int finalDir = g_DistanceDirValue.getProperDir();  

    return finalDir;  

}

//  

// int iDir : 방향 설정 변수  

//

```

```

int setRotateServo(int iDir)
{
    switch ( iDir )
    {
        case SERVO_DIR_CENTER:
            myservo.write(85-0);
            break;
        case SERVO_DIR_LEFT:
            myservo.write(85+40);
            break;
        case SERVO_DIR_RIGHT:
            myservo.write(85-40);
            break;
        default:
            // 지정된 함수 인자가 아닐 경우 처리.
            String errMsg="called int setRotateServo(int ";
            errMsg += iDir;
            errMsg += ")";
            Serial.println(errMsg);
            break;
    }
}

// initialize system.
void initSystem()
{
    myservo.attach(9); // attaches the servo on pin 9 to the servo object
    setRotateServo(SERVO_DIR_CENTER);
    delay(3000); // 3 초 지연. just delay at first time.
    setRotateServo(SERVO_DIR_LEFT);
    delay(300);
    setRotateServo(SERVO_DIR_RIGHT);
    delay(1000);
}

```

```
setRotateServo(SERVO_DIR_CENTER);
delay(1000);
}

void loop() {
    // for test. wait time.
    delay(3000);

    //
    int retDir=decisionDirection();
    // debug print each dir value;
    Serial.print(g_DistanceDirValue.left);
    Serial.print(" ");
    Serial.print(g_DistanceDirValue.center);
    Serial.print(" ");
    Serial.print(g_DistanceDirValue.right);
    Serial.print(" ");

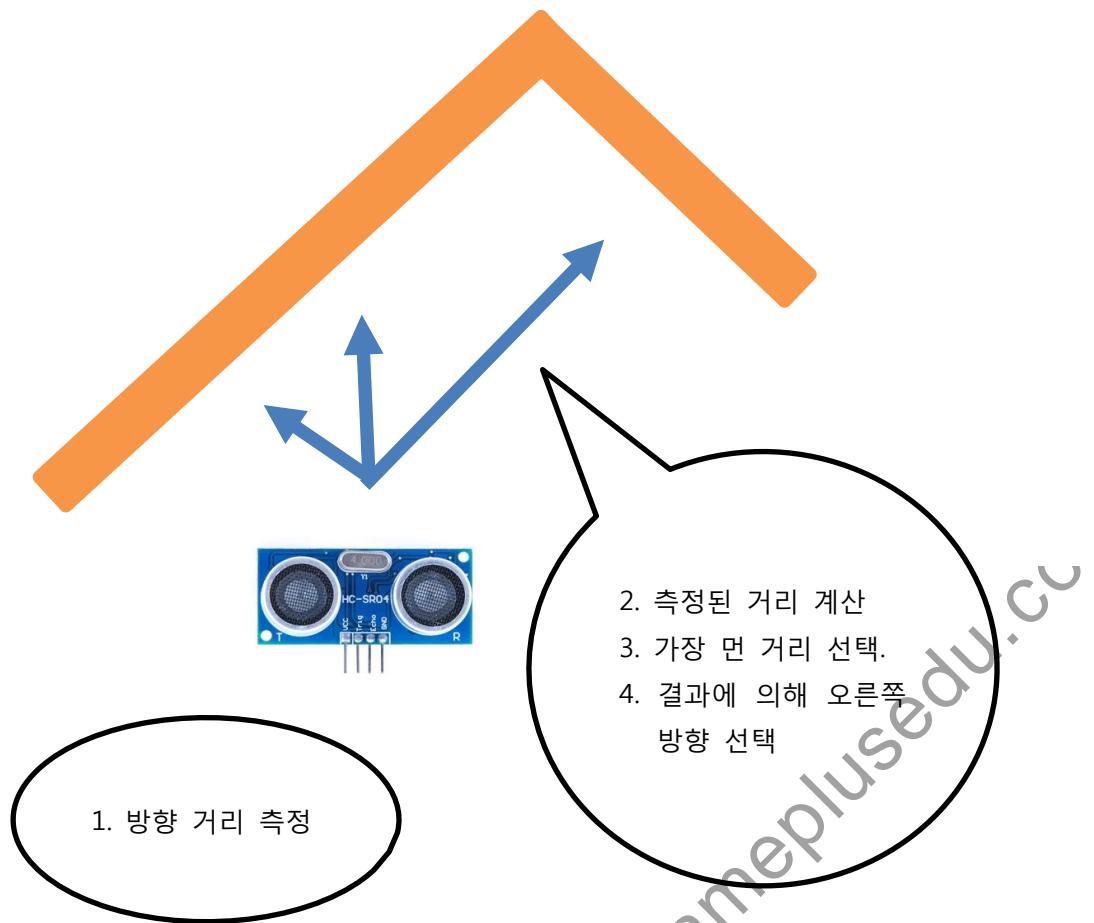
    if(retDir==DIR_DISC_CENTER)

        Serial.println("DIR CENTER");
    if(retDir==DIR_DISC_LEFT)
        Serial.println("DIR LEFT");
    if(retDir==DIR_DISC_RIGHT)
        Serial.println("DIR RIGHT");
}
```

## 10 OBJECT AVOIDANCE DRIVING(장애물 감지 운행)

### 10.1 장애물 감지 방향 전환

왼쪽/중앙/오른쪽 120 도를 3 등분 하여 40 도씩 방향 체크를 합니다. 등분된 각도에서 구한 거리 값을 참조하여, 가장 먼 거리 반환되는 방향으로 회전 하는 로직 입니다. 필요에 따라 3 등분된 방향 또는 4 등분도 가능합니다. 초음파 센서는 거리 값을 측정하기 위해서는 약간의 지연시간(Delay) 필요합니다. 3 등분 정도가 적당한 딜레이 소요됩니다. 좀 더 빠른 반응속도의 거리 측정을 위해서는 여러 가지의 거리 측정 센서가 사용 될 수도 있습니다.



코드: 4wd\_ex\_8

```
//  
// 방향 전환 결정 함수  
//  
int decisionDirection()  
{  
    // 왼쪽 방향의 거리를 측정.  
    g_DistanceDirValue.left =  
    measureDirDistance(SERVO_DIR_LEFT);  
    delay(700); // 딜레이 값은 작게 조절. 700 은 테스트 용도.  
  
    // 오른쪽 방향의 거리를 측정.  
    g_DistanceDirValue.right = measureDirDistance(SERVO_DIR_RIGHT);  
    delay(700);  
  
    // 중앙 방향 거리 측정.  
    g_DistanceDirValue.center = measureDirDistance(SERVO_DIR_CENTER);  
    delay(700);  
  
    //  
    // 가장 먼 거리의 방향으로 결정한다.  
    // c/c++ 코드 규칙에서는 구조체에서의 함수  
    // 정의 및 호출이 가능.  
    int finalDir = g_DistanceDirValue.getProperDir();  
    return finalDir;  
}
```

예제 코드 4wd\_ex\_8 과 4wd\_ex\_2 코드를 변형 및 재구성하여 장애물 감지 운행 코드를 작성합니다.

**ex\_8** 예제는 초음파 센서를 이용한 방향 구하는 코드입니다. **ex\_2** 예제는 DC 모터 방향 전환 코드입니다.

아두이노 우노 R3 부팅, 정지 상태입니다. 1초 후에 정방향 운행 시작합니다. 또는 1초 딜레이 없어도 무방합니다. 정방향 운전 시작과 동시에 초음파 거리 센서 측정 계속 진행됩니다. 방향은 정방향 운행으로 시작합니다. 운행 중 거리값이 최소 거리일 경우에는 후진을 합니다. 후진 후 잠시 정시 시켜 방향을 살펴보고 적절한 방향으로 운행을 시작합니다.

간단히 위의 행동을 반복하게 됩니다.

후진하게 되는 조건을 위한 변수가 추가되며 해당 변수를 사용하기 위한 간단한 로직이 추가 구현되게 됩니다. 운행 로직이 추가 구현 되면서 기존의 함수도 약간 변경될 수 있습니다.

## 10.2 편의 기능 구현

손으로 초음파 센서 근접 작동하여 운행 시작/정지 기능을 구현해 보도록 합니다. 초음파 장애물 감지 운행시 시작과 끝을 리모컨이나 스위치를 사용하지 않고 손으로 초음파 센서의 3 방향을 근접위치로 막아주면 운행 시작, 정지 기능을 구현해 봅니다. 로직은 간단히 3 방향의 거리값 평균이 일정 수치보다 작으면 버튼 역할을 하게 해봅니다.

아래는 수도(Pseudo)코드입니다.

```
// 3 방향의 평균을 구합니다.  
int averageDistance = (LEFT + CENTER + RIGHT) / 3;
```

```
if(averageDistance < 지정된 값 )  
{  
    현재 모드 운행 모드 체크 후 처리해줍니다.  
}
```

### 10.3 운행 모드 동작 QUEUE 개념을 적용해 봅니다.

운행 모드에 관련된 요구 작동들을 정의하고 처리해주는 큐(QUEUE) 로직을 구현해봅니다. 처리 순서는 FIFO로 합니다. 문자열에 명령어를 집어 넣고, 큐의 목록에 있는 요구 항목들을 처리해 줍니다. 프로젝트를 발전시키고 좀더 복잡하고 세밀한 스마트 드라이빙 제어를 위해서는 필수적으로 구현해주어야 하는 부분입니다.

### 10.4 초음파 센서 방향으로 차량 회전

초음파 센서의 패닝(수평축 회전)을 3 방향으로 지정해놓은 상태입니다. 현재 위치에서 정면에서 40 도 방향으로 회전을 시켜서 직진을 하게 합니다. 물론, 대략의 시간 지연을 사용하면 근접하게 맞추어서 코드를 구현할 수도 있습니다. 하지만 해당하는 원리를 알면 코드에 대한 이해와 차후에 여러 변수가 생길 경우 대처하기도 쉬울 것으로 추측됩니다.

그럼, 위에서 계산된 1회전 거리를 공식을 기반으로 각도를 계산해봅니다.

현재 방향을 0 각도로 보았을 때 초음파 센서는 중앙 기준 왼쪽, 오른쪽 40 방향의 거리를 구합니다. 그럼 거의 근접하게 40 도 각도로 차량을 회전 시켜, 직진 시켜주면 오차 및 오류가 적은 운행이 될 수 있습니다.

1 회전에 1 초가 걸린다는 가정하에 계산해 줍니다. 1 회전은 360으로 했을 경우, 40 도는 360 도의  $\frac{1}{9}$  입니다.

$$360 = 40 \times 9$$

그럼 1 회전이 1 초라 가정하면  $1/9 = 0.1111111$ .. 계산됩니다. 물론 DC 모터의 작동 시간 및 제어 보드에의 반응시간, 소모되는 시간은 적용 하지 않았으므로 정확하지는 않습니다. 하지만 대략적인, 아주 개략적인 근사값, 시간을 추측할 수 있습니다.

0.1 초라 는 시간 동안 오른쪽, 왼쪽 회전을 유지시켜주면 해당하는 각도가 나오리라 예상됩니다. 물론 작동 테스트 후 적절히 계산, 또는 수정하여 주도록 합니다. 그리고, 해당 시간 회전시간 경과 후 다시 직진을 하게 해줍니다.

## 10.5 장애물 감지 거리 측정 운행 구현

코드: 4wd\_ex\_8\_3

```
/*
> 초음파 센싱 거리 감지 운행.
> 동작 시작/종료 센싱 처리.
> 초음파 거리 평균값 적용 안함.
> ver 1.5
*/
#include <Servo.h> // for servo motor.
//#include <NewPing.h> // for Ultrasonic sensor.
//#include <IRremote.h> //

#define HIGH 1
#define LOW 0

/*
* 문자열 String 클래스를 이용한 QUEUE 구현. FIFO.
* 1 문자의 값을 판단해 운행.
* "문자열;" 한개의 커맨드 처리.
*/
```

```

/* ex) where to go command is "?"
*/
String cmdSzQueue;

//cmdSzQueue += "A";
void cmdEnQueue(const char* cmdString)
{
    cmdSzQueue += cmdString;
}

int isCmdQueue(void)
{
    return cmdSzQueue.length();
}

// 1회 호출시 cmdSzQueue 의 맨 앞의 문자 처리후, 삭제.
unsigned char cmdDeQueue(void)
{
    // assert(isCmdQueue());
    unsigned char /*char*/ cmdByte = cmdSzQueue[0];
    cmdSzQueue.remove(0, 1);
    return cmdByte;
}

// 편맵.
//
// for Ultrasonic module
//
#define TRIGGER_PIN 12 // Trigger Pin
#define ECHO_PIN      13 // Echo Pin

// #define MAX_DISTANCE 250 // distance max value. 2.5m

```

```
//  
// 주의: ENA, ENB 는 PWM 지원 포트에 연결한다.  
//  
#define ENA    6  
#define EN1    7  
#define EN2    3  
  
#define EN3    4  
#define EN4    2  
#define ENB    5  
  
  
#define SERVO_PIN    9  
//  
// Direction for servo motor  
//  
#define SERVO_DIR_CENTER    1  
#define SERVO_DIR_LEFT      2  
#define SERVO_DIR_RIGHT     3  
  
// 길 찾기 방향 종류.  
#define DIR_DISC_CENTER    1  
#define DIR_DISC_LEFT      2  
#define DIR_DISC_RIGHT     3  
  
// 최소 거리값. 충돌방지를 위한 최소 거리. cm  
#define DIR_DISTANCE_ALERT 15  
  
//  
// DC 모터 제어 운행  
// 자동차 진행 방향 정의  
//  
#define CAR_DIR_FW  0 // 전진.
```

```

#define CAR_DIR_BK 1 // 후진.
#define CAR_DIR_LF 2 // 좌회전.
#define CAR_DIR_RF 3 // 우회전
#define CAR_DIR_ST 4 // 정지.

//
// 차량 운행 방향 상태 전역 변수. // 정지 상태.
int g_carDirection = CAR_DIR_ST;

int g_carSpeed = 220; // 최대 속도의 60 % for testing.

//
#define CAR_OPERATE_MODE_ENABLE 1
#define CAR_OPERATE_MODE_DISABLE 0
//
int g_carUserMode = CAR_OPERATE_MODE_DISABLE; // operate
//
#define NO_ECHO 0

void init_ultrasonic_sensor()
{
    pinMode(TRIGGER_PIN, OUTPUT); // 센서 Trig 핀, D12
    pinMode(ECHO_PIN, INPUT); // 센서 Echo 핀, D11

    digitalWrite(TRIGGER_PIN, LOW);
    digitalWrite(ECHO_PIN, LOW);
}

long microsecondsToInches(long microseconds)
{
    // According to Parallax's datasheet for the PING)), there are
    // 73.746 microseconds per inch (i.e. sound travels at 1130 feet per
    // second). This gives the distance travelled by the ping, outbound

```

```

// and return, so we divide by 2 to get the distance of the obstacle.
// See: http://www.parallax.com/dl/docs/prod/acc/28015-PING-v1.3.pdf

// 시간에 대한 값을 인치로 변환.
return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds)
{
    // The speed of sound is 340 m/s or 29 microseconds per centimeter.
    // The ping travels out and back, so to find the distance of the
    // object we take half of the distance travelled.
    // 시간에 대한 값을 센티미터로 변환
    return microseconds / 29 / 2;
}

// global instance ultrasonic sensor.
//NewPing sonar_distance(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

// global instance servo motor.
Servo myservo; // create servo object to control a servo
// a maximum of eight servo objects can be created
//
// get value current distance from HC-SR04 ( ultrasonic )
//
int getCurrDistance()
{
    /*
    int uS = sonar_distance.ping();
    return uS / US_ROUNDTRIP_CM;
    */
    long duration, cm;
}

```

```

digitalWrite(ECHO_PIN, LOW);
digitalWrite(TRIGGER_PIN, LOW);
delayMicroseconds(10);

digitalWrite(TRIGGER_PIN, HIGH);
delayMicroseconds(10); //
digitalWrite(TRIGGER_PIN, LOW);
//
duration = pulseIn(ECHO_PIN, HIGH); // Echo pin: HIGH->Low 간격을
측정
cm = microsecondsToCentimeters(duration); // 거리(cm)로 변환
return (int)cm;

}

// prevent NO_ECHO, prevent 0 cm.
//
int getSafeDistance()
{
    const int nRetry = 3; // modify count if you want..
    int nDist = NO_ECHO;
    //
    // simple code.
    // while(nDist==getCurrDistance() != NO_ECHO) ; // while when
NO_ECHO
    // for debug message.
    do
    {
        nDist = getCurrDistance();
        if (nDist == NO_ECHO)
        {
            Serial.println("HC-SR04 returned NO_ECHO");
        }
    }
}

```

```
        }
    } while (nDist == NO_ECHO);

    // now return safe value.
    if (nDist != NO_ECHO)
        return nDist;
}

// 각 방향에 대한 거리 데이터 보존 구조체.
//  

struct DistanceDir
{
    int left;
    int center;
    int right;
    //  

    int choosedDir;

    int getMax()
    {
        int retMax = center;
        if (center<left)
        {
            retMax = left;
        }
        if (retMax<right)
        {
            retMax = right;
        }
        return retMax;
    }
}
```

```

//  

int getProperDir()  

{  

    choosedDir = DIR_DISC_CENTER; // 기본방향은 정방향.  

