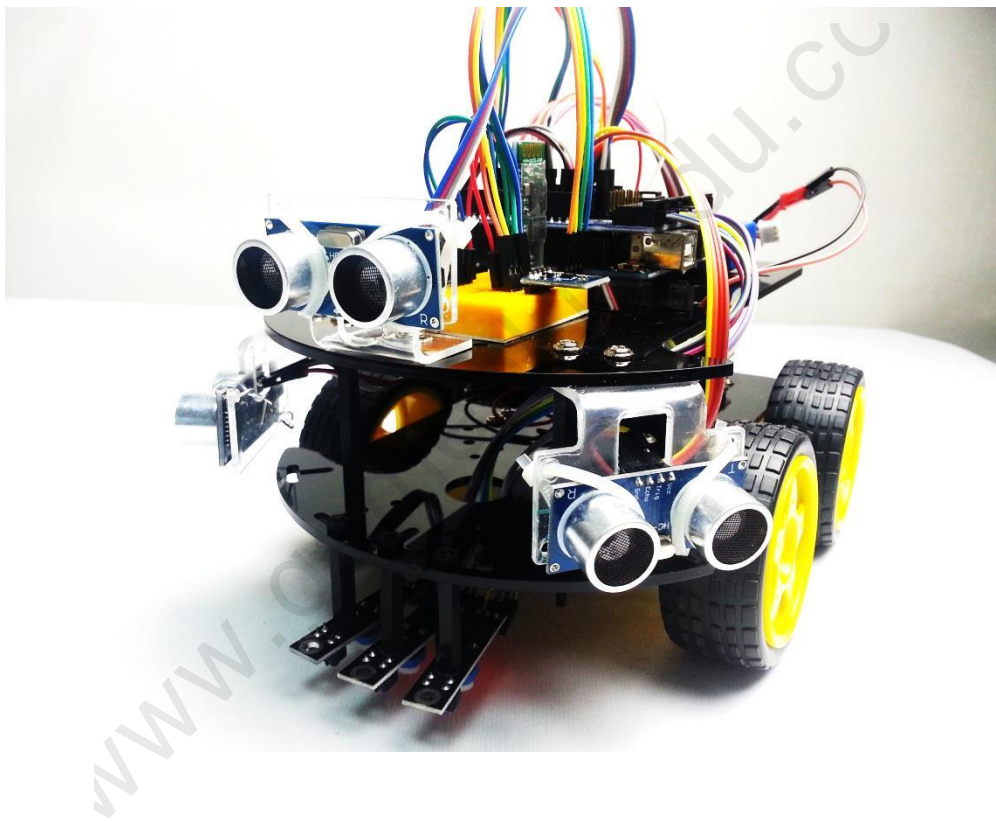


아두이노 스마트 로봇 자동차 키트

Make Obstacles Avoiding with Arduino SMART Robot Car



<http://www.gameplusedu.com>

igameplus.co.,ltd All Rights Reserved.

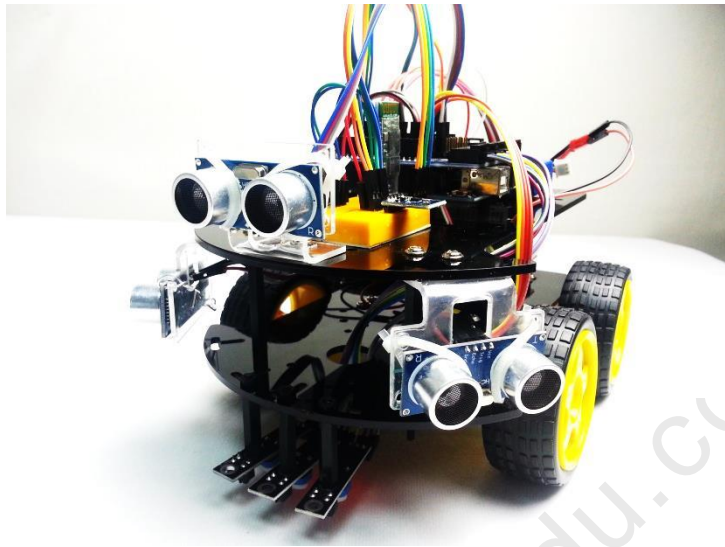
Arduino Robot Smart 4WD Car Kit V4

아두이노 로봇 스마트 자동차 4WD 키트 버전 4

문서 버전 11.3

<http://www.gameplusedu.com>

<http://www.igameplus.com>



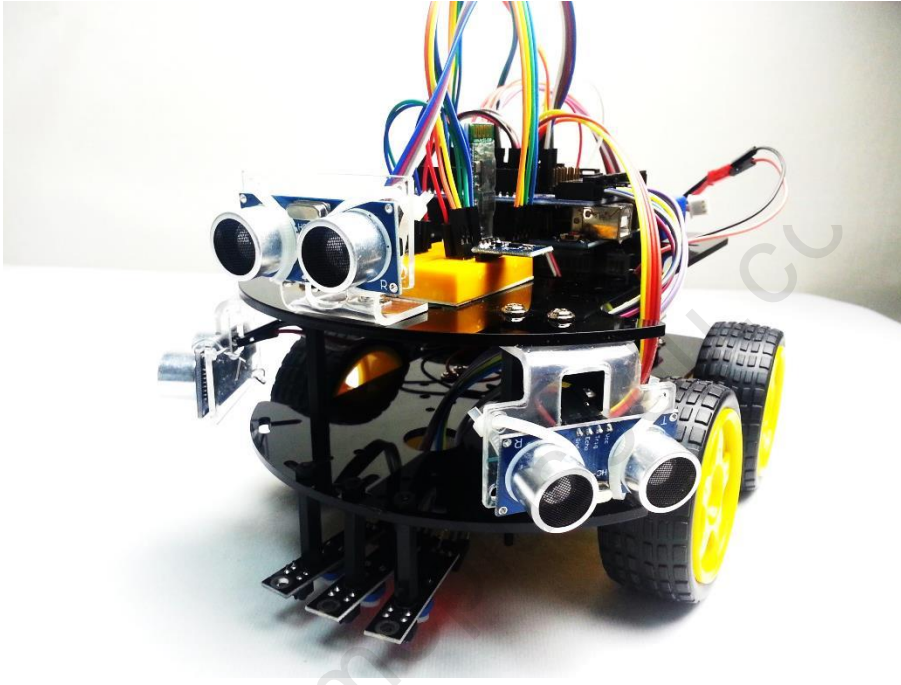
목차

1	스마트 로봇 자동차 키트 소개	5
2	스마트 로봇 자동차 기초 기술 요구 사항	6
3	Package included.....	7
4	스마트 로봇 자동차 조립	10
4.1	자동차 하부 조립	11
4.1.1	모터와 하판 조립	12
4.1.2	L293N 제어보드와 모터 케이블 연결	15
4.1.3	9V 배터리 박스	19
4.2	자동차 상부 조립	20
4.2.1	아두이노 우노, 센서 실드 V4, 배터리 박스	21
4.2.2	배터리-제어보드-센서실드 전원 연결	24
4.2.3	제어보드-센서실드 제어포트	27
4.3	자동차 주행 테스트	29
4.3.1	DC 모터 주행 기본 코드	29
5	블루투스 주행 제어	32
5.1	블루투스란	32
5.2	블루투스 모듈 연결	33
5.3	스마트폰 블루투스 페어링(Pairing).....	35
5.4	블루투스 제어 주행 코드	36
5.5	안드로이드 앱	48
6	DC 모터	50
6.1	DC 모터 설명	50
6.2	DC 모터 전원 연결 케이블 연결 시 주의 사항	52
6.3	DC 모터 리드선 연결 시 리드선 주의 사항	53
6.4	DC 모터 전원 연결선 정리	53
7	L298N 듀얼 모터 제어 드라이버	55
7.1	H-Bridge 회로	56
7.2	L298N 모터 제어 보드	58
7.3	5V_EN 포트 사용 설명	61
7.4	아두이노 우노 R3 & L298N 제어 포트 연결	62

7.5	자동차 속도 및 방향 조절	64
8	라인 추적 주행 (Line Tracking).....	67
8.1	트랙 라인 설치.....	67
8.2	라인 트레이서 모듈 기초 정보.....	69
8.2.1	라인센서(TCRT5000)모듈 인식 거리.....	69
8.3	라인 트레이서 모듈 장착.....	71
8.4	라인 추적 운행 기초 원리	73
8.5	아두이노 우노 R3 제어 보드와 연결.....	74
8.6	라인 추적 운행 코드.....	77
9	조옴파 센서 프로그래밍.....	86
9.1	조옴파 센서 작동 원리	87
9.2	조옴파 센서 연결하기.....	89
9.3	조옴파 센서를 이용한 거리 측정 구현.....	90
10	장애물 회피 자율주행.....	92
10.1	장애물 감지 조옴파 센서 장착.....	93
10.2	방향 탐지를 위한 전자나침반 모듈 장착.....	96
10.3	장애물 감지 방향 전환	99
10.4	장애물 회피 자율주행 구현.....	100
11	아두이노 우노 R3.....	112
11.1	개요.....	112
11.2	기술 사양	113
11.3	아두이노 소프트웨어.....	115
12	스마트폰과 자동차 연동	118
12.1	블루투스란	118
12.2	아두이노와 스마트폰의 블루투스 통신	118
12.3	블루투스 HC-06 슬레이브 모듈	119
12.4	아두이노와 블루투스 모듈 연결	121
12.4.1	아두이노 하드웨어 시리얼 포트	121
12.4.2	아두이노 소프트웨어 시리얼 통신	122
12.4.3	하드웨어, 소프트웨어 시리얼 통신의 차이점	122
12.5	블루투스 시리얼 데이터 연동 기초.....	123

1 스마트 로봇 자동차 키트 소개

아두이노 보드와 센서를 활용한 다기능 스마트 로봇 자동차 키트입니다.



아두이노 우노 R3 마이크로 컨트롤러 보드와 DC 모터를 사용하는 자동차 프레임, L298N 모터 제어 보드를 조립하고 스케치 프로그램으로 모터를 제어하여 자동차를 조종할 수 있습니다. 또한 센서를 부착하고 거리, 장애물, 라인 트랙 등을 감지하여 회피나 자율 주행 등의 특별한 기능을 프로그램 코딩으로 직접 체험해 볼 수 있습니다.