    // get max distance.  

    int compareDist = getMax();  
  

    if (compareDist == left)  

        choosedDir = DIR_DISC_LEFT;  

    if (compareDist == right)  

        choosedDir = DIR_DISC_RIGHT;  

    if (compareDist == center)  

        choosedDir = DIR_DISC_CENTER;  

    // return value.  
  

    //  

    return choosedDir;  

}  

//  

//  

bool isAlertForward()  

{  

    return (center < DIR_DISTANCE_ALERT) ? true : false;  

}  
  

//  

bool isClosedAll()  

{  

    // 사방의 거리 평균이 4~5cm 미만일 경우 가렸다고 판단. 정지.  

    int average = (left + center + right) / 3;  

    return average < 5 ? true : false;  

}  

}; // end of struct DistanceDir

```

```
struct DistanceDir g_DistanceDirValue;
//  
// measure distance  
//  
int measureDirDistance(int iDir)  
{  
    // servo motor rotate  
    setRotateServo(iDir);  
    // delay(10); //wait for servo.  
    int currDistance = getSafeDistance();  
    return currDistance;  
}  
  
//  
// direction  
//  
int decisionDirection()  
{  
    g_DistanceDirValue.left = measureDirDistance(SERVO_DIR_LEFT);  
    delay(100); //  
    g_DistanceDirValue.right = measureDirDistance(SERVO_DIR_RIGHT);  
    delay(100);  
    g_DistanceDirValue.center = measureDirDistance(SERVO_DIR_CENTER);  
    delay(100);  
  
    int finalDir = g_DistanceDirValue.getProperDir();  
    return finalDir;  
}  
  
//  
// int iDir : 방향 설정 변수  
//
```

```
void setRotateServo(int iDir)
{
#define DEG_CENTER 90 // 85~90 도 정도 설정합니다.
    // for debug
    //Serial.println("Servo curr " + String(myservo.read()) + " " + String(iDir));

    //
    int tempDeg[4] = { 0, DEG_CENTER, DEG_CENTER + 40,
                      DEG_CENTER - 40 };

    int curDeg = myservo.read(); //
    if (curDeg == tempDeg[iDir])
        return;

    //
    switch (iDir)
    {
        case SERVO_DIR_CENTER:
            myservo.write(DEG_CENTER - 0);
            break;
        case SERVO_DIR_LEFT:
            myservo.write(DEG_CENTER + 40);
            break;
        case SERVO_DIR_RIGHT:
            myservo.write(DEG_CENTER - 40);
            break;
        default:
            // 지정된 함수 인자가 아닐 경우 처리.
            String errMsg = "called int setRotateServo(int ";
            errMsg += iDir;
            errMsg += ")";
            Serial.println(errMsg);
            break;
    }
    // delay(10); 15ms
```

```

int degRange = abs(curDeg - tempDeg[iDir]);
//degRange=(int) ((float)degRange / (float)60.0f)*120.0f;
degRange /= 3;
degRange += 1; //
delay(degRange * 15);
}

int whereToGo(void)
{
    Serial.println("finding route ...");
    int retDir = decisionDirection();
    // debug print each dir value;
    Serial.print(g_DistanceDirValue.left);
    Serial.print(" ");
    Serial.print(g_DistanceDirValue.center);
    Serial.print(" ");
    Serial.print(g_DistanceDirValue.right);
    Serial.print(" ");

    if (retDir == DIR_DISC_CENTER)
        Serial.println("DIR CENTER");
    if (retDir == DIR_DISC_LEFT)
        Serial.println("DIR LEFT");
    if (retDir == DIR_DISC_RIGHT)
        Serial.println("DIR RIGHT");
    return retDir;
}

// initialize system.
void initSystem()
{
    myservo.attach(SERVO_PIN); // attaches the servo on pin 13 to the servo
}

```

```
object
//
delay(100);
setRotateServo(SERVO_DIR_CENTER);
delay(1000); //

init_car_controller_board();
car_update();
//
print_car_info();

init_ultrasonic_sensor();
}

/**/
void controllerCommand(String& szIRCmd)
{
    if (szIRCmd == "+") // speed up
    {
        g_carSpeed += 20;
        g_carSpeed = min(g_carSpeed, 255);
        Serial.print("Speed Up ");
        Serial.println(g_carSpeed);
        car_update();

        return;
    }
    else
    if (szIRCmd == "-") // speed down
    {
        g_carSpeed -= 20;
        g_carSpeed = max(g_carSpeed, 70);
    }
}
```

```
    Serial.print("Speed down ");
    Serial.println(g_carSpeed);
    car_update();

    return;
}

else
if (szIRCmd == "2") // 전진
{
    g_carDirection = CAR_DIR_FW;
    Serial.println("Forward");
    car_update();

    return;
}
else
if (szIRCmd == "5")
{
    g_carDirection = CAR_DIR_ST;
    Serial.println("Stop");
    car_update();

    return;
}
else
if (szIRCmd == "8")
{
    g_carDirection = CAR_DIR_BK;
    Serial.println("Backward");
    car_update();

    return;
}
```

```
    }
    else
    if (szIRCmd == "4")
    {
        g_carDirection = CAR_DIR_LF;
        Serial.println("Left");
        car_update();
        return;
    }
    else
    if (szIRCmd == "6")
    {
        g_carDirection = CAR_DIR_RF;
        Serial.println("Right");
        car_update();
        return;
    }
    else
    if (szIRCmd == "D") // continue delay.
    {
        delay(100); // 0.1 sec
        return;
    }
    else
    if (szIRCmd == "C") // continue delay.
    {
        //g_carDirection =
        car_update();
        return;
    }
    else
    if (szIRCmd == "?") // whereToGo
    {
```

```
int newDirection = whereToGo(); // 길찾기.  
Serial.print("New Direction ");  
Serial.println(newDirection);  
if (newDirection == DIR_DISC_CENTER)  
{  
    Serial.println("Forward ||||| ");  
    cmdEnQueue("2"); // 전방  
  
}  
else  
    //g_carDirection = CAR_DIR_FW;  
if (newDirection == DIR_DISC_LEFT)  
{  
    Serial.println("LEFT <<===== ");  
    //g_carDirection = CAR_DIR_LF;  
    cmdEnQueue("4"); //  
    cmdEnQueue("D"); // delay 100 ms  
    cmdEnQueue("2"); // 전방  
}  
else  
if (newDirection == DIR_DISC_RIGHT)  
{  
    Serial.println("RIGHT =====> >>> ");  
    //g_carDirection = CAR_DIR_RF;  
    cmdEnQueue("6"); // 6  
    cmdEnQueue("D"); // delay 100 ms  
    cmdEnQueue("2"); // 전방
```

```

        }
        //
        print_car_info();
        car_update();
        g_carDirection = CAR_DIR_FW;
        car_update();
        return;
    }

Serial.println(">>");
Serial.print("Unknown cmd ");
Serial.println(szIRCmd);
}

void init_car_controller_board()
{
    pinMode(ENA, OUTPUT); // ENA
    pinMode(EN1, OUTPUT); // EN1
    pinMode(EN2, OUTPUT); // EN2

    pinMode(ENB, OUTPUT); // ENB
    pinMode(EN3, OUTPUT); // EN3
    pinMode(EN4, OUTPUT); // EN4
}

//
// 전후좌우 4 개의 함수는 테스트시
// DC 모터 연결에 맞게 고쳐서 정정해야 합니다.
// DC 모터 연결 (+)(-) 연결 변경하거나 코드를 변경합니다.
void car_forward()
{
    digitalWrite(EN1, LOW);
    digitalWrite(EN2, HIGH);
}

```

```
analogWrite(ENA, g_carSpeed);

digitalWrite(EN3, LOW);
digitalWrite(EN4, HIGH);
analogWrite(ENB, g_carSpeed);

}

void car_backward()
{
    digitalWrite(EN1, HIGH);
    digitalWrite(EN2, LOW);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, HIGH);
    digitalWrite(EN4, LOW);
    analogWrite(ENB, g_carSpeed);
}

// 
void car_left()
{
    int push_old_speed = g_carSpeed;
    // g_carSpeed = 128; // 차후 전압에 의한 속도 차이 체크.
    digitalWrite(EN1, HIGH);
    digitalWrite(EN2, LOW);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, LOW);
    digitalWrite(EN4, HIGH);
    analogWrite(ENB, g_carSpeed);

    g_carSpeed = push_old_speed; // pop
```

```
}

// 
void car_right()
{
    int push_old_speed = g_carSpeed;
    // g_carSpeed = 128; // 차후 전압에 의한 속도 차이 체크.

    digitalWrite(EN1, LOW);
    digitalWrite(EN2, HIGH);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, HIGH);
    digitalWrite(EN4, LOW);
    analogWrite(ENB, g_carSpeed);
    g_carSpeed = push_old_speed; // pop
}

// 
// 
void car_stop()
{
    analogWrite(ENA, 0);
    analogWrite(ENB, 0);
}

// 
// 방향 전환값에 의해 차량 운행.
// 
void car_update()
{
    // %%
    // 현재는 매번 loop 에서 호출되지만 차후에 커맨드가 있을 경우만
```

호출될 수 있도록

```
// 변경해야 합니다.  
if (g_carDirection == CAR_DIR_FW) // 전진  
{  
    car_forward();  
}  
else  
if (g_carDirection == CAR_DIR_BK) // 후진.  
{  
    car_backward();  
}  
else  
if (g_carDirection == CAR_DIR_LF) // 좌회전  
{  
    car_left();  
}  
else  
if (g_carDirection == CAR_DIR_RF) // 우회전  
{  
    car_right();  
}  
else  
if (g_carDirection == CAR_DIR_ST) // 정지.  
{  
    car_stop();  
}  
  
//  
#define CAR_DIR_FW 0 // 전진.  
#define CAR_DIR_BK 1 // 후진.  
#define CAR_DIR_LF 2 // 좌회전.  
#define CAR_DIR_RF 3 // 우회전
```

```

#define CAR_DIR_ST 4 // 정지.

void print_car_info()
{
    String szDir = "#### ERROR ";
    if (g_carDirection == CAR_DIR_FW)
        szDir = "Forward";
    if (g_carDirection == CAR_DIR_BK)
        szDir = "Backward";
    if (g_carDirection == CAR_DIR_LF)
        szDir = "Left";
    if (g_carDirection == CAR_DIR_RF)
        szDir = "Right";
    if (g_carDirection == CAR_DIR_ST)
        szDir = "Stop";

    Serial.println("curr dir " + szDir);
    Serial.println("speed pwm value " + String(g_carSpeed));
}

//  

// return 남은 Queue 크기.
//  

int processingCommand(void)
{
    unsigned char cmd_a_process = 0; // cmd none

    if (isCmdQueue())
    {
        //Serial.print("Queue Size ");
        //Serial.println(isCmdQueue());
        cmd_a_process = cmdDeQueue();
        //Serial.println(isCmdQueue()); // remain queue cmd count
}

```

```

        }

        //

        // 처리할 프로세스가 있다면
        if (cmd_a_process != 0)
        {
            char szTemp[8];
            sprintf(szTemp, "%c", cmd_a_process);
            String szTempCmd(szTemp[0]);
            controllerCommand(szTempCmd);
            car_update();
        }

        return isCmdQueue();
    }

    //

    // 3 방향 모두 근접.
    // 초음파 센서 3 방향 물체에 의해 거리 측정 최소거리인 경우
    //

    bool isOperateModeChange()
    {
        return g_DistanceDirValue.isClosedAll();
    }

    void startSystem()
    {
        g_carUserMode = CAR_OPERATE_MODE_ENABLE;
        g_carDirection = CAR_DIR_ST;
        car_update();
    }

    void breakSystem()
    {
        g_carUserMode = CAR_OPERATE_MODE_DISABLE;
    }
}

```

```

g_carDirection = CAR_DIR_ST;
car_update();
//and etc..
// ..
}

//  

void setup()
{
    Serial.begin(9600);
    //
    initSystem();
    // 정지 상태.
}

void loop()
{
    // 앞부분 거리 측정.
    g_DistanceDirValue.center = measureDirDistance(SERVO_DIR_CENTER);
    //Serial.println(g_DistanceDirValue.center);
    //

    while (processingCommand()>0);

    //
    if (g_DistanceDirValue.isAlertForward()) // alert
    {
        Serial.println("### Alert Forward ## " +
String(g_DistanceDirValue.center));

        car_stop(); // 모터정지
        Serial.println("Stopped Car");
    }
}

```

```
//buzzer(0); // 부저

// 3 방향이 모두 짧은 거리로 판단되는 경우.
// 손으로 3 방향 가까이 가리는 경우 정지, 시작.
if (isOperateModeChange())
{
    if (g_carUserMode == CAR_OPERATE_MODE_DISABLE)
    {
        Serial.println("Starting....");
        startSystem();
        delay(1000); //
    }
    else
    {
        Serial.println("Stopping by USER??");
        breakSystem();
        delay(2500); // 2.5 초 잠시 정지.
    }
}

// 
// 
if (g_carUserMode == CAR_OPERATE_MODE_ENABLE)
{
    cmdEnQueue("5"); // 정지
    // 0.7 초 정지..
    for (int i = 0; i < 7; i++)
        cmdEnQueue("D"); // 100 ms 계속 후진.
    //
    cmdEnQueue("8"); // 후진.
    // 1.1 초동안 정방향 후진.
    for (int i = 0; i < 11; i++)
        cmdEnQueue("D"); // 100 ms 계속 후진.
```

```
cmdEnQueue("5"); // 정지  
cmdEnQueue("D"); // 100 ms 계속 후진.  
  
cmdEnQueue("?"); // 길찾기.  
cmdEnQueue("C"); // continue.  
}  
}  
else  
{  
    //  
}  
}
```

## 10.6 타이머를 이용한 주기적 초음파 거리 측정

SimpleTimer 라이브러리를 이용하여 주기적인 거리 측정을 합니다.

<http://playground.arduino.cc/Code/SimpleTimer> 사이트에서 참조 및 다운로드 후 라이브러리 설치를 합니다. SimpleTimer 예제를 참조하면 아래와 같이 간단히 repeatMe()라는 함수를 1초마다 호출할 수 있습니다.

```
// SimpleTimer 를 사용하기 위해 헤더파일 선언 해줍니다.  
#include <SimpleTimer.h>  
  
// 타이머 객체 전역 변수를 선언해줍니다.  
// the timer object  
SimpleTimer timer;  
  
// 1초마다 호출되는 함수 본체입니다.
```

```

// a function to be executed periodically
void repeatMe() {
    Serial.print("Uptime (s): ");
    Serial.println(millis() / 1000);
}

void setup() {
    Serial.begin(9600);
    timer.setInterval(1000, repeatMe);
}

void loop() {
    // timer class 멤버변수 SimpleTimer::run() 함수를 호출하여
    // 내부 시간 경과를 체크하게 됩니다.
    // 반드시 SimpleTimer::run() 함수는 호출해 주어야
    // 제대로 SimpleTimer 작동됩니다.
    timer.run();
}

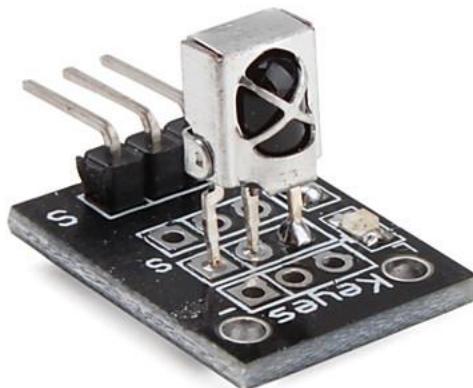
```

손쉽게 사용할 수 있는 타이머 관련 함수 및 기능을 구현할 수 있다면 간단한, 또는 복잡한 프로그램 개발에 도움이 될 수도 있으니 참조하기 바랍니다.

## 11 리모트 컨트롤 운행 제어

원격지에서 적외선 리모트 컨트롤 신호 또는 RF 모듈 등으로 제어를 할 수 있습니다.

### 11.1 IR 신호 수신



IR 수신 모듈 (IR 리모트 컨트롤러 수신 모듈)

IR 리모트 컨트롤러 수신 모듈입니다. IR 수신 센서 Breakout 모듈입니다.



IR 리모트 컨트롤러

## 11.2 리모트 컨트롤러



배터리 장착 시 앞 뒤 구분

## 11.3 IR 리모트 라이브러리

아두이노 공개 소스 IR 라이브러리를 사용하도록 합니다. 리모트 컨트롤 라이브러리 & 예제 코드입니다. 적외선 리모트 컨트롤러에 관련된 모든 참조를 할 수 있습니다.

다운로드 사이트 : <https://github.com/shirriff/Arduino-IRremote>

또는 직접 다운로드 : [Arduino-IRremote-master.zip](#)

Arduino-IRremote 라이브러리 설치 위치는 아두이노 사용자 라이브러리 위치 아래에 압축 해제하여 넣도록 합니다.

## 11.4 IR 리모트 라이브러리 사용시 주의점

적외선 리모트 컨트롤러 라이브러리 사용시 코드에는 아래와 같은 헤더 파일 선언이 필요합니다.

**#include <IRremote.h>**

>> 아두이노 라이브러리 파일 위치 컴파일 찾기 순서

아두이노는 내부적으로 C/C++ 컴파일러를 사용하고 있습니다.

작성되는 코드에서 #include <IRremote.h> 라고 선언하여 사용하는 경우 아두이노 컴파일러는 기본 라이브러리의 디렉터리의 파일부터 찾게 되어 있습니다. 만약 찾는 경우 나머지 사용자 라이브러리의 디렉터리는 찾지 않게 됩니다.

우연히 IRremote.h 파일의 내용이 같을 수는 있지만, 대부분 용도에 맞게 변경, 추가된 사항들이라 다른 목적으로 사용되고 있습니다.

여기에서 작성되는 코드에서 필요한 라이브러리는 아두이노 IRremote 라이브러리의 IRremote.h 파일이지만, 기본 라이브러리 IRremote.h 사용으로 인식되어 컴파일 에러 등의 문제가 있습니다. 우연히 컴파일, 빌드, 업로드가 되더라도

도 목적하는 결과와 전혀 다르게 나올 수 있습니다.

>> 빌드 시 아래와 비슷한 컴파일 에러 메시지가 나올 수 있습니다.

The screenshot shows a terminal window with the following text:

```
컴파일 오류 발생: C:\Users\user\Documents\Arduino\libraries\RobotIRremote\src\IRremoteTools.cpp:5: error: 'TKD2' was not declared in this scope
```

At the bottom of the terminal window, it says "Arduino Uno on COM20".

메시지의 내용

".....Arduino\arduino-1.5.6-r2-windows\arduino-1.5.6-r2\libraries\RobotIRremote\src\IRremoteTools.cpp:5: error: 'TKD2' was not declared in this scope

또는

remotectrl.ino: In function 'void dump(decode\_results\*)':  
remotectrl.ino:96:35: error: 'LG' was not declared in this scope

컴파일 에러 메시지가 보입니다.

IRRemote.h라는 라이브러리 헤더 파일 명칭이 아두이노의 로봇 IR 라이브러리와 중복되어 RobottIRRemote 스케치 기본 라이브러리 빌드 시 나오는 현상입니다.

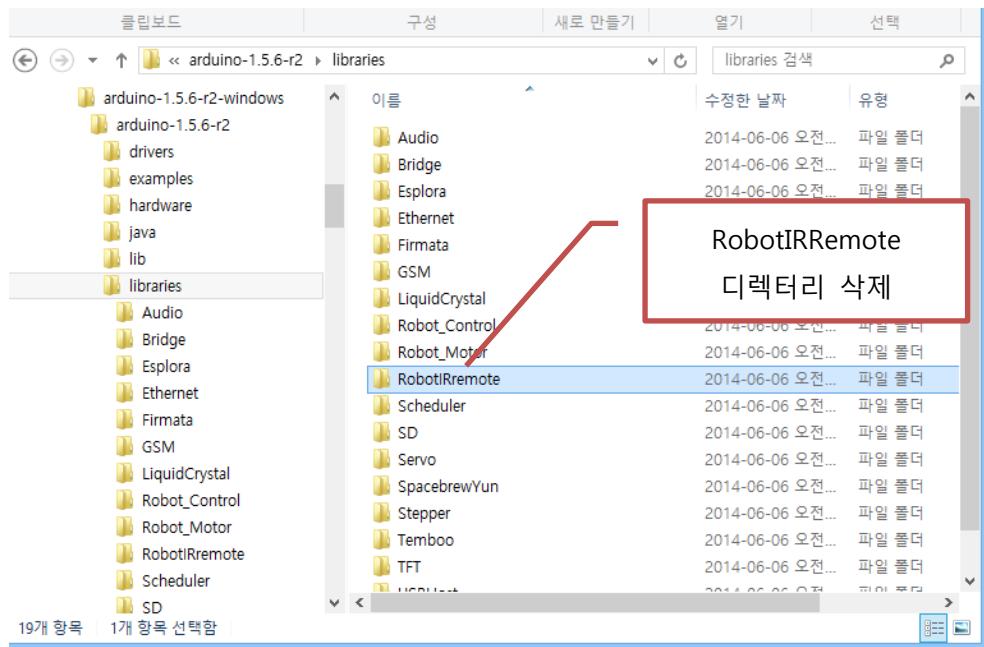
#### 11.4.1 해결 방법

아래와 같이 2 가지 방법이 있습니다. `1>>`, `2>>` 모두 적용하여도 무방하지만, 1 번 방법 권장합니다.

1 >> RobotIRRemote 라이브러리의 헤더파일 중복을 피하기 위해  
RobotIRRemote 디렉터리를 옮기거나 삭제를 합니다.

RotorIRRemote 디렉터리에는 Arduino.cc에서 판매되는 원형 자동차로봇에 사용되는 파일들이 있습니다. 사용하지 않으므로 지워도 무방합니다.

원도우 탐색기를 열어서 아두이노 스케치 프로그램이 있는 디렉터리로 갑니다.



아두이노의 실행 디렉터리의 기본 라이브러리 디렉터리 → RobotIRRemote 라는 디렉터리를 다른 곳으로 옮겨 놓습니다. 즉, 아두이노 스케치 기본 라이브러리에서 제거 또는 다른 디렉터리로 옮겨서 참조하지 않게 합니다.

아두이노 기본 라이브러리의 변동사항(디렉터리 변경, 삭제 등등)이 있는 경우,  
아두이노 프로그램을 재 실행해야 반영됩니다.

2 >> Arduino-IRremote-master 소스 파일 이름을 변경합니다.

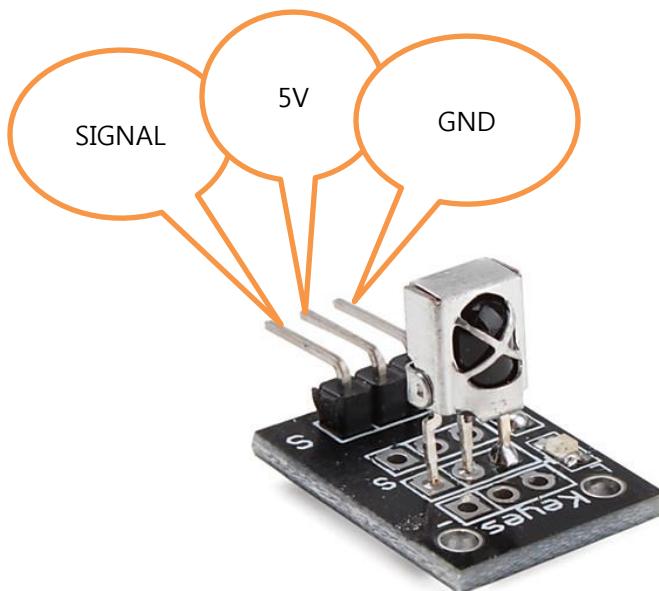
해당 라이브러리 디렉터리의 파일 중 "IRremote.h" 헤더 파일 이름을 변경하여 사용합니다.

IRremote.h → IRremoteModule.h   IRremote.cpp → IRremoteModule.cpp 변경, 그리고 IRRemoteModule.cpp 의 코드 20 라인의 #include "IRremote.h" 부분을 #include "IRremoteModule.h" 로 변경합니다.

그리고 ex\_8\_3 과 다른 모든 예제코드에서의 헤더파일 호출 선언  
#include "IRremote.h" → #include "IRremoteModule.h" 변경하여 사용해야 합니다.

## 11.5 IR 리모트 센서 연결

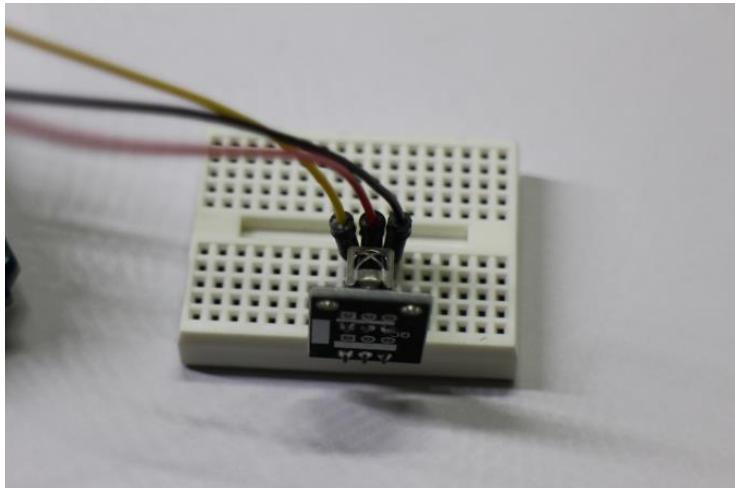
IR 수신 모듈 IR 수신기의 신호 포트와 아두이노 우노의 PWM 포트와 연결하면 됩니다. 또는 아날로그 포트에 연결해도 됩니다.



IR 리모트 수신 모듈 핀 설명

브레드보드에 와이어링 이미지입니다. 연결 와이어링의 색상을 참조하기 바랍니다. 왼쪽부터 SIGNAL / VCC / GND 입니다. 모듈의 중간 핀이 VCC입니다. 주의 하시기 바랍니다.

IR 수신모듈	아두이노
-	GND
+	5V
S	A3



IR 수신 모듈 미니브레드보드에 연결된 모습

IRremote 라이브러리에 포함된 수신기 예제 코드입니다. 아날로그 포트 A3 연결하여 테스트하도록 합니다.

코드: 4wd\_ex\_9

```
/*
IR_remote_tester_and_detector
*/
#include <IRremote.h>

const int irReceiverPin = A3; // 아날로그 포트에도 연결 가능.
IRrecv irrecv(irReceiverPin); //create an IRrecv object
decode_results decodedSignal; //stores results from IR sensor

void setup()
{
```

```

Serial.begin(9600);
pinMode(irReceiverPin, INPUT); //


irrecv.enableIRIn(); // Start the receiver object
}

// Dumps out the decode_results structure.
// Call this after IRrecv::decode()
// void * to work around compiler issue
//void dump(void *v) {
//  decode_results *results = (decode_results *)v

void dump(decode_results *results) {
    int count = results->rawlen;
    if (results->decode_type == UNKNOWN) {
        Serial.print("Unknown encoding: ");
    }
    else if (results->decode_type == NEC) {
        Serial.print("Decoded NEC: ");
    }
    else if (results->decode_type == SONY) {
        Serial.print("Decoded SONY: ");
    }
    else if (results->decode_type == RC5) {
        Serial.print("Decoded RC5: ");
    }
    else if (results->decode_type == RC6) {
        Serial.print("Decoded RC6: ");
    }
    else if (results->decode_type == PANASONIC) {
        Serial.print("Decoded PANASONIC - Address: ");
        Serial.print(results->panasonicAddress, HEX);
        Serial.print(" Value: ");
    }
}

```

```

}

else if (results->decode_type == LG) {
    Serial.print("Decoded LG: ");
}

else if (results->decode_type == JVC) {
    Serial.print("Decoded JVC: ");
}

Serial.print(results->value, HEX);
Serial.print(" (");
Serial.print(results->bits, DEC);
Serial.println(" bits)");
Serial.print("Raw (");
Serial.print(count, DEC);
Serial.print("): ");

for (int i = 0; i < count; i++) {
    if ((i % 2) == 1) {
        Serial.print(results->rawbuf[i] * USECPERTICK, DEC);
    }
    else {
        Serial.print(-(int)results->rawbuf[i] * USECPERTICK, DEC);
    }
    Serial.print(" ");
}
Serial.println("");
}

void loop()
{
    //this is true if a message has been received
    if (irrecv.decode(&decodedSignal) == true)
    {
        if (decodedSignal.bits > 0) // bit > 0 準 6.

```

```

    {
        dump(&decodedSignal);
    }
    irrecv.resume(); // watch out for another message
}
}

```

위의 예제 코드를 실행하여 시리얼 모니터 창을 열어서 입력 받은 수신 값을 기록하면 아래와 같습니다.

Button Name	Value
0	FF6897
1	FF30CF
2	FF18E7
3	FF7A85
4	FF10EF
5	FF38C7
6	FF5AA5
7	FF42BD
8	FF4AB5
9	FF52AD
100+	FF9867
200+	FFB04F
-	FFE01F
+	FFA857
EQ	FF906F
<<	FF22DD
>>	FF02FD
>	FFC23D
CH-	FFA25D
CH	FF629D
CH+	FFE21D

리모트 컨트롤러 수신 값을 반영하여 기본적인 DC 모터 제어를 해보도록 합니다.

## 11.6 리모트 컨트롤러 운행 예제 코드

리모트 컨트롤러의 버튼에 방향 지정을 하여 운행을 합니다.

아래의 그림을 참조하여 버튼 코드와 프로그램 코드에서 기능을 연관시켜서 프로그래밍하도록 해봅니다.



리모트 컨트롤러에서 사용할 기능

키트에 적용되는 기술 구현의 기본적인 동작 기능은 모두 리모트 컨트롤러의 동작 명칭과 동일하게 사용될 예정입니다. 리모트 버튼에 의한 동작 기능 함수 부분은 앞으로 진행될 기술 구현의 기초 코드로 사용됩니다. 전/후/좌/우 등에 관련된 동작 명령 구축과 행동 패턴의 기초 항목으로 사용됩니다.

코드: 4wd\_ex\_10

```
#include <IRremote.h>

// 자동차 진행 방향 정의
// 
#define CAR_DIR_FW 0 // 전진.
#define CAR_DIR_BK 1 // 후진.
#define CAR_DIR_LF 2 // 좌회전.
#define CAR_DIR_RF 3 // 우회전
#define CAR_DIR_ST 4 // 정지.