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

- 초음파 센서에 의한 거리 감지, 장애물 감지, 회피 주행
- 바닥 라인 트랙 유도선 주행
- 리모트 컨트롤러 제어 주행
- 블루투스 모듈 장착 스마트폰 앱으로 제어 운행

4WD 새시 프레임에는 사용되는 모든 부품 고정 홀더가 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 이행하고 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 UNO R3 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 Ultrasonic Sensor Module
- 3 x Line inductive module
- 1 x Infrared receiver module
- 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

* 제품에 따라 구성품목이 다를 수 있음.

스마트 자동차 교육 과정

안드로이드 앱을 이용한 자동차 제어 과정

4. 스마트 로봇 자동차 조립



5. 블루투스 주행 제어



테스트

라인센서를 이용한 자동 주행 과정

4. 스마트 로봇 자동차 조립



8. 라인 추적 주행



테스트

초음파 센서 이용한 장애물 회피 자율 주행 과정

4. 스마트 로봇 자동차 조립



9. 초음파센서 프로그래밍



10. 장애물회피 자율 주행

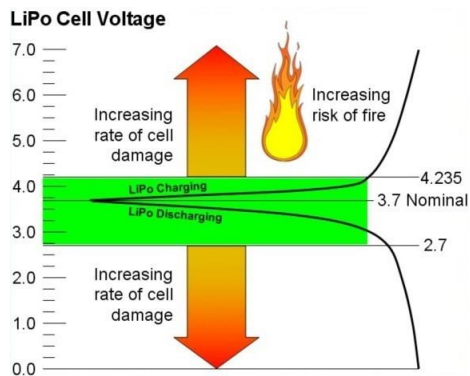


테스트

배터리 안전 사용법

리포(LiPo)배터리는 강력한 출력이 필요한 드론이나 RC 카에 많이 사용됩니다. 순간출력이 좋지만 폭발위험도 있어 주의가 필요합니다.

- 과충전이나 과방전되지 않도록 적절한 충전상태를 유지
- 배터리가 파손되지 않도록 주의(파손시 폭발/화재 발생)
- 사용한 배터리는 완전방전 후 폐기(소금물에 24 시간 보관)
- 배터리가 부풀어오르면 폐기처리



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 샤프트 아크릴에 많은 홀더가 있습니다. 본 키트의 차체 샤프트는 아두이노 보드, 센서 거치대, L298N 제어 보드, 배터리 상자 등의 고정 홀더들이 있습니다.

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



4.1 자동차 하부 조립

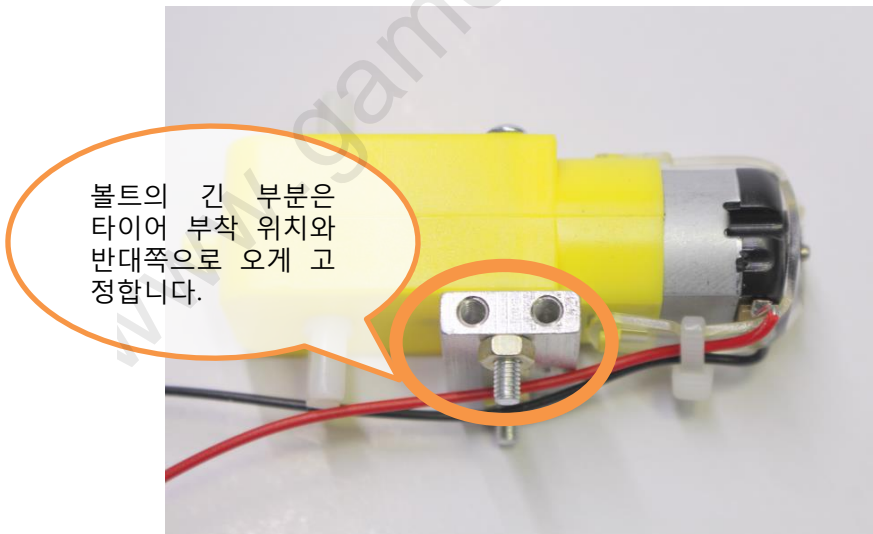
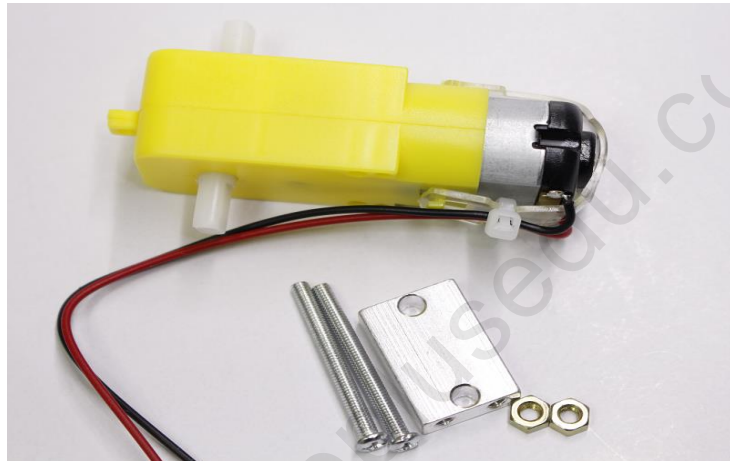
자동차 하부는 DC 모터, 바퀴, L298N 모터 제어보드, 차체 새시는 아크릴 유광 검은색 또는 업그레이드에 의해 검은색 / 빨간색 색상 등으로 되어 있습니다

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	
		

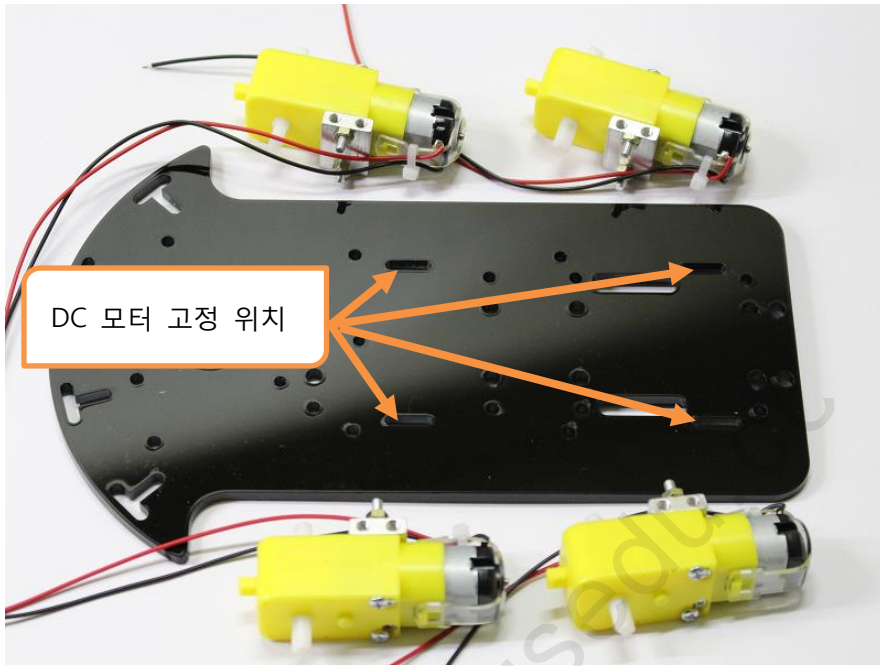
자동차 하부 부품

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

4.1.1 모터와 하판 조립

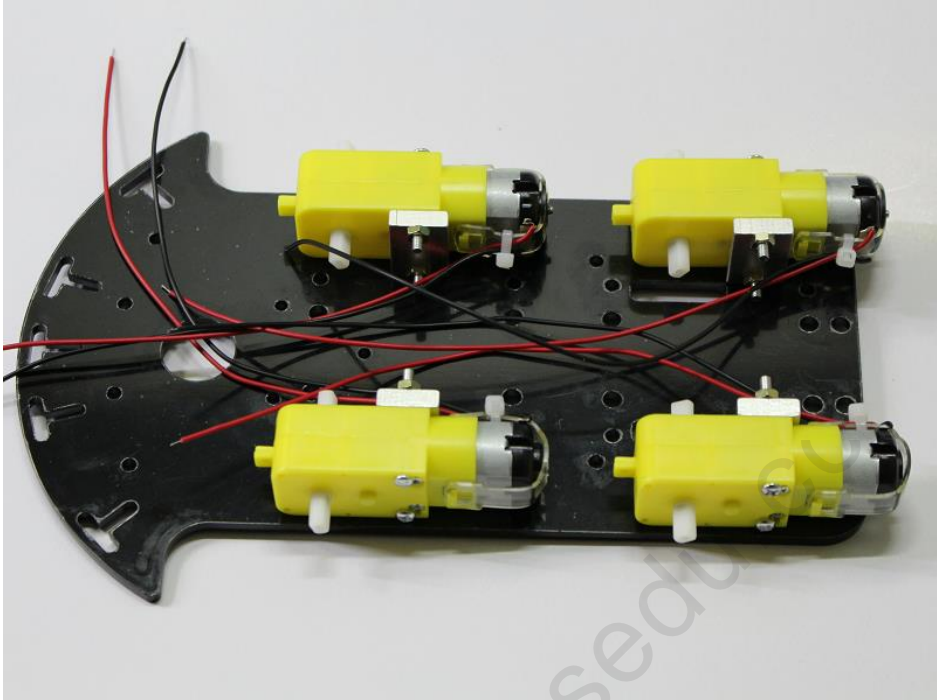


볼트와 너트, 기어모터 고정부품, DC 기어모터입니다. 고정판 틀과 볼트/너트를 연결하여 고정합니다.



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



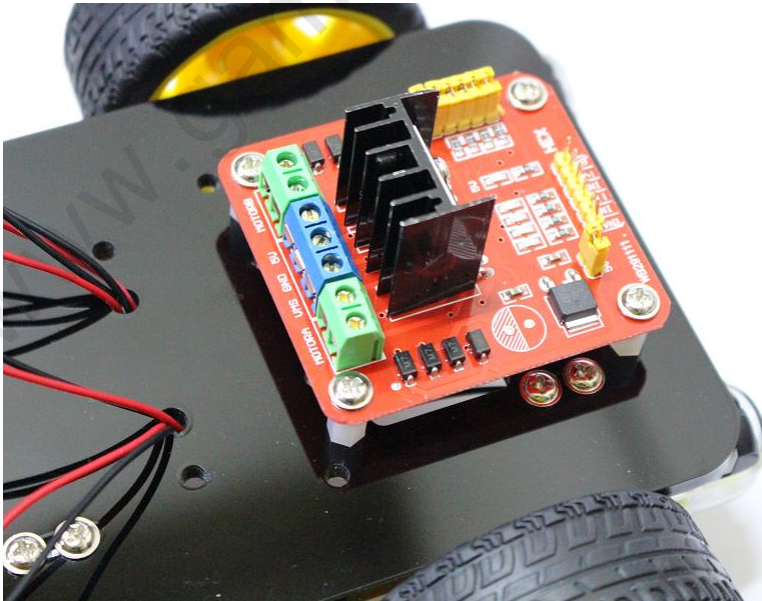
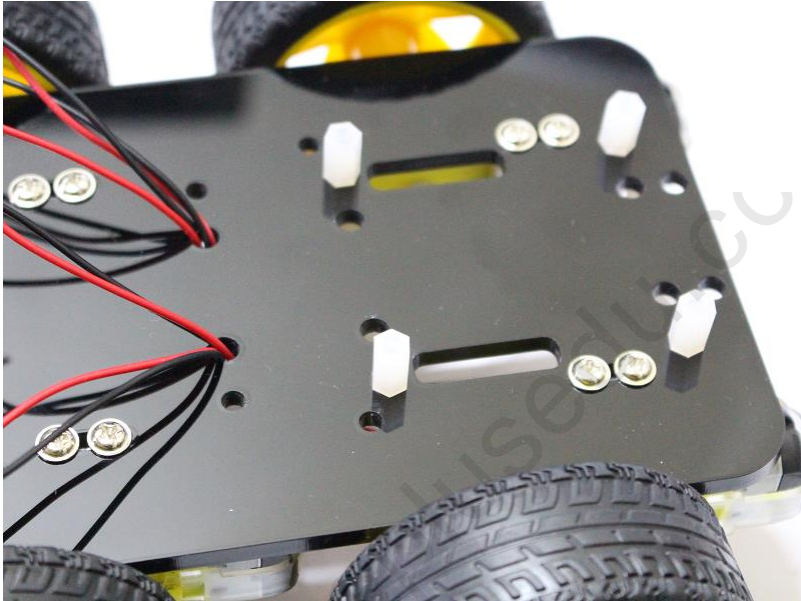


하판에 DC 모터 고정이 끝나면 타이어를 DC 모터 샤프트의 홈 모양에 맞춰 끼워서 바퀴를 조립합니다.

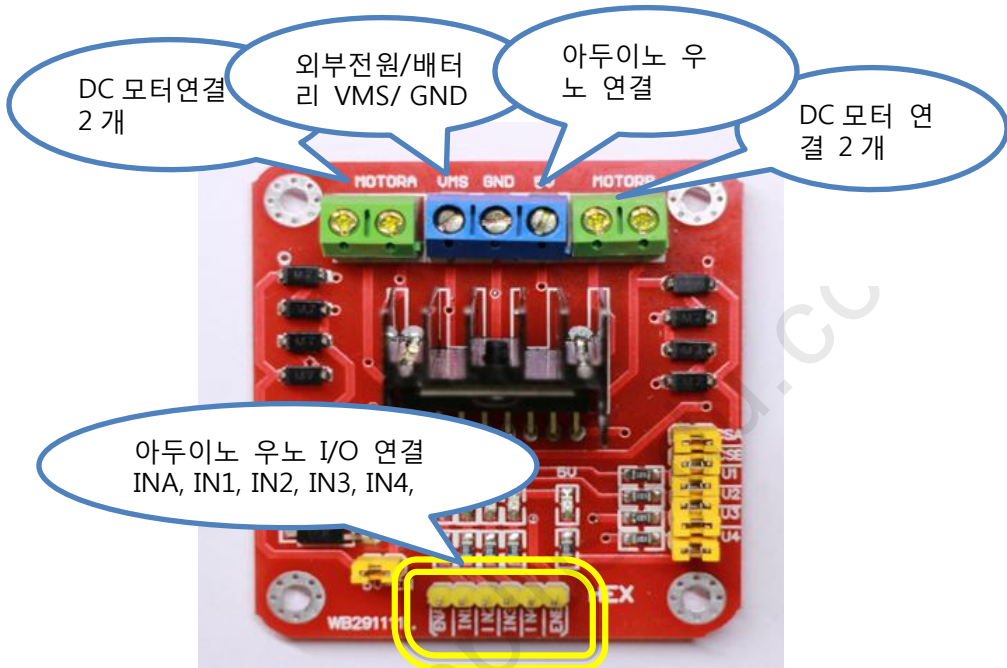


4.1.2 L293N 제어보드와 모터 케이블 연결

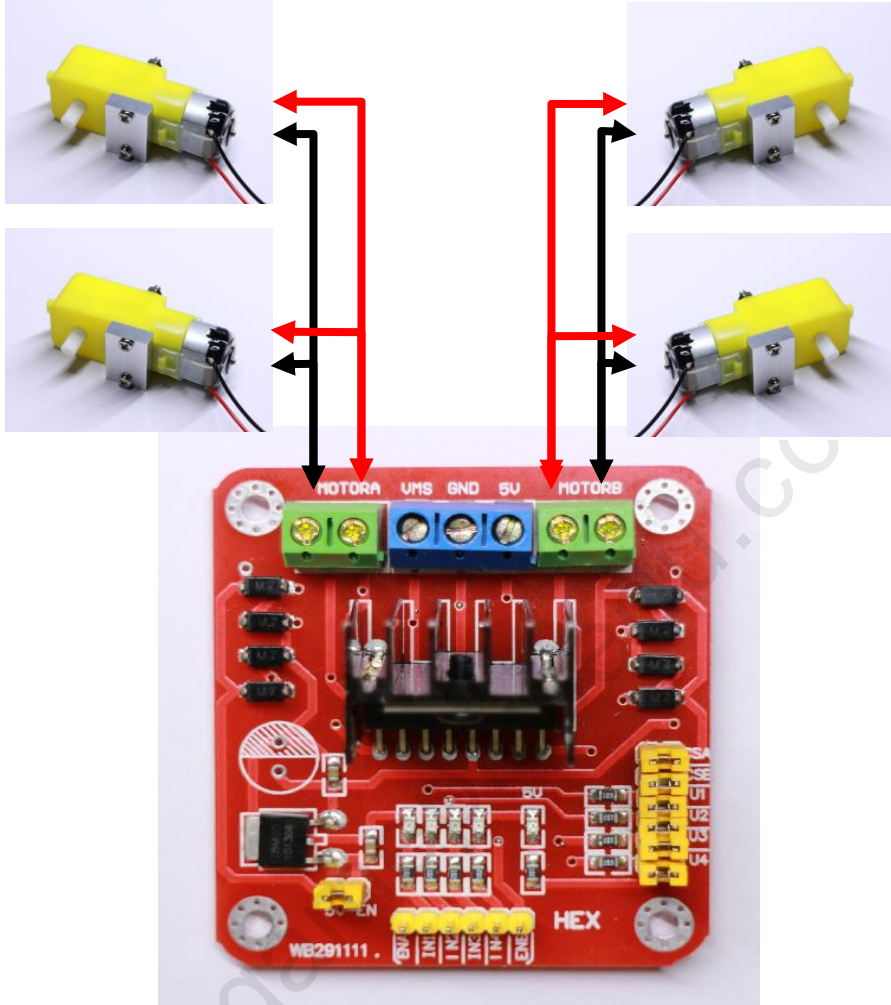
모터 케이블을 같은 방향 모터끼리 구멍을 통해서 윗쪽으로 뽑아냅니다. 다음은 L298N 모터제어보드 고정 볼트/너트를 조립하고 L293N 제어보드를 장착합니다.



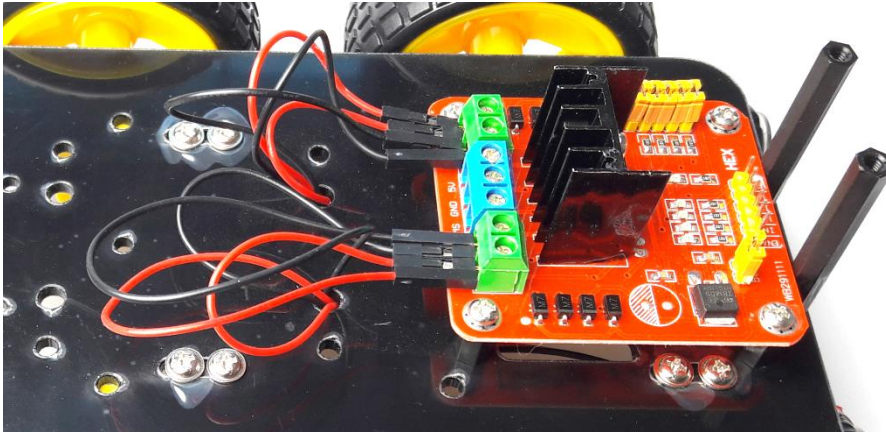
L293N 모터 드라이버 모듈은 DC 모터 방향, 속도를 제어하기 위해 사용하며 한 모듈로 2 개의 모터를 제어할 수 있습니다. 연결방법은 아래와 같습니다.



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.