// 차량 운행 방향 상태 전역 변수.
int g_carDirection = CAR_DIR_FW; //
int g_carSpeed = 153; // 최대속도의 60 퍼센트

// 리모트 컨트롤러 관련 전역 변수.
const int irReceiverPin = A3;

IRrecv irrecv(irReceiverPin); //create an IRrecv object
decode_results decodedSignal; //stores results from IR sensor
```

```

void init_IRreceiverModule()
{
    pinMode(irReceiverPin, INPUT); //
    irrecv.enableIRIn(); // Start the receiver object
}

struct IRvalueData
{
    String name;
    unsigned long value; //
};

IRvalueData irData[21]=
{
    { "0", 0xFF6897 },
    { "1", 0xFF30CF },
    { "2", 0xFF18E7 },
    { "3", 0xFF7A85 },
    { "4", 0xFF10EF },
    { "5", 0xFF38C7 },
    { "6", 0xFF5AA5 },
    { "7", 0xFF42BD },
    { "8", 0xFF4AB5 },
    { "9", 0xFF52AD },
    { "100+", 0xFF9867 },
    { "200+", 0xFFB04F },
    { "- ", 0FFE01F },
    { "+ ", 0xFFA857 },
    { "EQ", 0xFF906F },
    { "<<", 0xFF22DD },
    { ">>", 0xFF02FD },
    { ">|", 0xFFC23D },
    { "CH-", 0xFFA25D },
}

```

```

    { "CH", 0xFF629D },
    { "CH+", 0xFFE21D }
};

//



String decode_IRvalue(unsigned long irValue)
{
    for (int i = 0; i < 21;i++)
    {
        if (irData[i].value==irValue)
        {
            return irData[i].name;
        }
    }
    Serial.println("Not Defined.");
    return String("key Value None");
}

void controllerByIRCommand(String& szIRCmd)
{
    if (szIRCmd == "2") // 전진
    {
        g_carDirection = CAR_DIR_FW;
    }
    else
    if (szIRCmd == "5")
    {
        g_carDirection = CAR_DIR_ST;
    }
    else
    if (szIRCmd == "8")
    {
        g_carDirection = CAR_DIR_BK;
    }
}

```

```

        }

    else
        if (szIRCmd == "4")
    {
        g_carDirection = CAR_DIR_LF;
    }
    else
        if (szIRCmd == "6")
    {
        g_carDirection = CAR_DIR_RF;
    }
}

//



// IR 수신값 처리.
//


void update_IRreceiverModule()
{
    // IR 수신값이 있는지 판단.
    if (irrecv.decode(&decodedSignal) == true)
    {
        if (decodedSignal.bits > 0) // bit > 0 일 경우에만 수신값 체크.
        {
            String zRecvCmd=decode_IRvalue(decodedSignal.value);
            controllerByIRCommand(szRecvCmd);
        }
        irrecv.resume(); // watch out for another message
    }
}

//


// 주의: ENA, ENB 는 PWM 지원 포트에 연결한다.

```

```
//  
#define ENA    6  
#define EN1    7  
#define EN2    3  
  
#define EN3    4  
#define EN4    2  
#define ENB    5  
  
  
void init_car_controller_board()  
{  
    pinMode(ENA, OUTPUT); // ENA  
    pinMode(EN1, OUTPUT); // EN1  
    pinMode(EN2, OUTPUT); // EN2  
  
    pinMode(ENB, OUTPUT); // ENB  
    pinMode(EN3, OUTPUT); // EN3  
    pinMode(EN4, OUTPUT); // EN4  
}  
  
void car_forward()  
{  
    digitalWrite(EN1, LOW);  
    digitalWrite(EN2, HIGH);  
    analogWrite(ENA, g_carSpeed);  
  
    digitalWrite(EN3, LOW);  
    digitalWrite(EN4, HIGH);  
    analogWrite(ENB, g_carSpeed);  
}  
  
void car_backward()
```

```
{  
    digitalWrite(EN1, HIGH);  
    digitalWrite(EN2, LOW);  
    analogWrite(ENA, g_carSpeed);  
  
    digitalWrite(EN3, HIGH);  
    digitalWrite(EN4, LOW);  
    analogWrite(ENB, g_carSpeed);  
}  
  
void car_left()  
{  
    digitalWrite(EN1, HIGH);  
    digitalWrite(EN2, LOW);  
    analogWrite(ENA, g_carSpeed);  
  
    digitalWrite(EN3, LOW);  
    digitalWrite(EN4, HIGH);  
    analogWrite(ENB, g_carSpeed);  
  
}  
  
void car_right()  
{  
    digitalWrite(EN1, LOW);  
    digitalWrite(EN2, HIGH);  
    analogWrite(ENA, g_carSpeed);  
  
    digitalWrite(EN3, HIGH);  
    digitalWrite(EN4, LOW);  
    analogWrite(ENB, g_carSpeed);  
}
```

```
void car_stop()
{
    analogWrite(ENA, 0);
    analogWrite(ENB, 0);
}

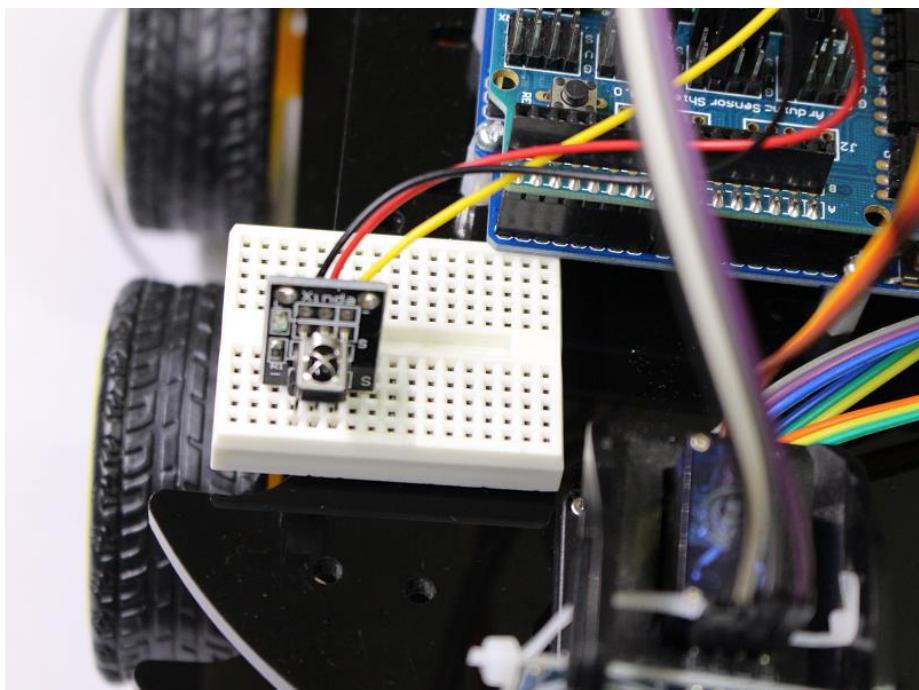
// 방향 전환값에 의해 차량 운행.
//

void car_update()
{
    if (g_carDirection == CAR_DIR_FW) // 전진
    {
        car_forward();
    }
    else
    if (g_carDirection == CAR_DIR_BK) // 후진.
    {
        car_backward();
    }
    else
    if (g_carDirection == CAR_DIR_LF) // 좌회전
    {
        car_left();
    }
    else
    if (g_carDirection == CAR_DIR_RF) // 우회전
    {
        car_right();
    }
    else
    if (g_carDirection == CAR_DIR_ST) // 정지.
```

```
{  
    car_stop();  
}  
}  
  
// 부팅 후 1회 실행되는 함수. 초기화 함수. Setup()  
void setup()  
{  
    //  
    Serial.begin(9600);  
    //  
    init_IRreceiverModule();  
    init_car_controller_board();  
}  
  
// 계속 실행되는 함수. Loop()  
void loop()  
{  
    car_update();  
    update_IRreceiverModule();  
}
```

## 11.7 리모트 수신 모듈 배치(미니브레드보드 사용)

상판 미니브레드보드에 IR 리시버 모듈을 배치합니다.



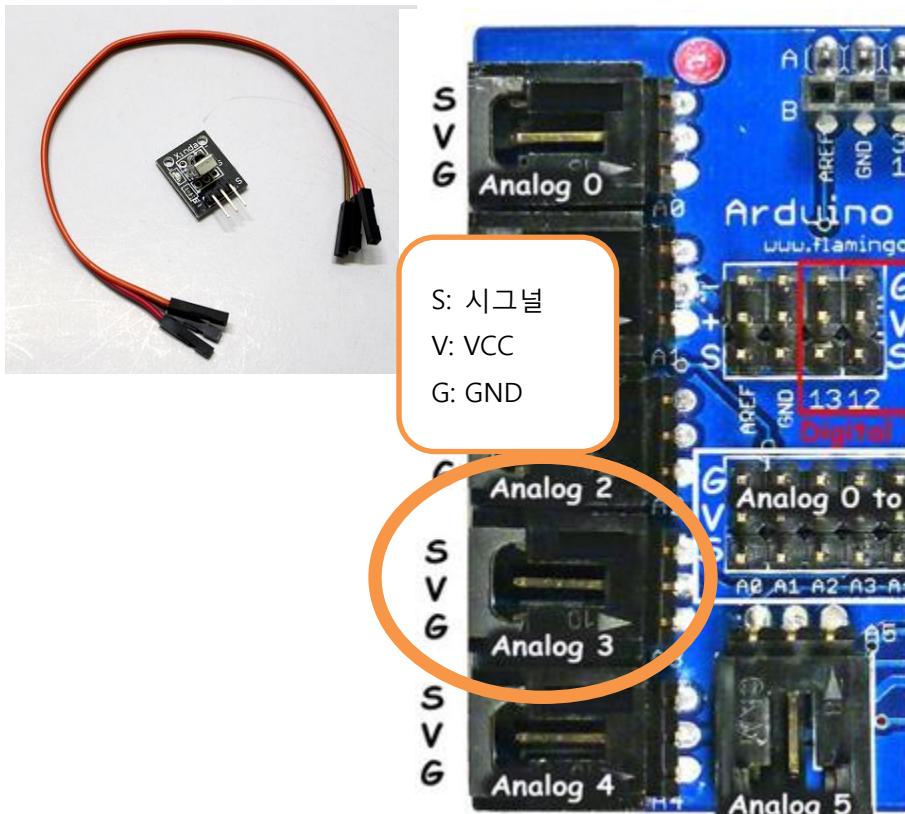
예제 키트에서는 브레드보드 점퍼선으로 위의 참조 이미지처럼 연결 합니다. 소량이지만 브레드보드 일반 점퍼선도 필요합니다. 또는 기본 제공되는 뒤풍 케이블(F to F)로 바로 연결하여 적절한 위치에 배치해도 괜찮습니다.

- 우노 센서 실드 GND 와 5V 를 미니 브레드보드에 연결
- 미니브레드보드의 GND 와 5V 와 IR 리모트 리시버모듈의 GND, 5V 연결.
- IR 리모트 리시버 모듈의 시그널 포트와 센서실드의 A3 연결
- 미니브레드보드의 GND 와 5V 와 L298N 의 GND 와 5V 연결

## 11.8 센서 실드와 뒤풋 케이블 사용하는 경우

센서 실드의 A3 3 포트에 바로 연결하여 사용하는 방법입니다.

- 뒤풋 Female To Female 3 가닥을 분리
- 센서 실드 V4 의 왼쪽 센서 전용 포트와 연결
- 뒤풋 케이블의 핀 간격도 2.54mm 이므로 정확히 일치함



## 11.9 리모트 컨트롤러 운행 코드에 속도제어 추가

코드: 4wd\_ex\_10 번 예제 코드에 Speed 업/다운 코드를 추가해봅니다. 리모트 컨트롤러의 "+", "-" 버튼에 속도 증/감 코드가 적용되고 있습니다. 증/감 코드는 g\_carSpeed 변수의 값을 조절하고 있습니다. g\_carSpeed 변수의 값은 PWM 설정에 사용되고 있습니다.

코드: 4wd\_ex\_10\_3

```
/*
리모트 컨트롤 운행
Ver 0.3 기초 코드.

*/
#include <IRremote.h>

// 자동차 진행 방향 정의
//
#define CAR_DIR_FW 0 // 전진.
#define CAR_DIR_BK 1 // 후진.
#define CAR_DIR_LF 2 // 좌회전.
#define CAR_DIR_RF 3 // 우회전
#define CAR_DIR_ST 4 // 정지.

//
// 차량 운행 방향 상태 전역 변수. // 정지 상태.
int g_carDirection = CAR_DIR_ST; //

int g_carSpeed = 200; // 최대 속도의 78 % for testing.

// 리모트 컨트롤러 관련 전역 변수.
const int irReceiverPin = A3;
IRrecv irrecv(irReceiverPin); //create an IRrecv object
```

```
decode_results decodedSignal; //stores results from IR sensor

void init_IRreceiverModule()
{
    pinMode(irReceiverPin, INPUT); //
    irrecv.enableIRIn(); // Start the receiver object
}

struct IRvalueData
{
    String name;
    unsigned long value;
};

IRvalueData irData[21] =
{
    { "0", 0xFF6897 },
    { "1", 0xFF30CF },
    { "2", 0xFF18E7 },
    { "3", 0xFF7A85 },
    { "4", 0xFF10EF },
    { "5", 0xFF38C7 },
    { "6", 0xFF5AA5 },
    { "7", 0xFF42BD },
    { "8", 0xFF4AB5 },
    { "9", 0xFF52AD },
    { "100+", 0xFF9867 },
    { "200+", 0xFFB04F },
    { "-", 0FFE01F },
    { "+", 0FFA857 },
    { "EQ", 0FF906F },
    { "<<", 0FF22DD },
```

```

    { ">>", 0xFF02FD },
    { ">|", 0xFFC23D },
    { "CH-", 0xFFA25D },
    { "CH", 0xFF629D },
    { "CH+", 0FFE21D }
};

// Dumps out the decode_results structure.
// Call this after IRrecv::decode()
// void * to work around compiler issue
//void dump(void *v) {
//  decode_results *results = (decode_results *)v

void dump(decode_results *results) {
    int count = results->rawlen;
    if (results->decode_type == UNKNOWN) {
        Serial.print("Unknown encoding: ");
    }
    else if (results->decode_type == NEC) {
        Serial.print("Decoded NEC: ");
    }
    else if (results->decode_type == SONY) {
        Serial.print("Decoded SONY: ");
    }
    else if (results->decode_type == RC5) {
        Serial.print("Decoded RC5: ");
    }
    else if (results->decode_type == RC6) {
        Serial.print("Decoded RC6: ");
    }
    else if (results->decode_type == PANASONIC) {
        Serial.print("Decoded PANASONIC - Address: ");
    }
}

```

```

        Serial.print(results->panasonicAddress, HEX);
        Serial.print(" Value: ");
    }

    else if (results->decode_type == LG) {
        Serial.print("Decoded LG: ");
    }

    else if (results->decode_type == JVC) {
        Serial.print("Decoded JVC: ");
    }

    Serial.print(results->value, HEX);
    Serial.print(" (");
    Serial.print(results->bits, DEC);
    Serial.println(" bits)");
    Serial.print("Raw (");
    Serial.print(count, DEC);
    Serial.print("): ");

    for (int i = 0; i < count; i++) {
        if ((i % 2) == 1) {
            Serial.print(results->rawbuf[i] * USECPERTICK, DEC);
        }
        else {
            Serial.print(-(int)results->rawbuf[i] * USECPERTICK, DEC);
        }
        Serial.print(" ");
    }
    Serial.println("");
}

// String decode_IRvalue(unsigned long irValue)
{
    for (int i = 0; i < 21; i++)

```

```

{
    if (irData[i].value == irValue)
    {
        return irData[i].name;
    }
}
Serial.println("Not Defined.");
return String("key Value None");
}

void controllerByIRCommand(String& szIRCmd)
{
    if (szIRCmd == "+") // speed up
    {
        g_carSpeed += 20;
        g_carSpeed = min(g_carSpeed, 255);
        Serial.print("Speed Up ");
        Serial.println(g_carSpeed);
    }
    else
    if (szIRCmd == "-") // speed down
    {
        g_carSpeed -= 20;
        g_carSpeed = max(g_carSpeed, 70);

        Serial.print("Speed down ");
        Serial.println(g_carSpeed);
    }
    else
    if (szIRCmd == "2") // 전진
    {
        g_carDirection = CAR_DIR_FW;
        Serial.println("Forward");
    }
}

```

```
        }
    else
        if (szIRCmd == "5")
    {
        g_carDirection = CAR_DIR_ST;
        Serial.println("Stop");
    }
    else
        if (szIRCmd == "8")
    {
        g_carDirection = CAR_DIR_BK;
        Serial.println("Backward");
    }
    else
        if (szIRCmd == "4")
    {
        g_carDirection = CAR_DIR_LF;
        Serial.println("Left");
    }
    else
        if (szIRCmd == "6")
    {
        g_carDirection = CAR_DIR_RF;
        Serial.println("Right");
    }
}

//  

// IR 수신값 처리.  