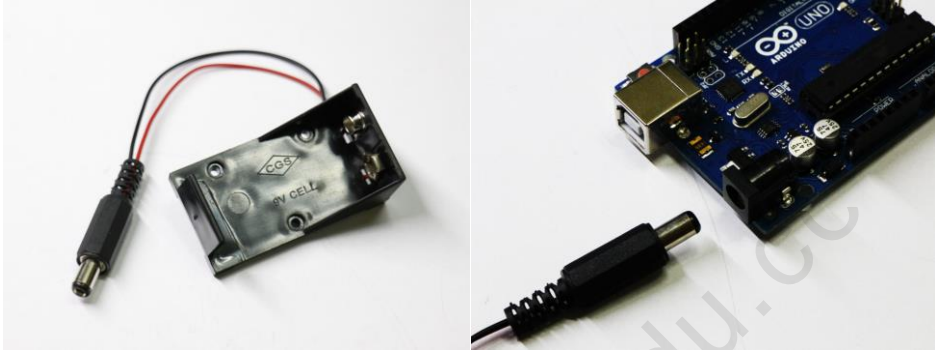


L293N 제어보드에서 MotorA, MotorB 포트 위에 있는 나사를 약간 푼다음 모터 케이블을 해당 위치에 꼽고 나사를 조여서 케이블이 빠지지 않도록 고정시킵니다. 너무 강하게 조이면 포트부분이 손상될 수 있으니 주의해서 작업합니다.

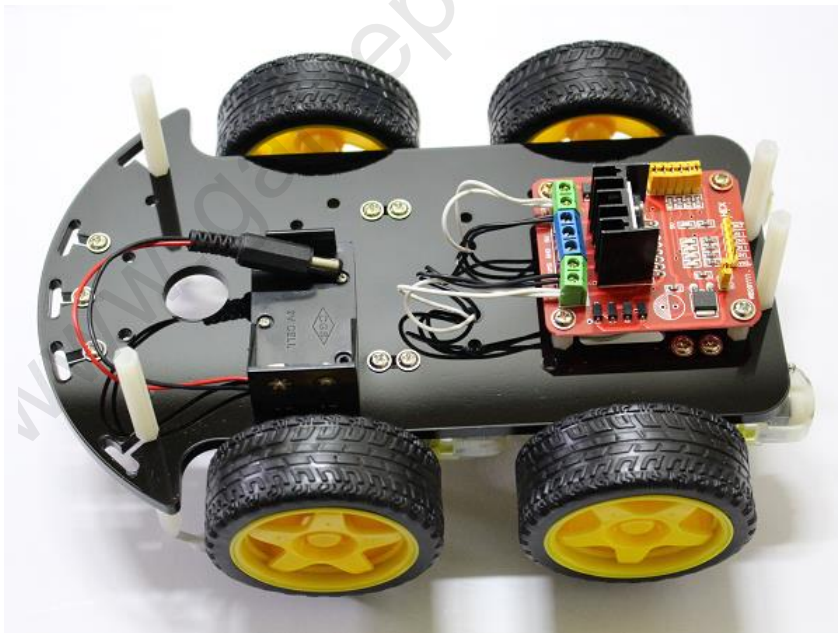


4.1.3 9V 배터리 박스

아두이노 보드의 DC 전원포트에 연결하여 사용하는 9V 배터리 홀더 박스를 조립합니다.



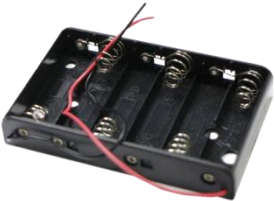


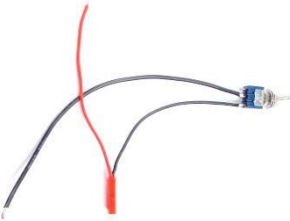


아래의 그림을 참조하여 고정을 합니다. 볼트 2mm, 너트를 사용하여 고정합니다. 1 곳, 2 곳만 고정하여도 무방합니다.



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
		
미니 브레드 보드	배터리커넥터/스위치	
		

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.



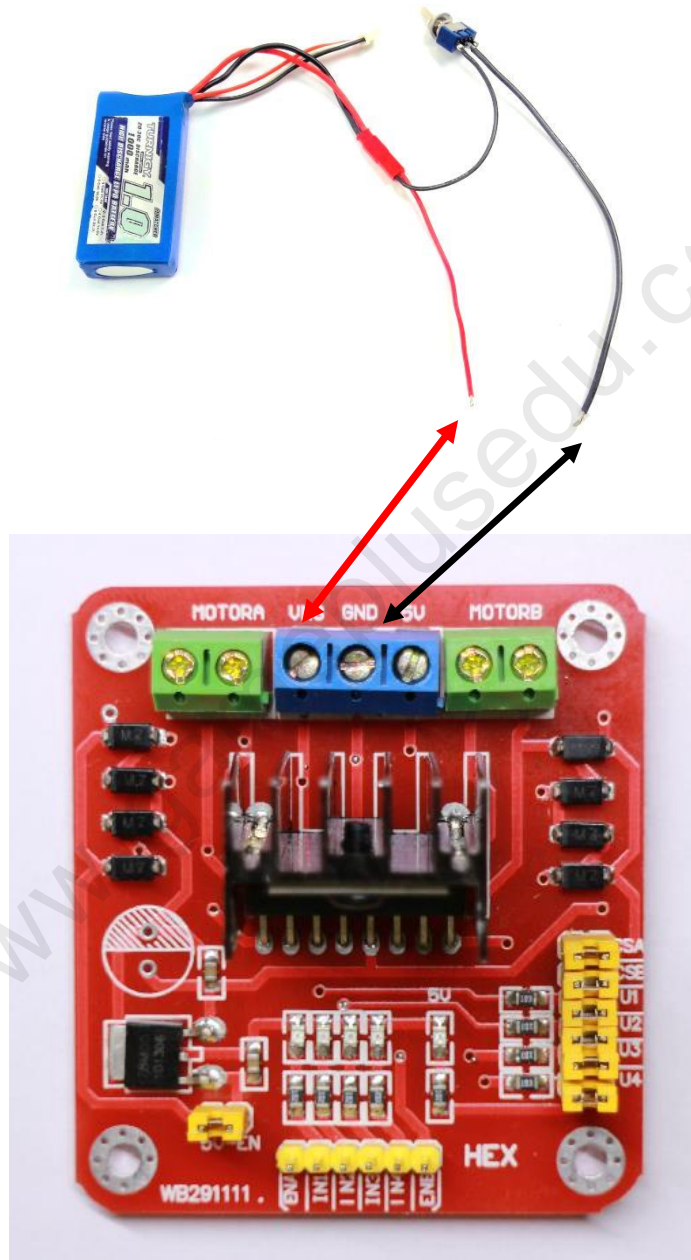
4.2.2 배터리-제어보드-센서실드 전원 연결

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 GND 에 연결하여 제어보드에 전원을 공급합니다. 상판 홀더를 통해서 케이블을 하판 제어보드에 연결합니다.

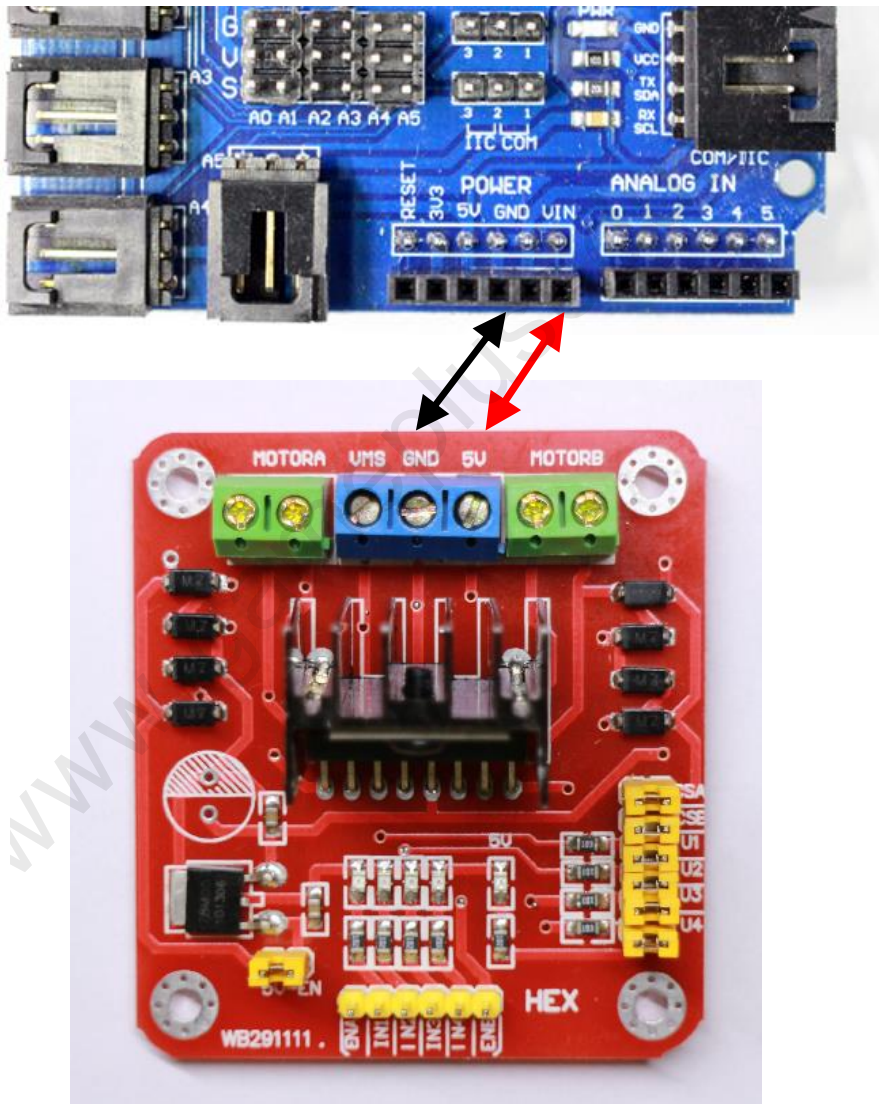


리포배터리를 이용하는 경우 아래 그림처럼 리포전용 커넥터와 스위치를 모터 드라이브와 연결합니다.