//  

void update_IRreceiverModule()
{
    // IR 수신값이 있는지 판단.
```

```

if (irrecv.decode(&decodedSignal) == true)
{
    if (decodedSignal.bits > 0) // bit > 0 일 경우에만 수신값 체크.
    {
        Serial.println(decodedSignal.value);
        //dump(&decodedSignal);
        String szRecvCmd = decode_IRvalue(decodedSignal.value);
        controllerByIRCommand(szRecvCmd);
    }
    irrecv.resume(); // watch out for another message
}

// 주의: ENA, ENB 는 PWM 지원 포트에 연결한다.
//

#define ENA    6
#define EN1   7
#define EN2   3

#define EN3   4
#define EN4   2
#define ENB   5

void init_car_controller_board()
{
    pinMode(ENA, OUTPUT); // ENA
    pinMode(EN1, OUTPUT); // EN1
    pinMode(EN2, OUTPUT); // EN2

    pinMode(ENB, OUTPUT); // ENB
}

```

```
pinMode(EN3, OUTPUT); // EN3
pinMode(EN4, OUTPUT); // EN4
}

//
// 전후좌우 4 개의 함수는 테스트시
// DC 모터 연결에 맞게 고쳐서 정정해야 합니다.
// DC 모터 연결 (+)(-) 연결 변경하거나 코드를 변경합니다.
//
void car_forward()
{
    digitalWrite(EN1, LOW);
    digitalWrite(EN2, HIGH);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, LOW);
    digitalWrite(EN4, HIGH);
    analogWrite(ENB, g_carSpeed);
}

void car_backward()
{
    digitalWrite(EN1, HIGH);
    digitalWrite(EN2, LOW);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, HIGH);
    digitalWrite(EN4, LOW);
    analogWrite(ENB, g_carSpeed);
}

}
```

```
void car_left()
{
    digitalWrite(EN1, HIGH);
    digitalWrite(EN2, LOW);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, LOW);
    digitalWrite(EN4, HIGH);
    analogWrite(ENB, g_carSpeed);
}

//  

void car_right()
{
    digitalWrite(EN1, LOW);
    digitalWrite(EN2, HIGH);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, HIGH);
    digitalWrite(EN4, LOW);
    analogWrite(ENB, g_carSpeed);
}

//  

//  

void car_stop()
{
    analogWrite(ENA, 0);
    analogWrite(ENB, 0);
}

//
```

```
// 방향 전환값에 의해 차량 운행.  
//  
void car_update()  
{  
    // %%  
    // 현재는 매번 loop 에서 호출되지만 차후에 커맨드가 있을 경우만 호출될  
    // 수 있도록  
    // 변경해야 합니다.  
    if (g_carDirection == CAR_DIR_FW) // 전진  
    {  
        car_forward();  
    }  
    else  
    if (g_carDirection == CAR_DIR_BK) // 후진.  
    {  
        car_backward();  
    }  
    else  
    if (g_carDirection == CAR_DIR_LF) // 좌회전  
    {  
        car_left();  
    }  
    else  
    if (g_carDirection == CAR_DIR_RF) // 우회전  
    {  
        car_right();  
    }  
    else  
    if (g_carDirection == CAR_DIR_ST) // 정지.  
    {  
        car_stop();  
    }  
}
```

```
void print_car_info()
{
    Serial.println("direction value " + g_carDirection);
    Serial.println("speed pwm value "+g_carSpeed);
}

// 부팅 후 1회 실행되는 함수. 초기화 함수. Setup()
void setup()
{
    //
    Serial.begin(9600);
    //
    init_IRreceiverModule();
    init_car_controller_board();

    print_car_info();
}

void loop()
{
    car_update();
    update_IRreceiverModule();
}
```

## 12 라인 추적 운행 (LINE TRACKING)

### 12.1 라인 추적 운행 소개

바닥의 유도선 또는 해당 높이의 유도선을 따라 운행합니다. 유도라인 운행 또는 라인 추적 운행 라인 흐름 운행 등의 용어가 사용됩니다. 본 키트는 바닥의 유도선 추적 운행을 하도록 구성 되어 있습니다.

### 12.2 라인 추적 운행 준비

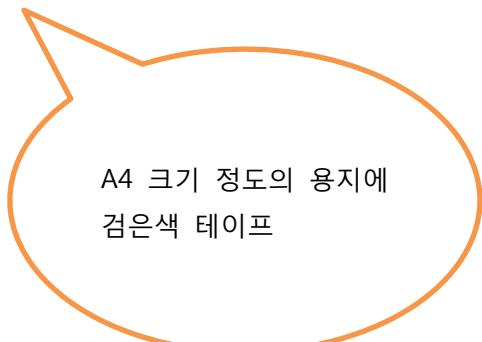
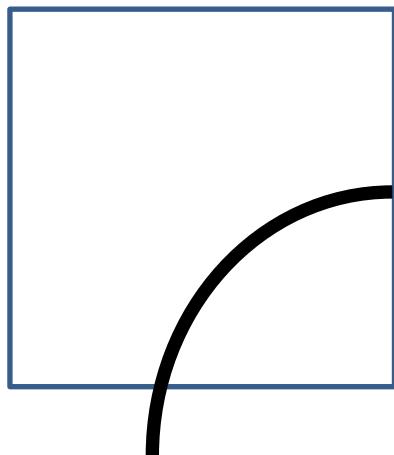
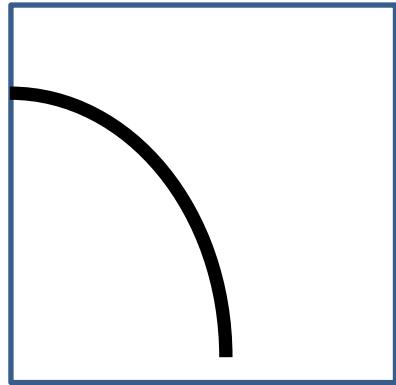
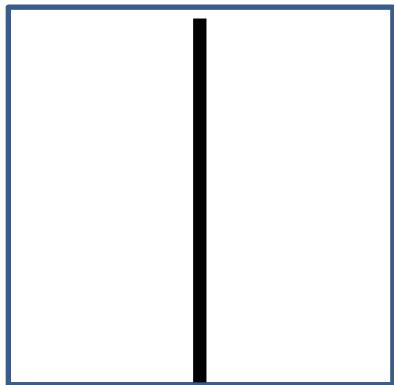
#### 12.2.1 트랙 라인 설치

검은색 흰색을 구분하는 모듈을 사용하므로 절연테이프를 바닥에 붙입니다. 절연 테이프가 아니더라도 검은색, 흰색 구분할 정도의 색상이면 가능합니다. 바닥에 설치될 검은색 라인의 두께 2cm 정도의 넓이를 가져야 합니다. 라인 트레이서의 배치에 따라 두께는 조절될 수 있습니다.



라인 추적 모듈 테스트에 필요한 A4 크기의 용지 몇 장 정도 필요합니다. 절연 테이프를 A4 용지에 붙여서 트랙을 만들어서 기초 테스트를 하게 됩니다. 기본 테스트를 위해 A4 용지 또는 흰색 넓은 종이에 검정색 절연 테이프를 붙입니다. 붙이는 방법은 직진 좌, 우 회전, 곡선 등의 원하는 운행 환경에 붙이도록 합니다.

최적의 테스트 환경은 넓은 공간에 흰색 바탕에 검은색 유도선을 부착합니다.

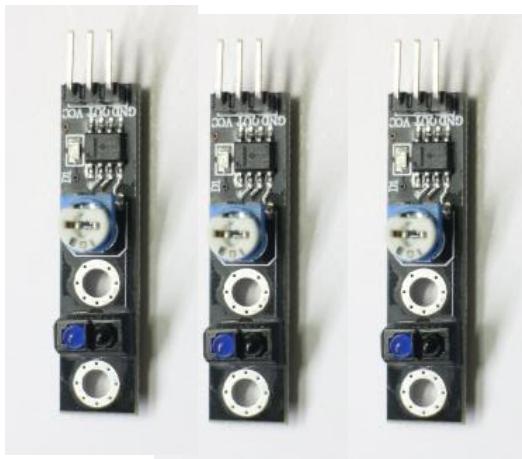


A4 크기 정도의 용지에  
검은색 테이프

필요에 따라 여러 형태의 곡선으로 만들어서 테스트 및 기본 추적 운행 코드를 바탕으로 프로그래밍을 하도록 합니다.

## 12.3 라인 트레이서 모듈 기초 정보

라인 추적 모듈 TCRT5000 모듈 3 개를 사용합니다.



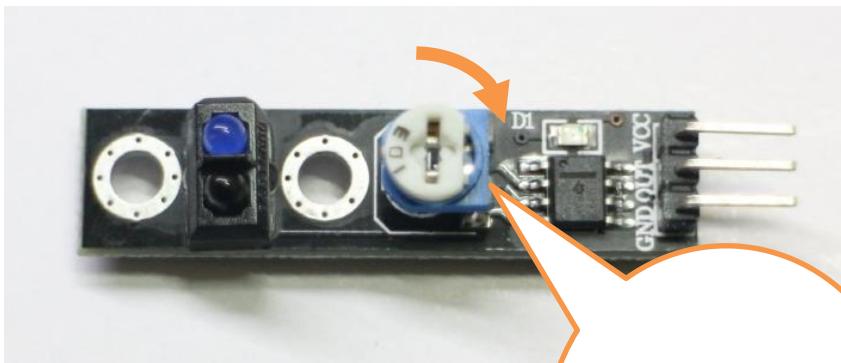
프로그래밍 하는 방식에 따라 3 개 이상 사용할 수도 있고, 2 개도 사용 가능합니다. 본 키트에 적용되는 예제는 3 개의 모듈을 사용하는 방식으로 기술 되어 있습니다.

### 12.3.1 라인센서(TCRT5000)모듈 인식 거리

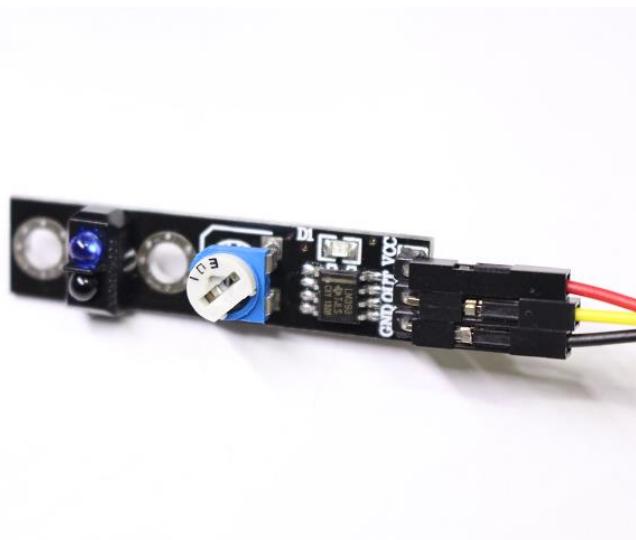
본 예제에 사용되는 TCRT500 모듈의 인식 거리는 1 센티 내외, 최대 1.5 센티 정도입니다. 새시 차량 하단부에 TCRT5000 모듈 장착할 경우 바닥에서부터의 센서 높이는 1 센티 내외가 적당합니다.

TCRT500 모듈의 가변저항을 아래의 그림처럼 약간 돌려주어 거리 조절을 해줍니다.

정확한 센싱 거리는 12mm 이지만 가변저항을 돌려서 조절하는 경우 좀더 짧게 또는 길게 조절 가능하도록 되어 있습니다.

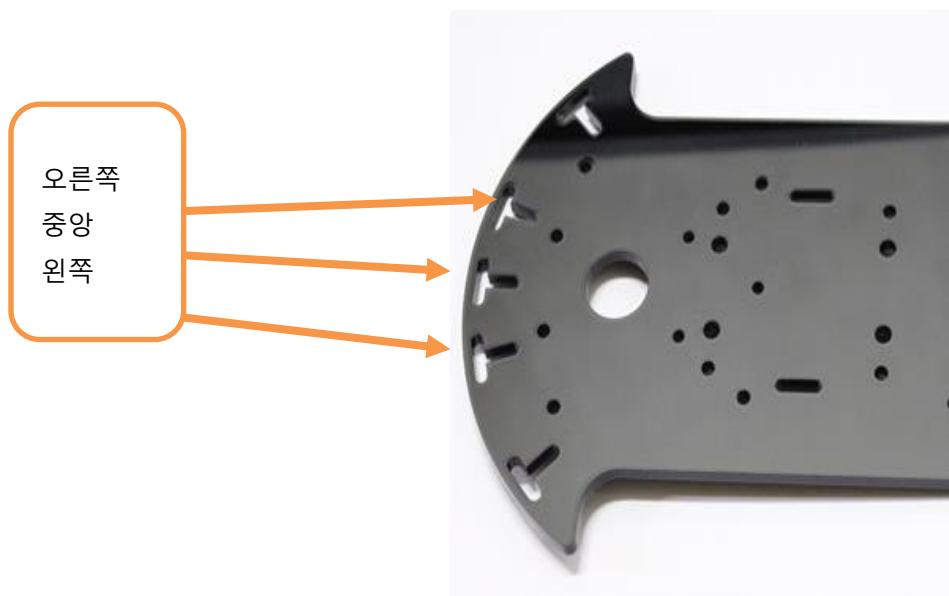


가변저항을 약간  
돌려서 거리 조  
절을 합니다.



## 12.4 라인 트레이서 모듈 장착

장착 위치는 하단 셰시의 앞쪽 밑으로 장착을 하게 됩니다.

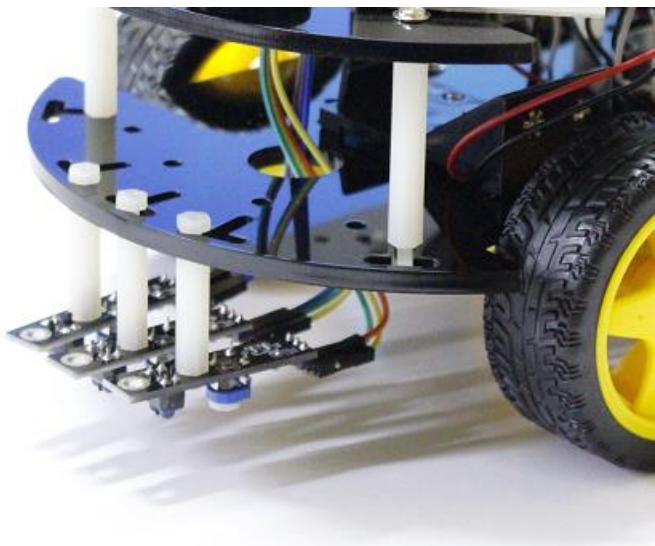


라인트레이서 고정 기둥 위치

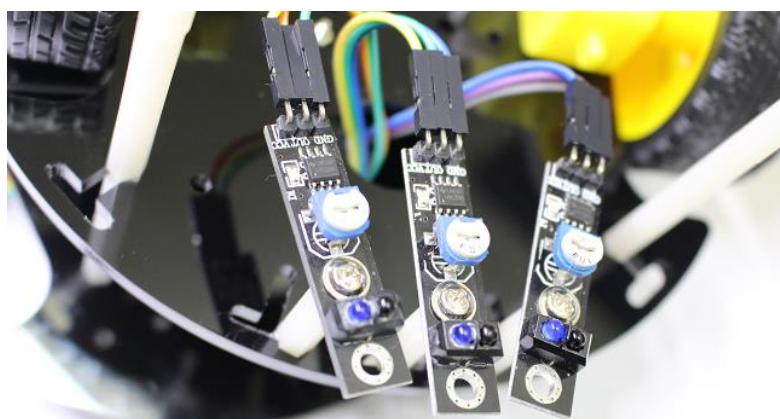


라인 트레이서 모듈 고정 시 사용되는 부품

하단 새시에 장착된 그림입니다.



라인트레이서 모듈 장착된 모습



장착 후 바닥에서 본 모습

## 12.5 라인 추적 운행 기초 원리

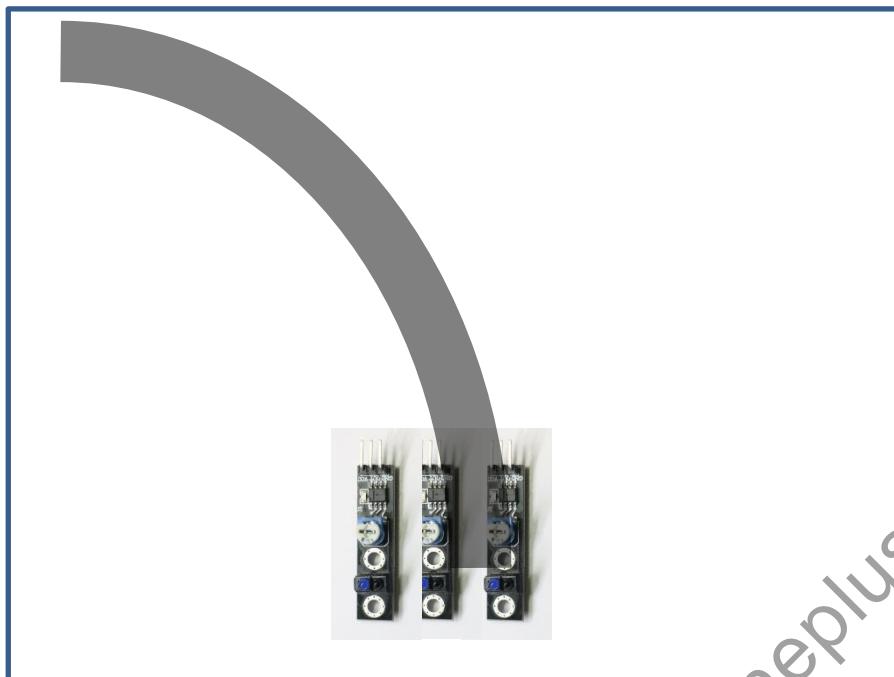
라인 추적 운행을 하기 위해 3 개 정도의 TCRT5000 모듈을 불입니다. 왼쪽, 중앙, 오른쪽 3 개의 모듈로부터 값을 가져옵니다. 모듈에 따라 아날로그 값일 수도 있으며, 디지털 값 (0 or 1) 일 경우도 있습니다.

본 예제에 사용되는 모듈의 반환값은 디지털이며 흰색 바닥일 경우 **0**

검정색 트랙인 경우에는 **1** 이 반환됩니다.

왼쪽 모듈의 값에서 라인 추적 반응이 있으면 왼쪽으로 방향 조절을 합니다. 좌측으로 방향 조절은 라인 추적 모듈의 값이 0 일 때까지 합니다.

동일하게 오른쪽 모듈의 값에서 라인 추적 반응이 있으면 오른쪽으로 방향 조절을 합니다. 역시 오른쪽 라인 추적 모듈의 값이 0 으로 될 때까지 회전합니다.



## 12.6 아두이노 우노 R3 제어 보드와 연결

기존 포트는 거의 사용 중이므로 아날로그 포트를 사용합니다. A0, A1, A2 포트를 사용하도록 합니다.

왼쪽 모듈

TCRT5000 모듈 왼쪽	아두이노
VCC	5V
OUT	A0
GND	GND

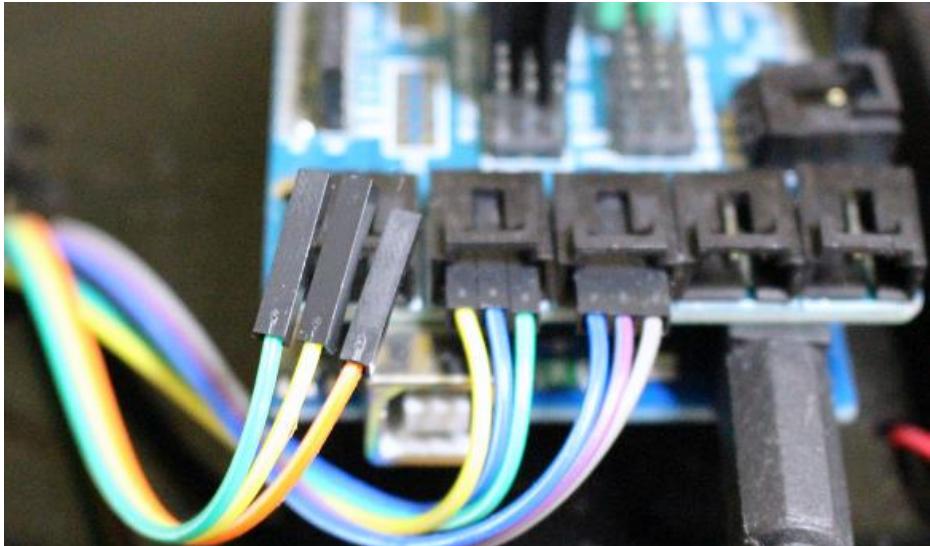
중앙 모듈

TCRT5000 모듈 중앙	아두이노
VCC	5V
OUT	A1
GND	GND

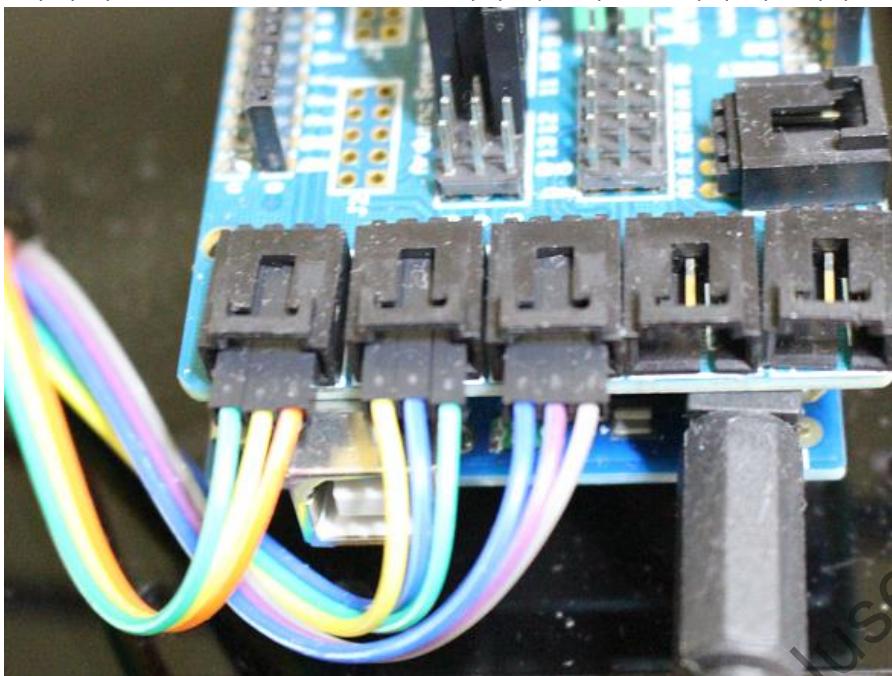
오른쪽 모듈

TCRT5000 모듈 오른쪽	아두이노
VCC	5V
OUT	A2
GND	GND

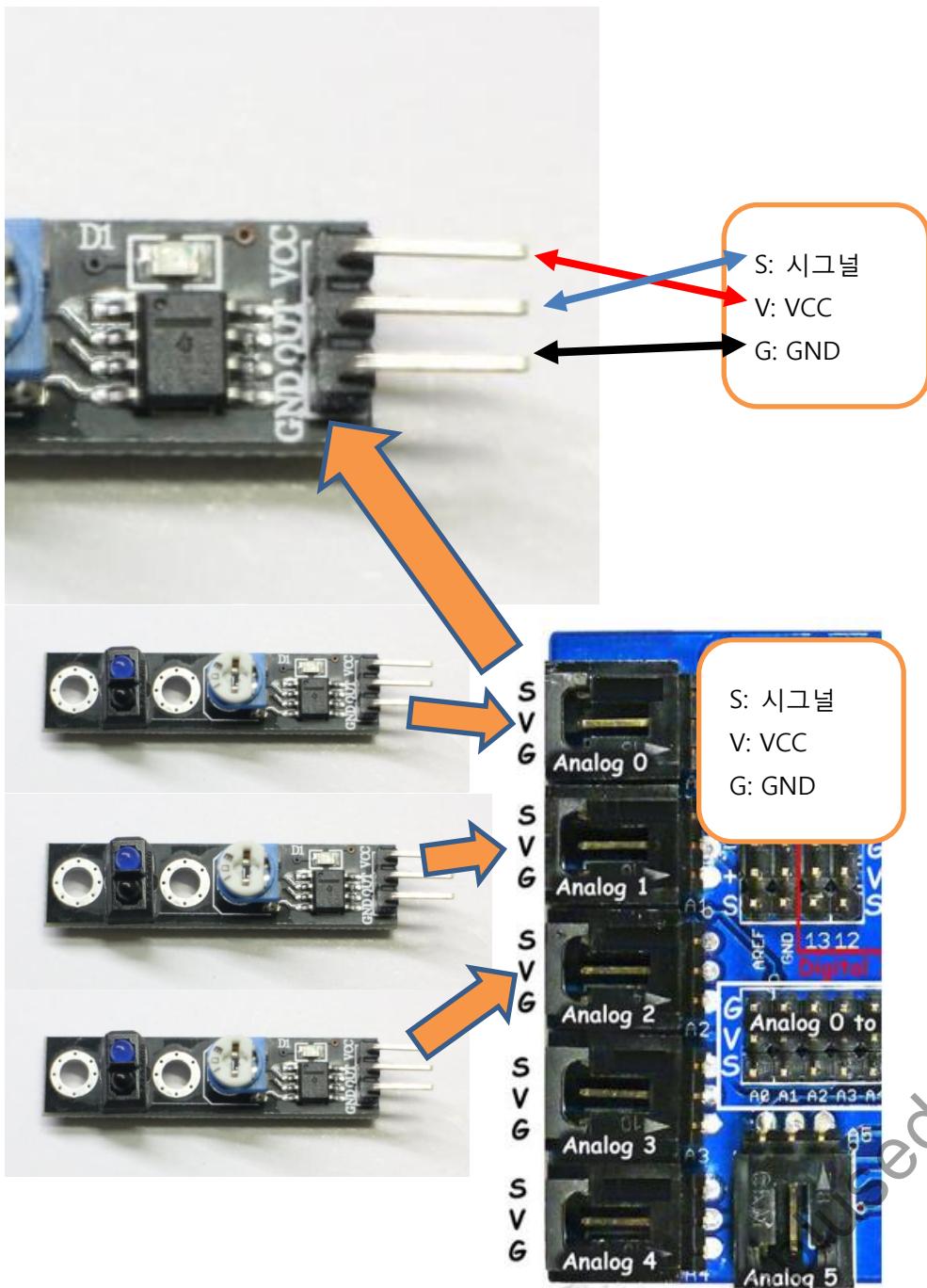
본 키트는 뒤풀 케이블 2.54pitch 제공되므로 3 가닥씩 잘라서 사용합니다.  
Female To Female 형태의 케이블입니다.



연결 시 주의할 점은 GND, VCC, OUT 정확히 확인 후 연결하기 바랍니다.



TCRT500 모듈의 핀은 VCC, OUT, GND 순서로 되어 있습니다. 케이블 연결 시  
참조하기 바랍니다.



## 12.7 라인 추적 운행 코드

라인센서(TCRT5000) 모듈 테스트를 하는 코드입니다. 아두이노의 A0 포트에 연결된 TCRT5000 모듈의 값을 확인하는 예제 코드입니다.

아래 예제 코드를 사용하여 0, 1 이 제대로 넘어 오는지 확인합니다.

코드: 4wd\_ex\_16-1