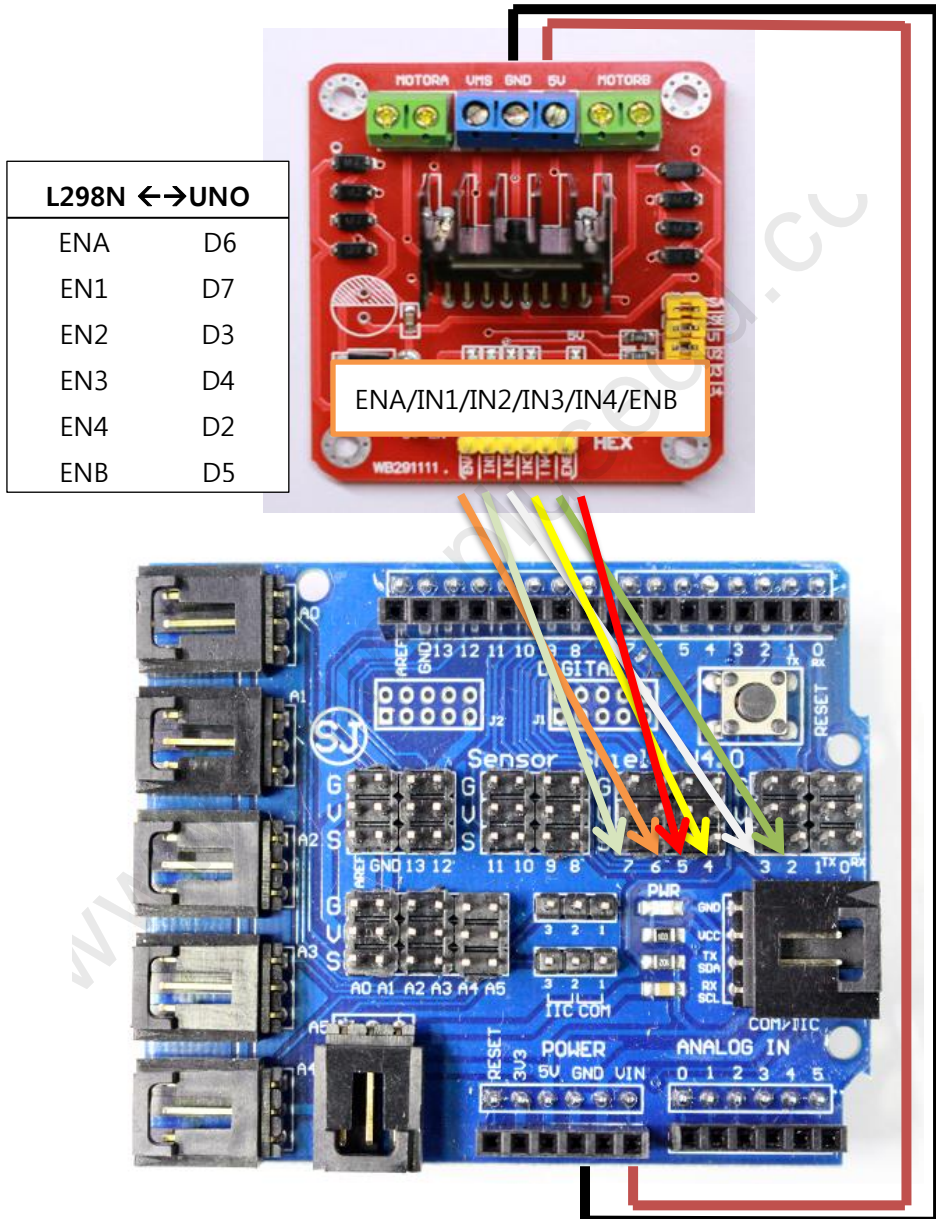
6xAA 배터리 박스를 설치한 경우 박스에 연결된 적색 케이블은 '+', 검정색 케이블은 '-' 입니다. 리포 배터리를 이용하는 경우는 M-M 듀풍 케이블로 제어보드와 배터리를 연결하는데 반드시 **양극, 음극이 바뀌지 않도록 주의**해야 합니다.

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

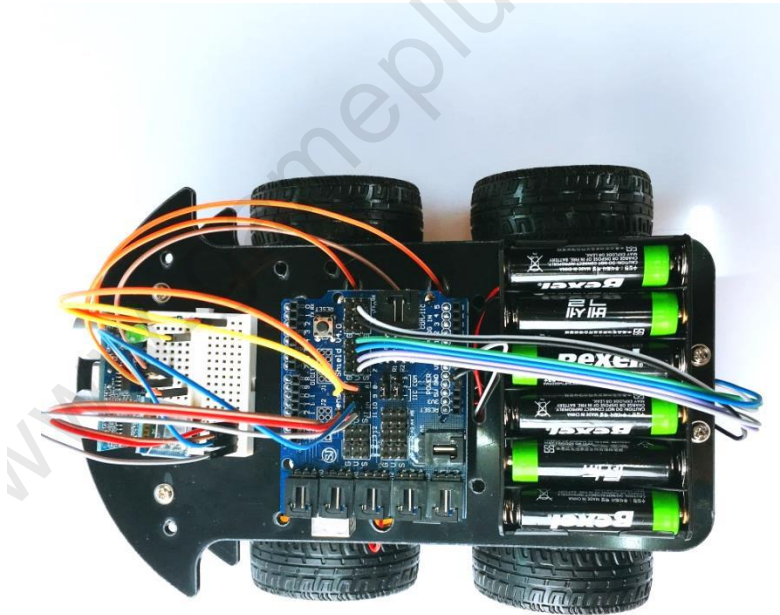
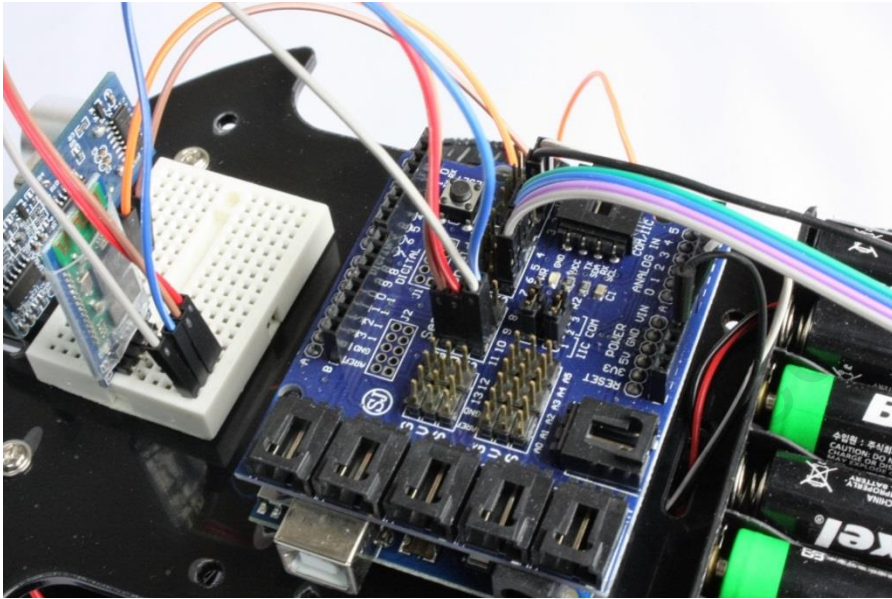


4.2.3 제어보드-센서실드 제어포트

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



ENA, ENB 는 PWM 지원포트로 연결하여 속도를 조절합니다.