```
void setup() {  
    Serial.begin(9600);  
}  
  
void loop() {  
    int sensorValue = digitalRead(A0); //  
    Serial.println(sensorValue);  
    delay(100);  
}
```

다음은 라인센서 3 개 모듈을 장착하고 아래의 코드를 업로드 후 준비된 트랙(절연테이프, A4 용지)으로 테스트 합니다.

코드: 4wd\_ex\_16-2

```

void setup() {
    Serial.begin(9600);
}

void loop() {
    int sensorValue1 = digitalRead(A0);
    int sensorValue2 = digitalRead(A1);
    int sensorValue3 = digitalRead(A2);
    char szTemp[64];
    sprintf(szTemp,"%d %d %d",sensorValue1,sensorValue2,sensorValue3);
    Serial.println(szTemp);
    delay(100);           // delay in between reads for stability
}

```

라인 트레이싱 운행 코드입니다. 상당히 기초적인 로직 구현 코드입니다.

코드: 4wd\_ex\_16-3

```

/*
Line Tracer basic code.

*/

// 자동차 진행 방향 정의
// 

#define CAR_DIR_FW 0 // 전진.
#define CAR_DIR_BK 1 // 후진.
#define CAR_DIR_LF 2 // 좌회전.

```

```

#define CAR_DIR_RF 3 // 우회전
#define CAR_DIR_ST 4 // 정지.

//
// 차량 운행 방향 상태 전역 변수. // 정지 상태.
int g_carDirection = CAR_DIR_ST; //

int g_carSpeed = 128; // 최대 속도의 50~ 60 % for testing.

//
// 주의: ENA, ENB 는 PWM 지원 포트에 연결한다.
// 다음 업데이트시 변경합니다.

#define ENA 6
#define EN1 7
#define EN2 3

#define EN3 4
#define EN4 2
#define ENB 5

void init_car_controller_board()
{
    pinMode(ENA, OUTPUT); // ENA
    pinMode(EN1, OUTPUT); // EN1
    pinMode(EN2, OUTPUT); // EN2

    pinMode(ENB, OUTPUT); // ENB
    pinMode(EN3, OUTPUT); // EN3
    pinMode(EN4, OUTPUT); // EN4
}

//
// 전후좌우 4 개의 함수는 테스트시

```

```
// DC 모터 연결에 맞게 고쳐서 정정해야 합니다.  
// DC 모터 연결 (+)(-) 연결 변경하거나 코드를 변경합니다.  
  
//  
void car_forward()  
{  
    digitalWrite(EN1, LOW);  
    digitalWrite(EN2, HIGH);  
    analogWrite(ENA, g_carSpeed);  
  
    digitalWrite(EN3, LOW);  
    digitalWrite(EN4, HIGH);  
    analogWrite(ENB, g_carSpeed);  
}  
  
void car_backward()  
{  
  
    digitalWrite(EN1, HIGH);  
    digitalWrite(EN2, LOW);  
    analogWrite(ENA, g_carSpeed);  
  
    digitalWrite(EN3, HIGH);  
    digitalWrite(EN4, LOW);  
    analogWrite(ENB, g_carSpeed);  
}  
  
//  
void car_left()  
{  
    digitalWrite(EN1, HIGH);  
    digitalWrite(EN2, LOW);  
    analogWrite(ENA, g_carSpeed);
```

```

digitalWrite(EN3, LOW);
digitalWrite(EN4, HIGH);
analogWrite(ENB, g_carSpeed);
}

//



void car_right()
{
    digitalWrite(EN1, LOW);
    digitalWrite(EN2, HIGH);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, HIGH);
    digitalWrite(EN4, LOW);
    analogWrite(ENB, g_carSpeed);
}

//


//


void car_stop()
{
    analogWrite(ENA, 0);
    analogWrite(ENB, 0);
}

//


// 방향 전환값에 의해 차량 운행.
//


void car_update()
{
    // %%
    // 매번 loop 에서 호출되지만 커맨드가 있을 경우만 호출될 수 있도록
}

```

```
// 변경해야 합니다.
if (g_carDirection == CAR_DIR_FW) // 전진
{
    car_forward();
}
else
if (g_carDirection == CAR_DIR_BK) // 후진.
{
    car_backward();
}
else
if (g_carDirection == CAR_DIR_LF) // 좌회전
{
    car_left();
}
else
if (g_carDirection == CAR_DIR_RF) // 우회전
{
    car_right();
}
else
if (g_carDirection == CAR_DIR_ST) // 정지.
{
    car_stop();
}

void print_car_info()
{
    Serial.println("direction value " + g_carDirection);
    Serial.println("speed pwm value " + g_carSpeed);
}
```

```
////  
// 라인트레이서 모듈 핀맵  
#define LT_MODULE_L A0 // 14  
#define LT_MODULE_F A1 // 15  
#define LT_MODULE_R A2 // 16  
  
void init_line_tracer_modules()  
{  
    pinMode(LT_MODULE_L, INPUT);  
    pinMode(LT_MODULE_F, INPUT);  
    pinMode(LT_MODULE_R, INPUT);  
}  
  
// is detected left  
bool lt_isLeft()  
{  
    int ret = digitalRead(LT_MODULE_L);  
    return ret == 1 ? true : false;  
  
}  
  
bool lt_isForward()  
{  
    int ret = digitalRead(LT_MODULE_F);  
    return ret == 1 ? true : false;  
  
}  
  
bool lt_isRight()  
{  
    int ret = digitalRead(LT_MODULE_R);  
    return ret == 1 ? true : false;
```

```
}

//  
void lt_mode_update()  
{  
    int ll = lt_isLeft();  
    int ff = lt_isForward();  
    int rr = lt_isRight();  
  
    if (ll&&ff&&rr)  
    {  
        g_carDirection = CAR_DIR_ST; // stop  
    }  
    else  
    if (!ll&&!ff&&!rr)  
    {  
        g_carDirection = CAR_DIR_ST; // stop  
    }  
    else  
    if (ll)  
    {  
        g_carDirection = CAR_DIR_LF;  
    }  
    else  
    if (rr)  
    {  
        g_carDirection = CAR_DIR_RF;  
    }  
    else  
    if (ff)  
    {  
        g_carDirection = CAR_DIR_FW;  
    }  
}
```