4.3 자동차 주행 테스트

자동차 조립을 끝내고 실제 주행에 필요한 펌웨어와 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 1 회 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);
}
}

```


5 블루투스 주행 제어

스마트폰의 블루투스 통신 기능을 이용하여 스마트 자동차 주행을 제어해 봅니다. 본 키트에 포함된 아두이노 보드와 블루투스 모듈을 사용하여 자동차 주행을 제어 할 수 있습니다.

블루투스 HC-06	미니 브레드 보드	듀폰케이블 F to F
		

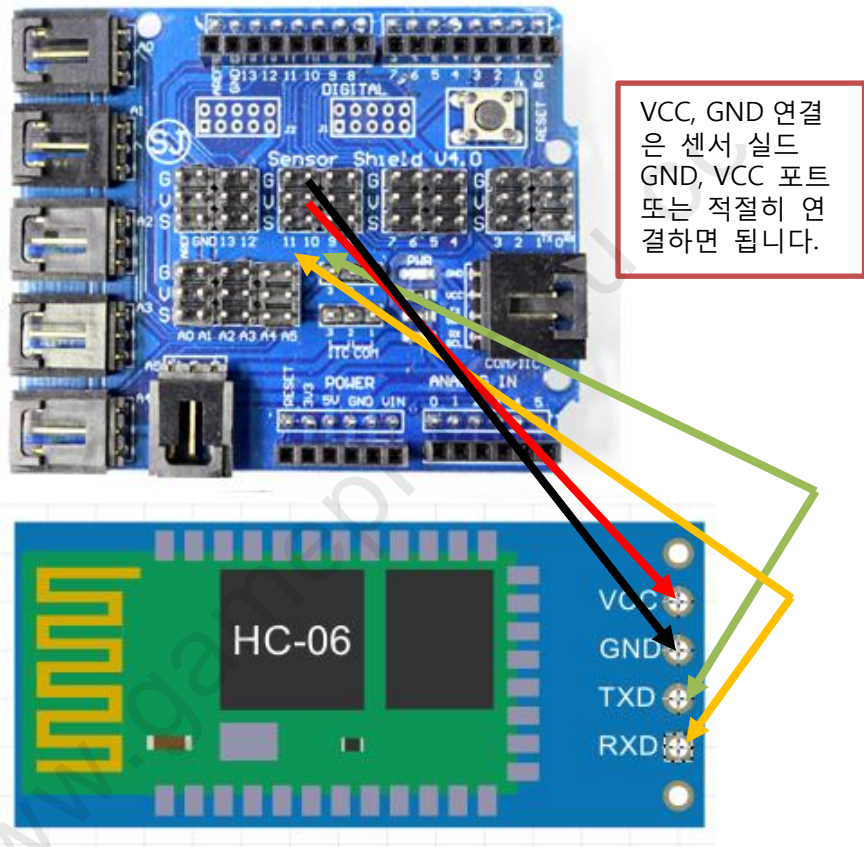
5.1 블루투스란

블루투스란 근거리 무선 통신 기술 규약을 의미합니다. 휴대용 기기, 근거리에서 사용되는 독립된 장치 등에 많이 사용됩니다. 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.



5.2 블루투스 모듈 연결

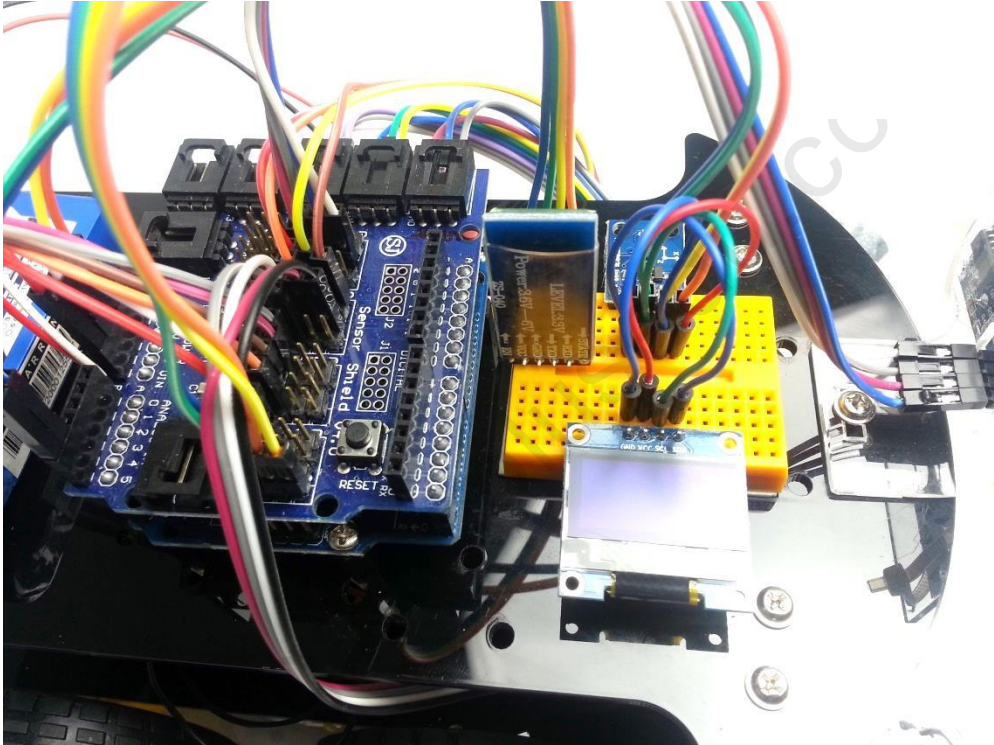
키트에 들어 있는 듀풍케이블을 이용하여 블루투스 모듈(HC-06)을 자동차의 센서실드와 아래와 같이 연결합니다. 스마트 로봇 자동차는 실드 10(RX/수신), 11(TX/송신) 핀을 블루투스 통신에 사용합니다.



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 송신/수신 핀을 변경하는 경우 소스 코드도 수정해야 합니다.



5.3 스마트폰 블루투스 페어링(PAIRING)

스마트폰에서 자동차의 블루투스 모듈에 접속하려면 먼저 블루투스 페어링을 해야 합니다. 스마트 로봇 자동차에 전원을 넣고 블루투스 모듈에 빨간 불이 들어 오는지 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-06 으로 표시됩니다. 같은 공간에 있는 모든 블루투스 디바이스가 동일한 이름을 사용하여 나중에 자신의 디바이스 식별이 어렵습니다. HC-06 을 자신만의 이름으로 변경하세요.

5.4 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()
{