```
}

// 부팅 후 1회 실행되는 함수. 초기화 함수. Setup()
void setup()
{
    //
    Serial.begin(9600);
    //
    init_car_controller_board();

    print_car_info();
}

// 계속 실행되는 함수. Loop()
void loop()
{
    car_update();
    lt_mode_update();
}
```

## 13 스마트폰과 4WD 스마트 자동차 연동

스마트폰과 4WD 제어 보드, 아두이노 우노와 블루투스 통신을 해보도록 합니다. 본 키트에 사용되는 아두이노 보드와 블루투스 모듈을 사용하여 운행 할 수 있습니다.

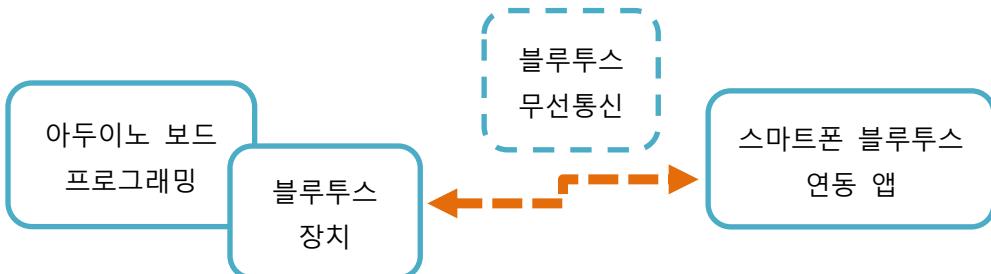
### 13.1 블루투스란

블루투스란 근거리 무선 통신 기술 규약을 의미합니다. 현재까지는 휴대용 기기, 근거리에서 사용되는 독립된 장치 등에 많이 사용됩니다. 블루투스의 출발은 휴대폰 제조사의 에릭슨, 노키아에서 개발된 만큼, 휴대용 기기에 많이 사용되고 있습니다. 블루투스의 출발은 소형 기기와의 통신을 위한 용도이므로 저전력을 사용하는 무선 통신 규약입니다. 스마트폰(휴대폰), 노트북, 태블릿 PC, 이어폰, 헤드폰, GPS 단말기, 키보드, 카메라, MP3 등에 사용됩니다.

블루투스의 사용 및 응용 범위는 계속 확대 추세이며, 현재의 블루투스는 4.0(~4.1)버전의 규약까지 나와 있습니다. 블루투스 무선 통신 범위는 증가 (50~100m)되면서 전력 소모는 줄어들어 더욱 더 효율적인 매체로 진보하는 중입니다. 차후, 다양한 형태의 블루투스 규약과 기기들이 등장할 것으로 예상됩니다.



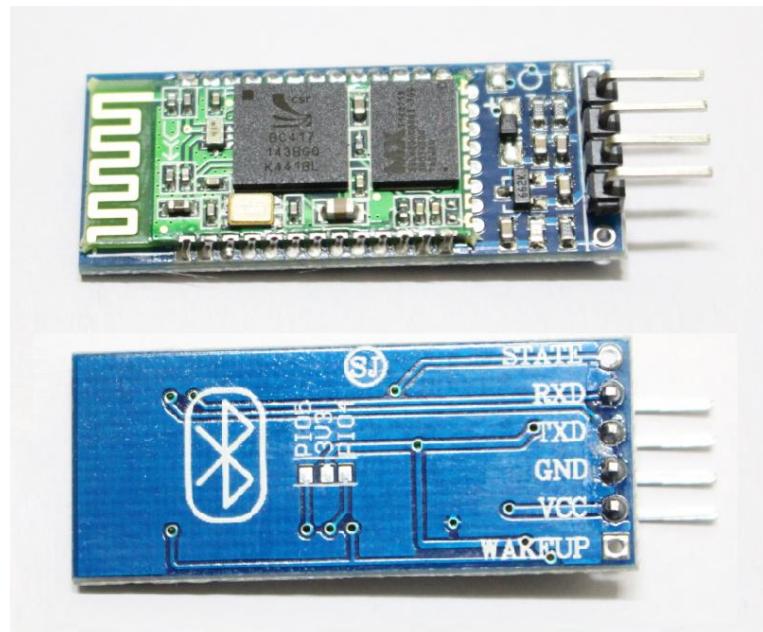
## 13.2 아두이노와 스마트폰의 블루투스 통신



블루투스 통신은 위에서 언급된 13.1 설명대로 말 그대로 근거리 통신망에 사용되는 무선통신 규약입니다. 게다가 사용하기 편하게 시리얼 통신이 가능한 칩셋 구조로 생산 되므로, 시리얼 통신 가능한 MCU, PC 등의 대부분의 장치에서 사용할 수 있습니다. 시리얼 통신은 최소 송/수신 2 개의 포트만 있으면 사용 가능한 구조입니다. 아두이노 보드도 안정적인 시리얼 통신이 가능한 기기이므로 블루투스 모듈을 사용하면, 안정적인 근거리 무선 시리얼 통신이 가능합니다.

## 13.3 블루투스 HC-06 슬레이브 모듈'

스마트 로봇 자동차의 근거리 통신 제어를 담당할 부품으로 블루투스 HC-06 슬레이브 모듈을 사용 합니다. 아두이노 보드에 연결하여 사용 가능한 블루투스 모듈은 HC-06 슬레이브 모듈과 HC-05 마스터 모듈도 사용 가능합니다. HC-05 블루투스 모듈은 마스터, 슬레이브 기능이 모두 지원됩니다.



블루투스 HC-06 슬레이브 모듈

아두이노 보드를 비롯하여 MCU 보드에서 주로 사용되는 블루투스 모듈입니다. 아두이노 보드가 대중화 되면서 HC-06 슬레이브 명칭으로 많이 사용 되고 있는 모듈중의 하나입니다.

아두이노 또는 MCU 보드를 사용하는 프로젝트에서 근거리 무선 통신에 많이 사용되는 블루투스 모듈입니다. HC-06 블루투스 모듈의 통신 거리는 10m 내외입니다. 좀더 먼 거리를 사용하기 위해서는 블루투스 4.0 지원 모듈, RF 무선 통신 UART 모듈 등을 사용하기도 합니다.

#### 블루투스 모듈 기술 사양:

- 2.4GHz 안테나 내장
- EDR 블루투스 2.0, 2Mbps - 3Mbps
- 외부 8Mbit FLASH
- 3.3V ~5V 사용 가능. (최대 7V)
- 표준 HCI 포트 (UART)
- 옵션 PIO 제어
- SMD 배치 프로세스로 모듈
- 디지털 2.4GHz 무선 송신

- CSR BC04 블루투스 칩 기술
- 블루투스 클래스 2 전력 레벨
- RoHS 규제 절차
- 보관 온도: -40 ~ +85 도, 작동 온도: -25 도 ~ 75 도
- 외형 (27mm × 13mm × 2mm)

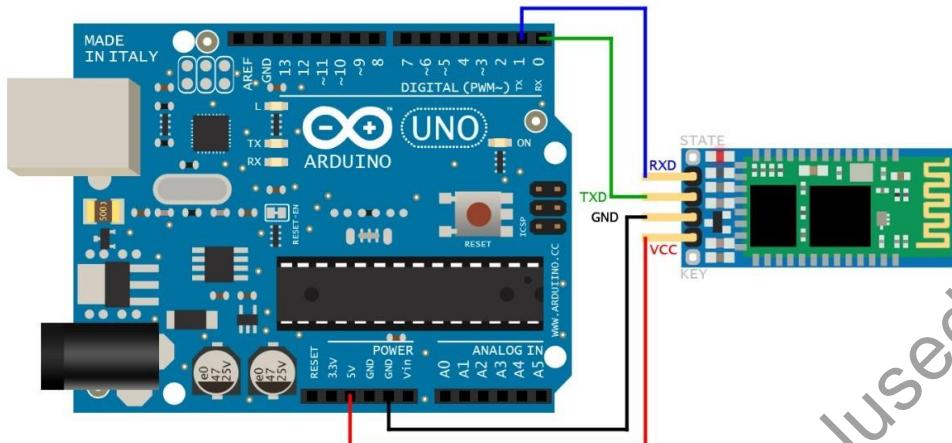
본 키트에 사용되는 블루투스 모듈은 GND, VCC, RX, TX 핀과 아두이노 핀에 연결하여 사용하는 모듈입니다. 별도의 RX/TX 5V ↔ 3V3 쉬프트 다운 레벨 컨버터 저항 연결 구성 필요 없습니다.

### 13.4 아두이노와 블루투스 모듈 연결

HC-06 모듈(또는 HC-05 모듈)은 블루투스 무선 통신을 시리얼 데이터로 주고 받을 수 있습니다.

블루투스 마스터/슬레이브 1:1 패어링이 된 상태인 경우 기기와의 통신 데이터는 모두 내부 시리얼 포트로 통신 가능하도록 구성 되어 있습니다.

즉, HC-06 (또는 HC-05) 모듈을 사용하기 위해서는 상대방 기기에서도 시리얼 포트 통신이 가능해야 합니다.



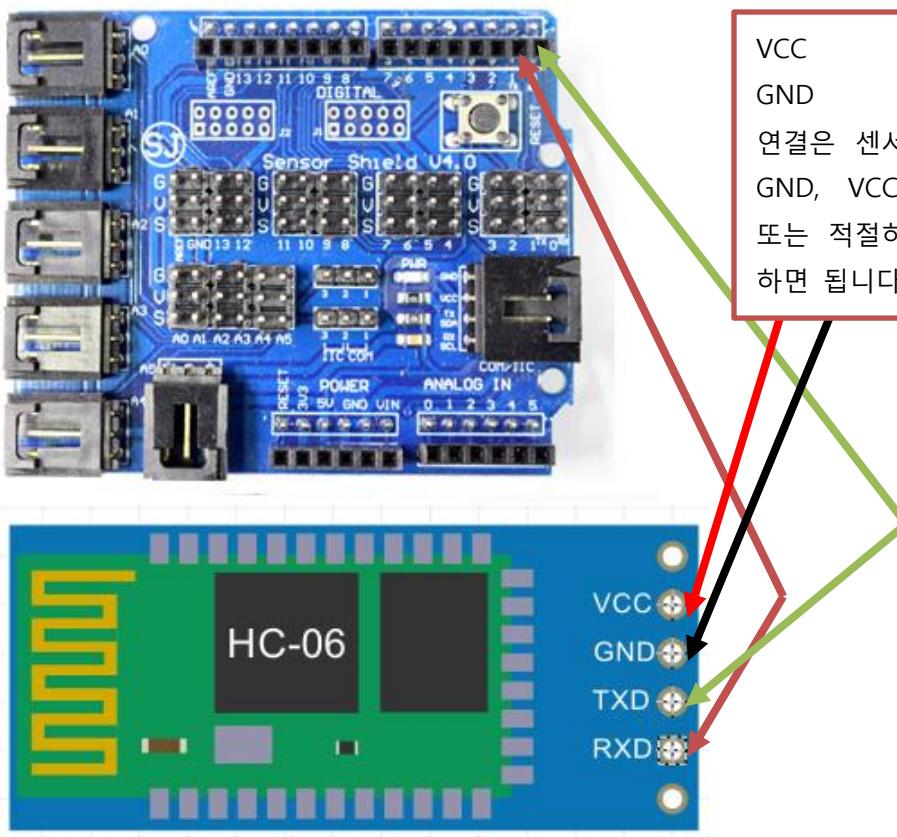
아두이노 우노 R3 & 블루투스 모듈 연결

### 13.4.1 아두이노 하드웨어 시리얼 포트

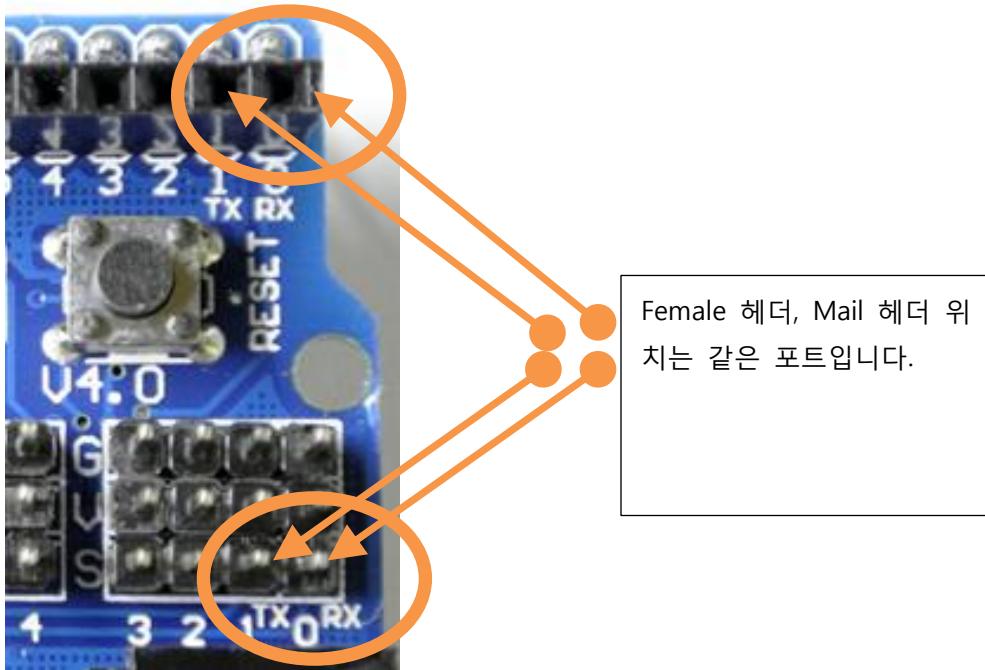
아두이노에서의 시리얼 통신 자원은 포트 1 개가 있습니다. 참고로 아두이노 메가 2560 보드는 하드웨어 시리얼 포트가 4 개 있습니다. 아두이노 우노 R3 보드의 MCU 는 ATmega328p 를 사용합니다. 아트메가 328P MCU 에서는 하드웨어 시리얼 통신 포트가 1 개만 지원됩니다.

아두이노 우노 R3 보드에서는 D0, D1 포트 2 개를 하드웨어 시리얼 포트로 펌웨어 업로드 포트로 고정하여 사용하고 있습니다. USB 케이블 연결 후 아두이노 IDE 로 펌웨어 업로드 및 테스트 시 사용되고 있습니다.

블루투스 HC-06 모듈에 RX (수신 포트), TX(송신 포트)가 있습니다. D0, D1 연결하여 사용 할 수 있습니다. 블루투스 모듈의 RX/TX, D0, D1 연결 중 일 때는 아두이노 IDE 스케치에서의 프로그램 업로드가 안됩니다. 펌웨어 업로드 후 단독으로 아두이노 사용할 경우에는 블루투스 모듈을 D0, D1 연결하여 사용할 수 있지만 다소 사용하기 불편합니다. 아두이노 IDE 스케치 프로그램과 연동하여 USB 시리얼 포트를 사용한 펌웨어 업로드 포트로 사용되고 있습니다.



HC-06 블루투스 모듈	아두이노
VCC	5V
GND	GND
TXD	D0 (RX)
RXD	D1 (TX)



### 13.4.2 아두이노 소프트웨어 시리얼 통신

아두이노 IDE 스케치 기본 라이브러리 중에는 “소프트웨어 시리얼 라이브러리”가 있습니다. 소프트웨어적인 방식으로 시리얼 통신을 지원합니다. 아두이노 보드의 임의의 포트를 지정하여 수신/송신 포트로 사용할 수 있습니다. 수신(RX) D2, 송신(TX) D3 포트를 사용하는 경우에는 블루투스 모듈의 TX, RX 포트를 연결하면 됩니다.

### 13.4.3 하드웨어, 소프트웨어 시리얼 통신의 차이점

아두이노를 비롯한 MCU 보드에서의 시리얼 통신 자원을 하드웨어 포트, 소프트웨어 포트 방식으로 사용하면 일반적인 시리얼 통신은 MCU 클럭 및 환경에 따라 최대 속도가 다르다는 차이점이 있습니다. 그 외에 시리얼 FIFO 버퍼 용량 등이 있습니다.

#### >> 차이점

안정적인 시리얼 통신을 하기 위해서는 하드웨어 시리얼 포트를 사용 해야 합니다.

소프트웨어 시리얼 포트를 사용하는 경우에는 통신 처리를 하기 위해 다른 나머지 포트들이 제대로 신호 처리를 할 수 없는 경우가 대부분입니다. 반대로 다른 포트들이 신호 변경 되는 동안에는 소프트웨어 시리얼 통신이 불가 합니다.

최근의 아두이노 사이트에서의 소프트웨어시리얼 2 라이브러리를 사용하는 경우에는 하드웨어 시리얼 포트처럼 다른 포트에 영향을 받지 않고 사용 가능하다고 합니다. 필요한 경우에는 테스트해보시기 바랍니다.

### 13.5 블루투스 시리얼 데이터 연동 기초 코드

블루투스로 데이터를 주고 받기 위해서는 블루투스 기기(모듈) 페어링(Pairing)되어 있어야 합니다. 페어링이란 용어 그대로 둘씩 짹짓기입니다. 블루투스 기기의 통신 기능은 모두 페어링 상태에서 작동이 가능하게 되어 있습니다.

스마트폰은 거의 모두 블루투스 마스터/슬레이브 기능을 지원하고 있습니다.

#### >> 블루투스 마스터 기능

마스터 기능은 쉽게 말하면 주변에 있는 블루투스 지원 기기들을 검색하여 페어링을 할 수 있습니다. 물론 반대로 슬레이브 기능도 가지고 있습니다. 즉, 다른 기기에서의 페어링 요청을 받아들여 사용될 수도 있습니다.

블루투스 마스터 모듈도 기본 기능 상태는 대부분 슬레이브 상태입니다. 마스터 기능으로 사용하기 위해서는 블루투스 모듈의 명령어 설정 모드로 진입하여 일련의 명령어를 입력하여 사용하게 됩니다.

#### >> 블루투스 슬레이브 기능

슬레이브 기능은 주변 블루투스의 요청에 의해 페어링이 될 수 있습니다. 스마트폰의 블루투스를 마스터로 사용하여 아두이노와 연결된 블루투스 모듈과 페어링을 합니다. 리모트 컨트롤 작동 또는 초음파 센싱 운행과 리모트 컨트롤 코드를 구현한 상태인 경우 거의 비슷하게 수정하여 변경 할 수 있습니다.

블루투스 통신에 의한 운행 제어도 쉽게 적용 시킬 수 있습니다. 시리얼 통신 기반으로 명령어 전송 및 처리 부분 구현해 주면 됩니다. 먼저 블루투스 시리얼 통신을 하는 방법을 알아보도록 하겠습니다.

### 13.6 아두이노 블루투스 시리얼 통신 예제코드

HC-06 블루투스 모듈	아두이노
VCC	5V
GND	GND
TXD	D2
RXD	D3

예제코드 ex17: 소프트웨어 시리얼 통신 테스트.