```

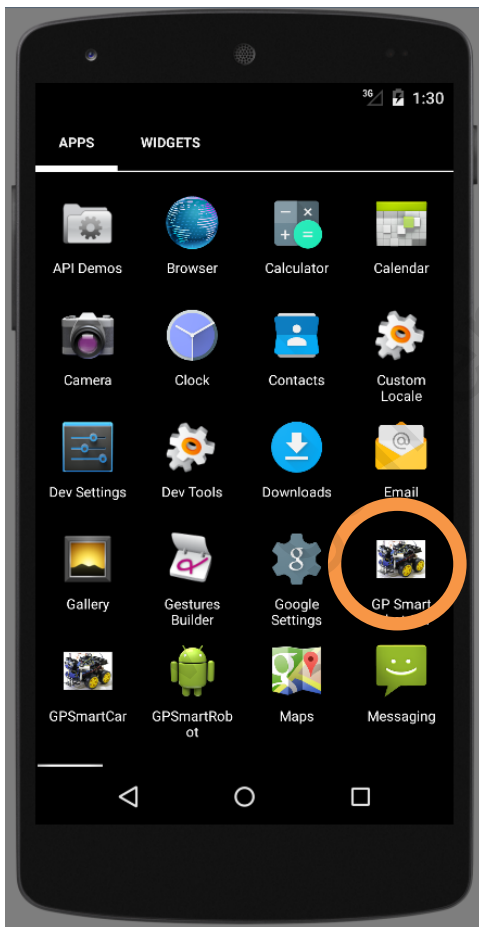
```
if (btSerial.available()) {  
  
    unsigned char cByte;  
  
    cByte = btSerial.read();  
  
    SerialCommData.addPool(cByte);    // store the serial input to Buffer  
  
    process_SerialCommModule();      // parse and change the input value  
                                     // to MOVE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

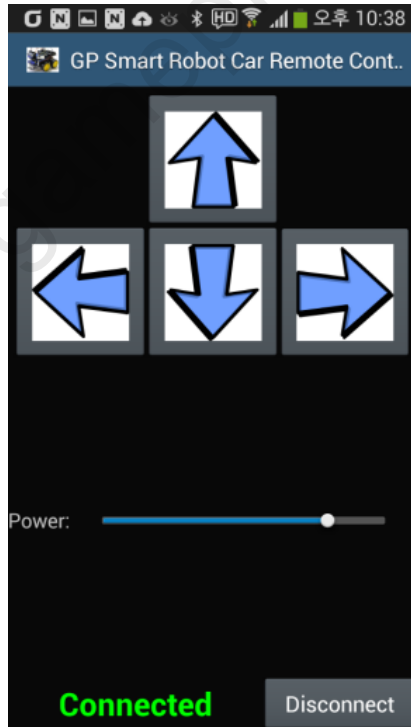
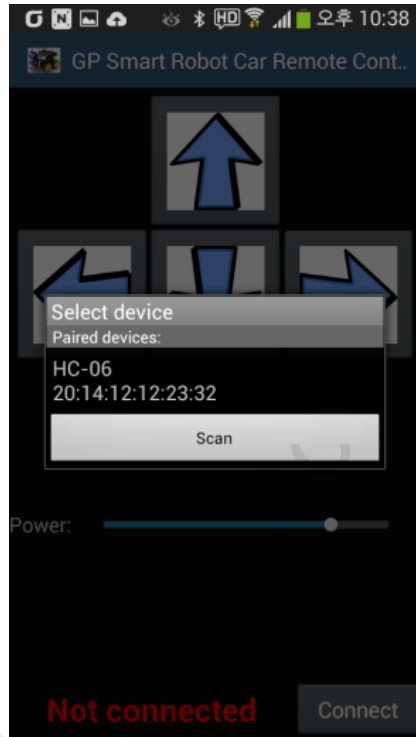
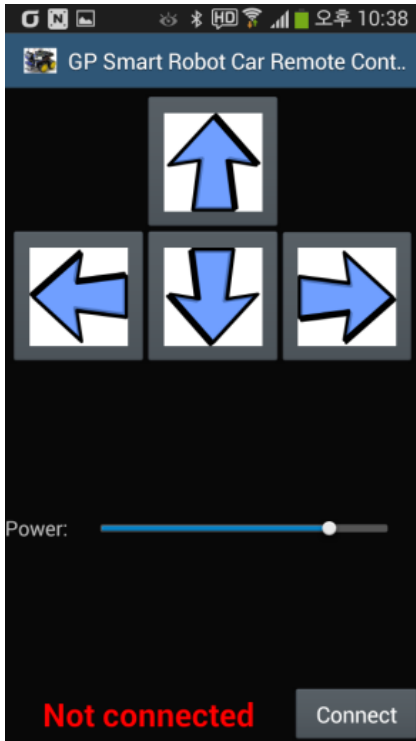
< 사이트에서 스케치 코드를 다운로드 받아 사용 할 수 있습니다 >

5.5 안드로이드 앱

스마트 자동차 주행 제어에 안드로이드 앱을 이용할 수 있습니다. 게임플러스에듀 사이트(www.gameplusedu.com) 게시판에서 앱을 다운받아 설치하세요. (아이폰은 지원하지 않습니다)

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





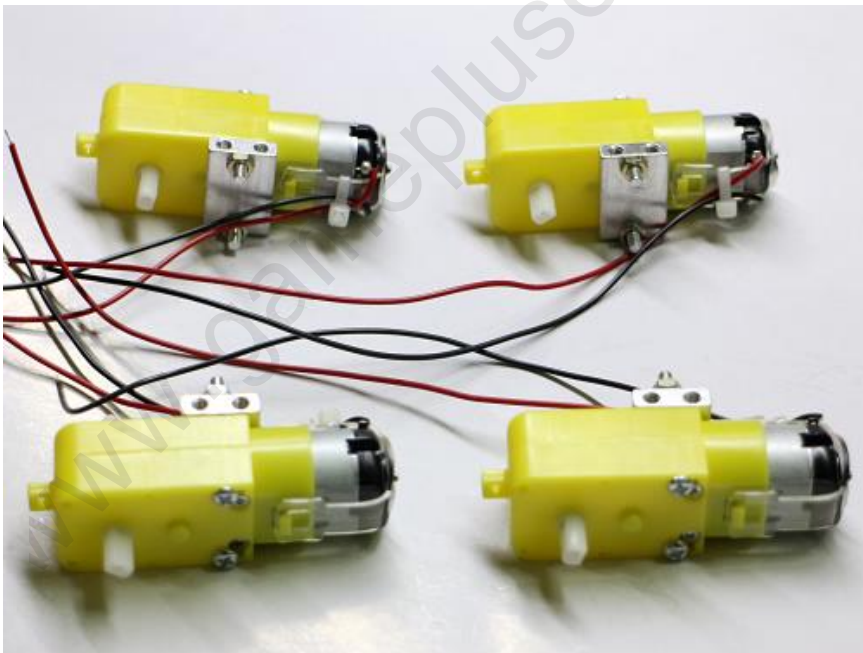
6 DC 모터

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.

6.1 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.

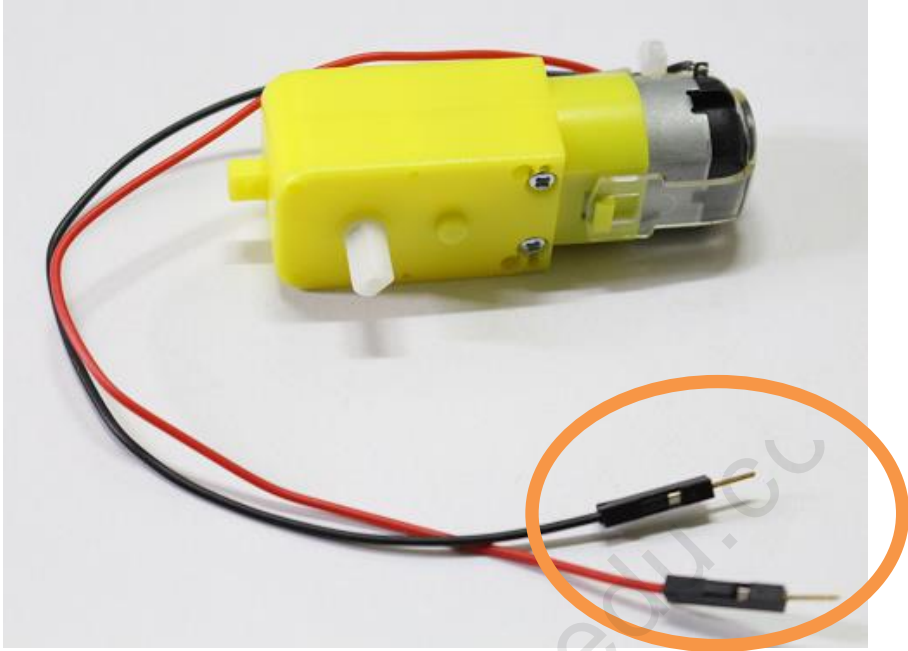
키트에 사용되는 DC 기어 모터입니다.



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.

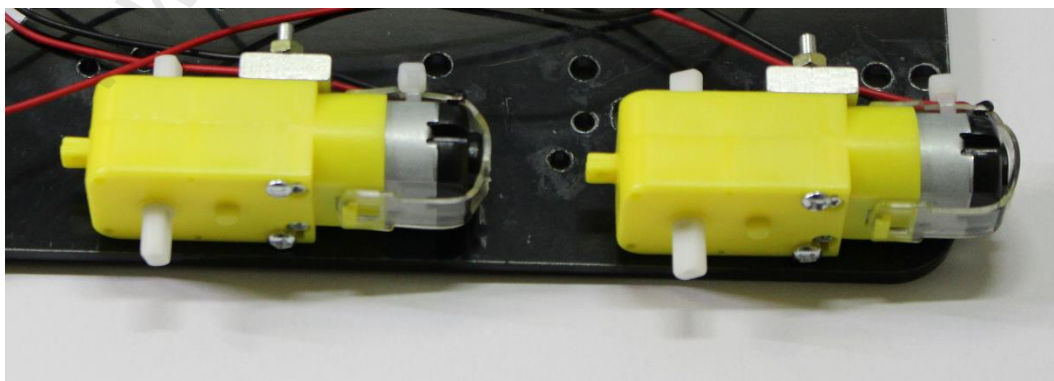
6.2 DC 모터 전원 연결 케이블 연결 시 주의 사항

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.

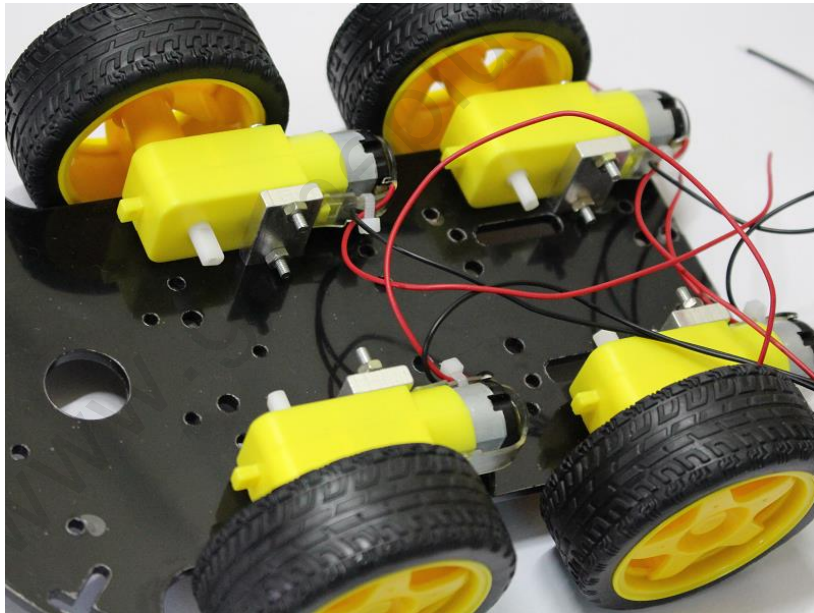


6.3 DC 모터 리드선 연결 시 리드선 주의 사항

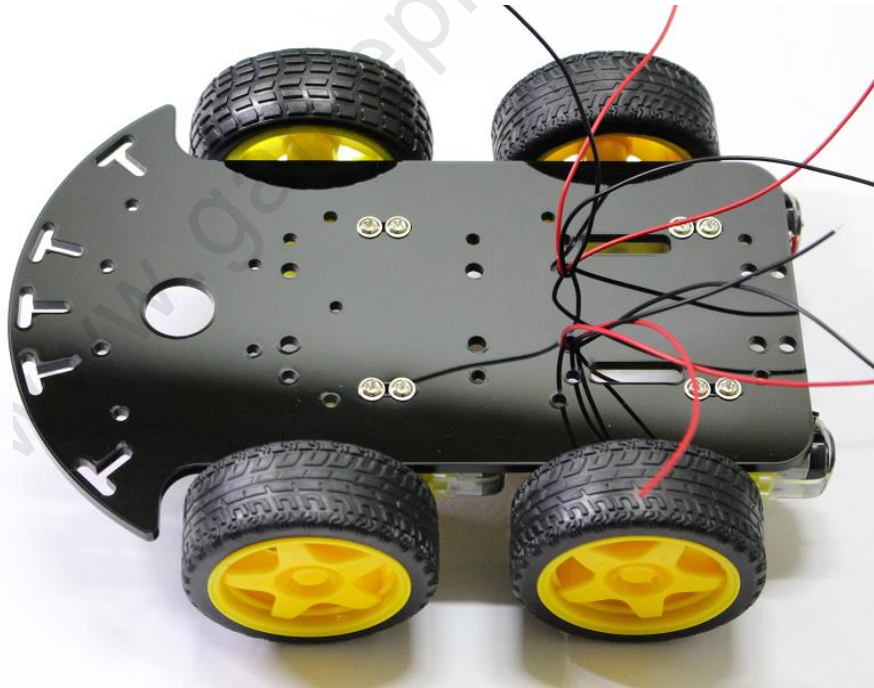