```
#include <SoftwareSerial .h>
/*
D2 포트는 블루투스 모듈의 TX
D3 포트는 블루투스 모듈의 RX
*/
SoftwareSerial bt(2, 3); // 소프트웨어 시리얼 포트 지정.

void setup()
{
```

```

Serial.begin(9600); // 시리얼 통신 초기화
bt.begin(9600); //
Serial.println("ready");
}

void loop()
{
    if(bt.available()) //블루투스를 통해 데이터가 날아오면
    {
        Serial.write("phone : ");
        Serial.write(bt.read()); //시리얼 모니터로 받은 내용 출력
        Serial.println();
    }

    if(Serial.available()) //시리얼 모니터에서 데이터를 보내면
    {
        bt.write("PC : ");
        bt.write(Serial.read()); //스마트폰으로 받은 데이터 출력
        bt.println();
    }
}

```

## 13.7 스마트폰과의 페어링

스마트폰과의 페어링을 위해 블루투스 모듈이 작동 중이어야 합니다. 스마트폰에서의 블루투스 설정 메뉴로 들어가 블루투스 검색을 눌러 페어링을 합니다. HC-06 블루투스 모듈에 전원을 입력하도록 합니다.

HC-06 모듈의 장치 명칭은 별도로 변경하지 않은 상태인 경우 "HC-06"이라는 이름으로 보입니다. 검색된 블루투스 모듈 목록에서 "HC-06" 선택합니다. 블루투스 HC-06 페어링을 합니다. 기본 암호는 1234 입니다. 또는 0000 입니다.

## 13.8 안드로이드 앱 설치

안드로이드 앱 다운로드 주소.

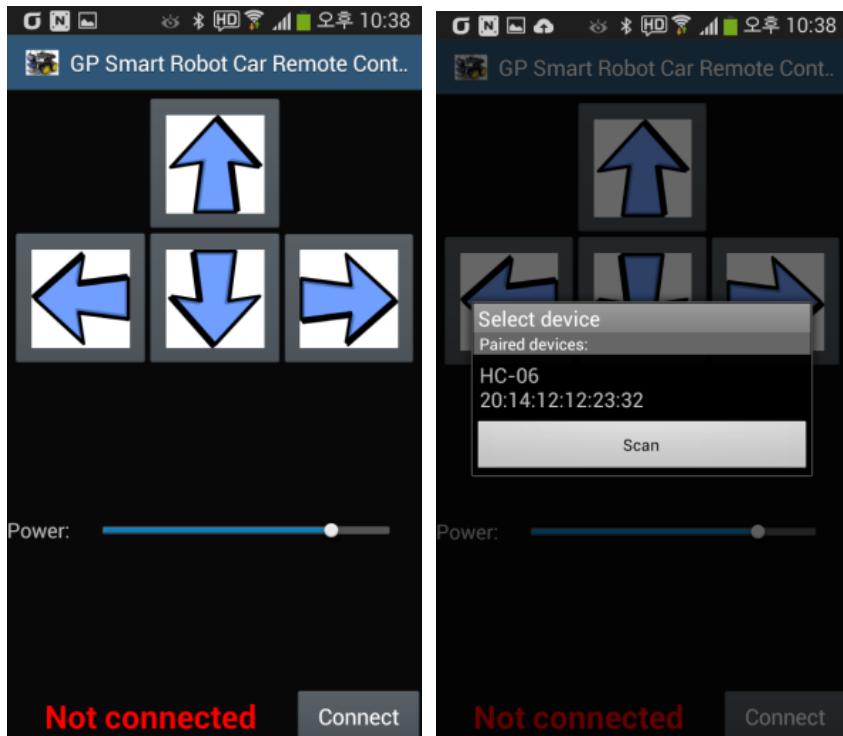
[http://www.gameplusedu.com/pds/gpshop/arduino/robotics/kit\\_example/car\\_4wd/android\\_apk/GPSmartCarRemoteManager.apk](http://www.gameplusedu.com/pds/gpshop/arduino/robotics/kit_example/car_4wd/android_apk/GPSmartCarRemoteManager.apk)

[GPSmartCarRemoteManager.apk](http://www.gameplusedu.com/pds/gpshop/arduino/robotics/kit_example/car_4wd/android_apk/GPSmartCarRemoteManager.apk)

앱 설치 후 아래와 같은 실행 아이콘이 보입니다.



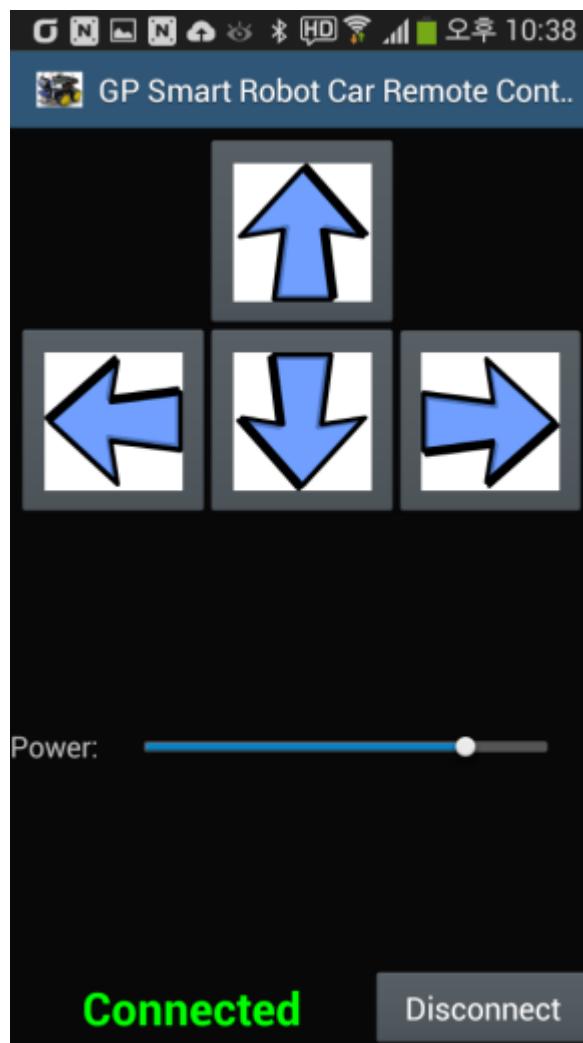
태블릿, 스마트폰 참조 그림..



실행화면,

연결 버튼("Connect") 을 누르면 블루투스 페어링 설정된 목록을 보게 됩니다.

정상 블루투스 연결 후 아래와 같은 연결 화면을 보게 됩니다.



버튼을 누르게 되면 스마트 로봇 자동차는 원하는 방향으로 동작을 합니다.

스마트폰의 방향 버튼 눌림 상태(터치 상태)에서 해당 방향으로 동작합니다.  
방향 버튼 터치 상태가 아닌 경우에는, 스마트 로봇 자동차는 정지 상태로 바로  
변경됩니다.

### 13.8.1 블루투스 제어 주행 코드

스마트폰 앱과 블루투스 통신을 통하여 로봇 자동차를 조종하는 주행 코드입니다. 다음 스케치를 컴파일하고 아두이노 우노 보드로 업로드 합니다.

```
/*
 * Smart Robot Car V3
 * - Bluetooth(HC-06) control version
 * - Android app provided
 */

#include <SoftwareSerial.h>

///////////
// <BT>      <UNO>
// TX <----> RX
// RX <----> TX
///////////

SoftwareSerial btSerial(10, 11); // RX, TX(UNO)

///////////
// Note: ENA and ENB must be connected to PWD supported pins
//
#define ENA 6 // PWD
#define EN1 7
#define EN2 3

#define EN3 4
#define EN4 2
#define ENB 5 // PWD

///////////
// Ultrasonic sensor
```

```
//////////  
int TRIG_pin = 12; // 센서 Trig 핀, D12  
int ECHO_pin = 13; // 센서 Echo 핀, D13  
  
#define blinkLED 8 // for crash warning  
  
//////////  
// Car direction  
//  
#define CAR_DIR_FW 0 // forward  
#define CAR_DIR_BK 1 // backward  
#define CAR_DIR_LT 2 // left turn  
#define CAR_DIR_RT 3 // right turn  
#define CAR_DIR_ST 4 // stop  
  
//////////  
// Default direction and speed  
//  
int g_carDirection = CAR_DIR_ST;  
int g_carSpeed = 230; // 60% of max speed for testing  
  
//////////  
// Note : confirm HIGH/LOW for correct movement  
//  
void car_forward()  
{  
    digitalWrite(EN1, HIGH);  
    digitalWrite(EN2, LOW);  
    analogWrite(ENA, g_carSpeed);  
  
    digitalWrite(EN3, HIGH);  
    digitalWrite(EN4, LOW);  
    analogWrite(ENB, g_carSpeed);
```

```
}

void car_backward()
{
    digitalWrite(EN1, LOW);
    digitalWrite(EN2, HIGH);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, LOW);
    digitalWrite(EN4, HIGH);
    analogWrite(ENB, g_carSpeed);
}

void car_left()
{
    digitalWrite(EN1, LOW);
    digitalWrite(EN2, HIGH);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, HIGH);
    digitalWrite(EN4, LOW);
    analogWrite(ENB, g_carSpeed);
}

void car_right()
{
    digitalWrite(EN1, HIGH);
    digitalWrite(EN2, LOW);
    analogWrite(ENA, g_carSpeed);

    digitalWrite(EN3, LOW);
    digitalWrite(EN4, HIGH);
    analogWrite(ENB, g_carSpeed);
```

```
}

void car_stop()
{
    analogWrite(ENA, 0);
    analogWrite(ENB, 0);
}

///////////////////////
// Execute car moving
//
void update_Car()
{
    switch ( g_carDirection ) {
        case CAR_DIR_FW:
            car_forward();
            break;
        case CAR_DIR_BK:
            car_backward();
            break;
        case CAR_DIR_LT:
            car_left();
            break;
        case CAR_DIR_RT:
            car_right();
            break;
        case CAR_DIR_ST:
            car_stop();
            break;
        default :
            ;
    }
    return;
}
```

```
}

///////////
// Class - Serial Protocol
//
class _CommProtocol
{
private:
    unsigned char protocolPool[28];
    int bufPoint;

public:
    _CommProtocol()
    {
    }

void addPool(unsigned char cByte)
{
    if (bufPoint < 28)
    {
        if (bufPoint == 0 and cByte != 0x0c)
            return; // invalid code

        protocolPool[bufPoint++]=cByte;
        //Serial.print("bufPoint -> ");
        //Serial.println(bufPoint);
    }
}

void clearPool()
{
    bufPoint = 0;
```

```
        memset(protocolPool, 0x00, 28);
        Serial.println("clearPool");
    }

bool isValidPool()
{
    if (bufPoint >= 28)
    {
        //Serial.print("protocol length : ");

        if (protocolPool[0] == 0x0c && protocolPool[14] == 0x0c)
        {
            //Serial.println(protocolPool.length());
            return true;
        }
        else
        {
            clearPool();
            Serial.println("isValidPool 28 OVER");
        }
    }
    return false;
}

unsigned char getMotorLValue()
{
    unsigned char szProto[14];
    memcpy(szProto, protocolPool, 14);
    if (szProto[0] == 0x0C &&
        szProto[1] == 0x00 &&
        szProto[2] == 0x80 &&
        szProto[3] == 0x04 &&
        szProto[4] == 0x02)
```

```

{
    unsigned char l = szProto[5];// -0x32;
    return l;
}
return 0x00;
}

unsigned char getMotorRValue()
{
    unsigned char szProto[14];
    memcpy(szProto, &protocolPool[14], 14);
    if (szProto[0] == 0x0C &&
        szProto[1] == 0x00 &&
        szProto[2] == 0x80 &&
        szProto[3] == 0x04 &&
        szProto[4] == 0x01)
    {
        unsigned char l = szProto[5];// -0x32;
        return l;
    }
    return 0x00;
}
}; // class(_CommProtocol)

///////////////////////////////
// Create an instance of class(_CommProtocol)
//
_CommProtocol SerialCommData;

///////////////////////////////
// Parse the serial input value and convert to MOVE command
//
void process_SerialCommModule()

```

```

{
if (SerialCommData.isValidPool())
{
    char motorLR[2];

    motorLR[0] = (char)SerialCommData.getMotorLValue();
    motorLR[1] = (char)SerialCommData.getMotorRValue();
    SerialCommData.clearPool();

    //
    Serial.print("Left [");
    Serial.print(motorLR[0],DEC);
    Serial.print("] Right [");
    Serial.print(motorLR[1],DEC);
    Serial.println("]");

    //

    char szCmdValue = '5';
    // set MOVE commands
    if (motorLR[0] == 0 && motorLR[1] == 0) { // (0,0) stop
        szCmdValue = '5';
    }
    else
    {
        int nSpeed;
        nSpeed = max(abs(motorLR[0]), abs(motorLR[1]));

        // Set direction
        if (motorLR[0] > 0 && motorLR[1] > 0) // (+,+) forward
        {
            szCmdValue = '2';
            g_carSpeed = 255.0f * ((float)nSpeed / 100.0f);
        }
    }
}

```

```

else if (motorLR[0] < 0 && motorLR[1] < 0) // (-,-) backward
{
    szCmdValue = '8';
    g_carSpeed = 255.0f * ((float)nSpeed / 100.0f);
}

else if (motorLR[0] < 0 && motorLR[1] > 0) // (-,+) left turn
{
    szCmdValue = '4';
    g_carSpeed = 255.0f * ((float)((float)nSpeed*1.66f) / 100.0f);
}

else if (motorLR[0] > 0 && motorLR[1] < 0) // (+,-) right turn
{
    szCmdValue = '6';
    g_carSpeed = 255.0f * ((float)((float)nSpeed*1.66f) / 100.0f);
}

}

// Serial.print("speed ");
Serial.print(g_carSpeed);
Serial.print(" ");
Serial.println(szCmdValue);
//


// Set the direction and speed with command
controlByCommand(szCmdValue);

}
}

///////////////////////////////
// Hint : Command codes come from keypad numbers
//
void controlByCommand(char doCommand)
{

```

```
switch ( doCommand ) {  
    case '+' :    // speed up  
        g_carSpeed += 20;  
        g_carSpeed = min(g_carSpeed, 255);  
        break;  
    case '-' :    // speed down  
        g_carSpeed -= 20;  
        g_carSpeed = max(g_carSpeed, 75);  
        break;  
    case '2' :    // forward  
        g_carDirection = CAR_DIR_FW;  
        break;  
    case '5' :    // stop  
        g_carDirection = CAR_DIR_ST;  
        break;  
    case '8' :    // backward  
        g_carDirection = CAR_DIR_BK;  
        break;  
    case '4' :    // left  
        g_carDirection = CAR_DIR_LT;  
        break;  
    case '6' :    // right  
        g_carDirection = CAR_DIR_RT;  
        break;  
    default   :  
        ;  
}  
return;  
}  
  
//////////  
// Setup to run once  
//
```

```
void setup()
{
    Serial.begin(9600);      // PC serial monitor debugging
    btSerial.begin(9600);    // bluetooth serial connection

    //init car control board
    pinMode(ENA, OUTPUT);  // ENA
    pinMode(EN1, OUTPUT);  // EN1
    pinMode(EN2, OUTPUT);  // EN2

    pinMode(ENB, OUTPUT);  // ENB
    pinMode(EN3, OUTPUT);  // EN3
    pinMode(EN4, OUTPUT);  // EN4

    pinMode(blinkLED, OUTPUT); // for crash check
    pinMode(TRIG_pin, OUTPUT);
    pinMode(ECHO_pin, INPUT);

    //
    Serial.print("direction value ");
    Serial.println(g_carDirection);
    Serial.print("speed pwm value ");
    Serial.print(g_carSpeed);
    Serial.println("");
    //
}

///////////////////////////////
// main code to run repeatedly
//
void loop()
{
```

```
if (btSerial.available()) {  
  
    unsigned char cByte;  
  
    cByte = btSerial.read();  
  
    SerialCommData.addPool(cByte); // store the serial input to Buffer  
  
    process_SerialCommModule(); // parse and change the input value  
                            // to MOVE command  
    update_Car();          // execute car MOVE  
  
}  
  
}
```

감사합니다

<http://www.gameplusedu.com>