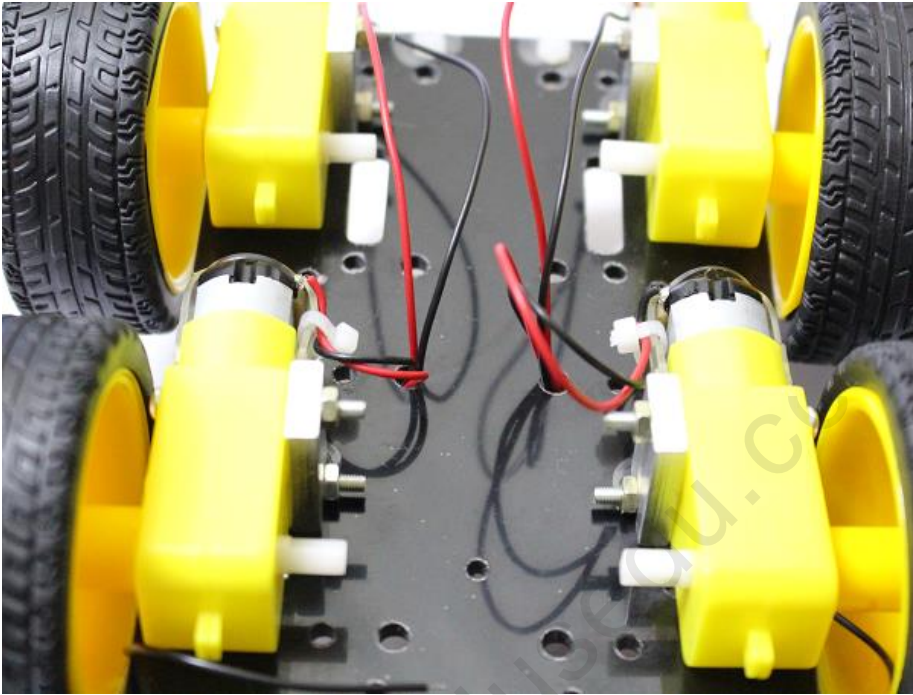
부품 입고 순서 및 사정에 의해 DC 모터 리드선의 색상이 다를 경우에는 납땀이 가능하다면 전선을 분리, 일괄적으로 배선 순서를 다시 납땀 해주어도 무방합니다. 그냥 사용하는 경우에는 L298N 제어 보드의 DC 모터 전원 연결 배선할 때 리드선의 연결된 위치를 염두에 두시고 배선하기 바랍니다.

6.4 DC 모터 전원 연결선 정리

하판 새시 프레임에 DC 모터 & 타이어 고정 후 그림입니다. 뒷면으로 전선들을 홀더 사이로 넣어 줍니다.



전선들을 위로 올려 주는 홀더 위치 선택은 적절히 해줍니다.



7 L298N 듀얼 모터 제어 드라이버

DC 모터의 회전 및 속도를 제어하기 위해 사용하는 모듈입니다. L298N 는 모터 제어에 많이 사용됩니다. 모터 포트의 극성을 변경할 수 있으며 전압 조절 포트 로 입력 전압에 의해 모터 출력 포트로 전압 조절이 가능합니다.

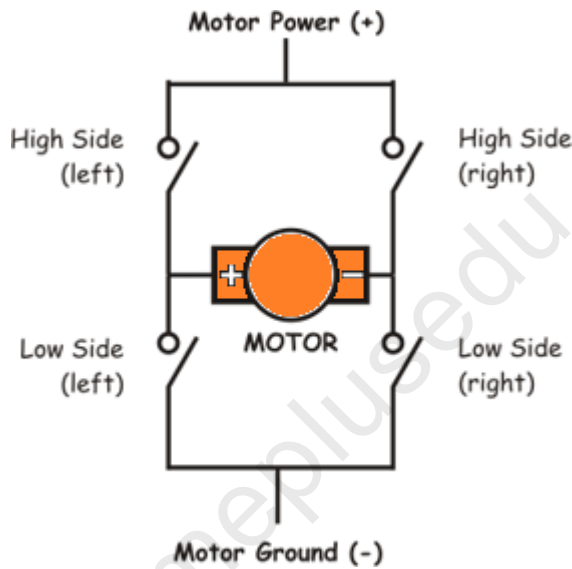


2 개의 DC 모터 포트가 있으며 최대 2A 까지 지원 합니다. 전원 입력은 VMS 포트에 **5 V** 부터 **35 V** 까지 지원 되어 많은 종류의 DC 모터를 구동 시킬 수 있습니다. 권장 전압은 5V 부터 12V~24V 까지입니다.

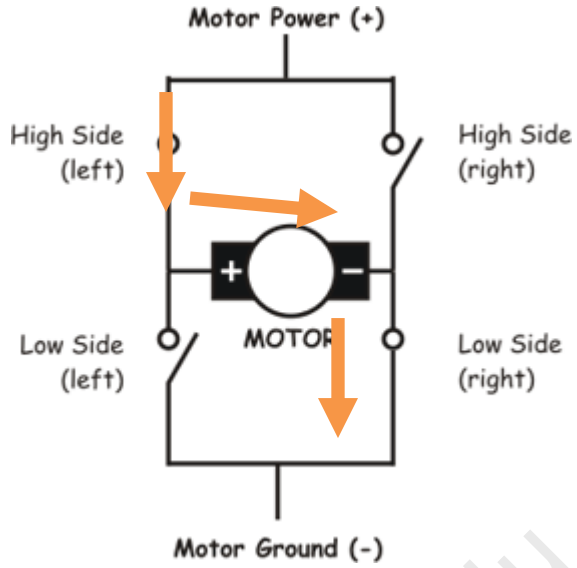
L298N 듀얼 H-Bridge 모듈은 2 개의 모터를 제어 하는데 사용합니다. 정방향/역방향 변경 가능한 DC 모터 전원 연결 포트가 2 개 있습니다. DC 모터는 전원 연결시 (+)(-) 극성 구분이 없습니다. 연결된 (+)(-)를 반대로 연결하면 회전 방향이 역방향이 됩니다.

7.1 H-BRIDGE 회로

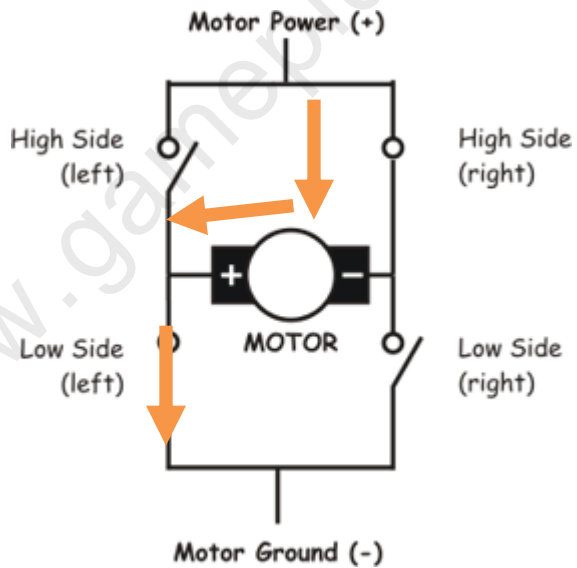
H-BRIDGE 전류의 흐름을 정방향/역방향 변경해 주는 회로의 정식 명칭이기도 합니다. 알파벳 H 문자와 비슷합니다. DC 모터 연결 후 전류의 흐름을 바꾸어 주면 모터의 회전 방향이 바뀌게 됩니다.



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

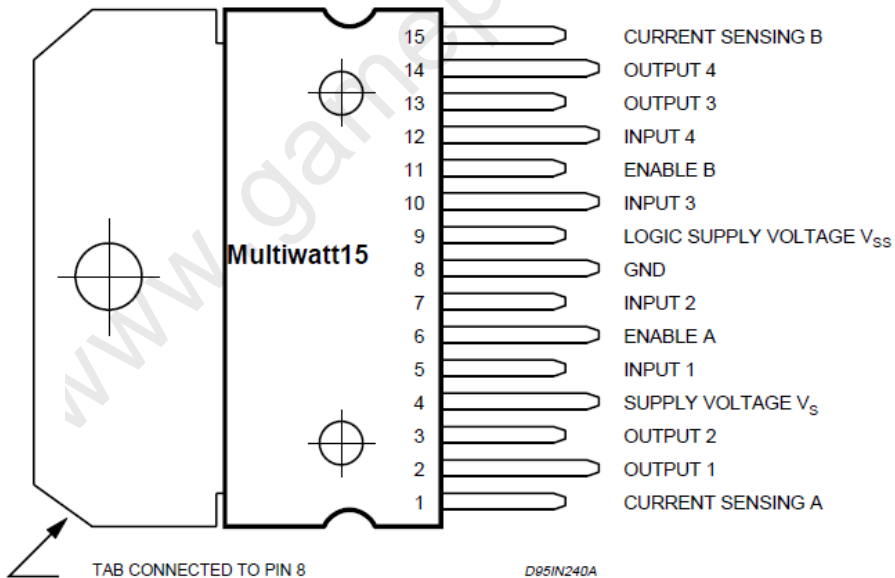


DC 모터 방향 변환 음극에서 양극으로 흐를 경우

7.2 L298N 모터 제어 보드

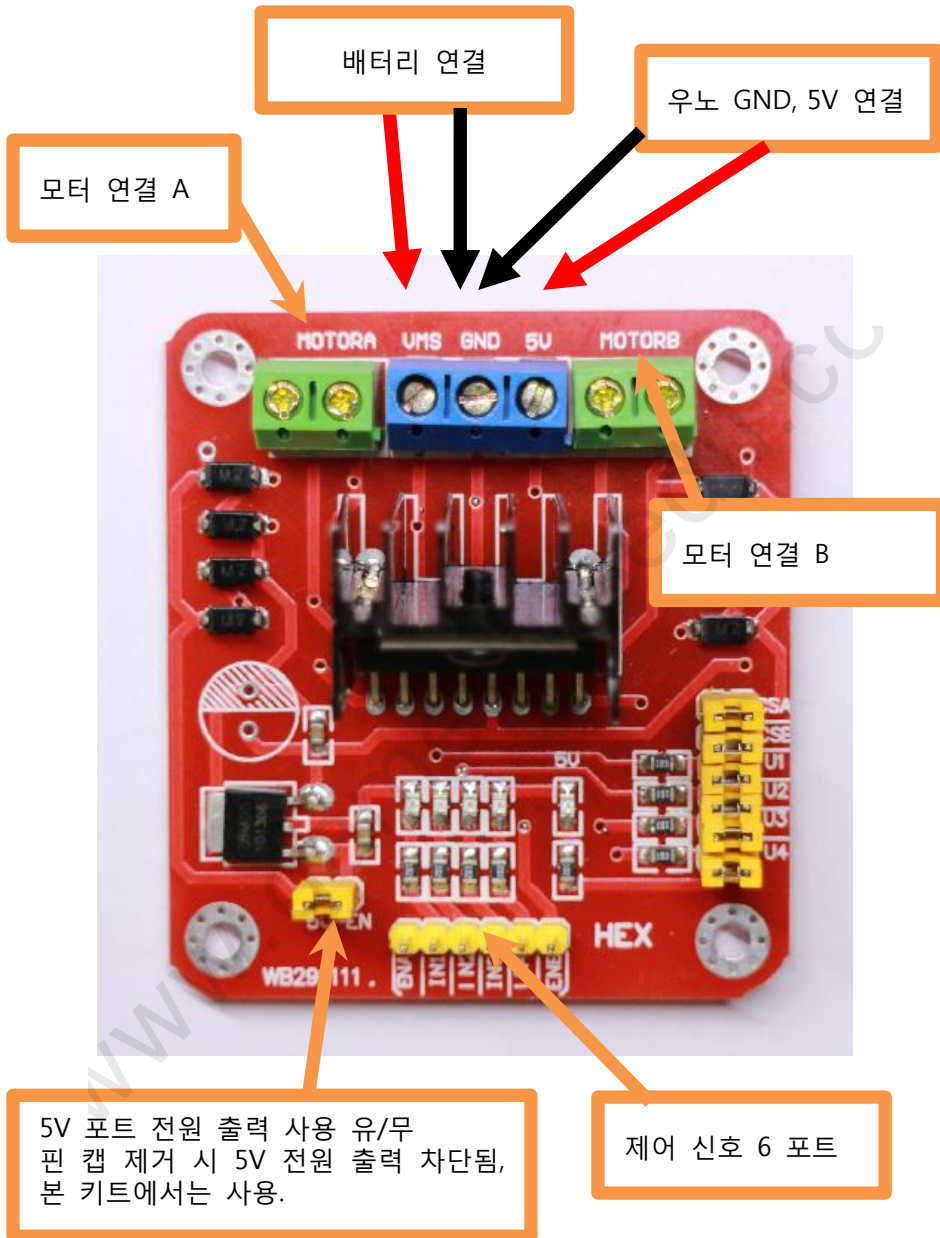
L298N 제어 보드 사양은 아래와 같습니다.

- 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.

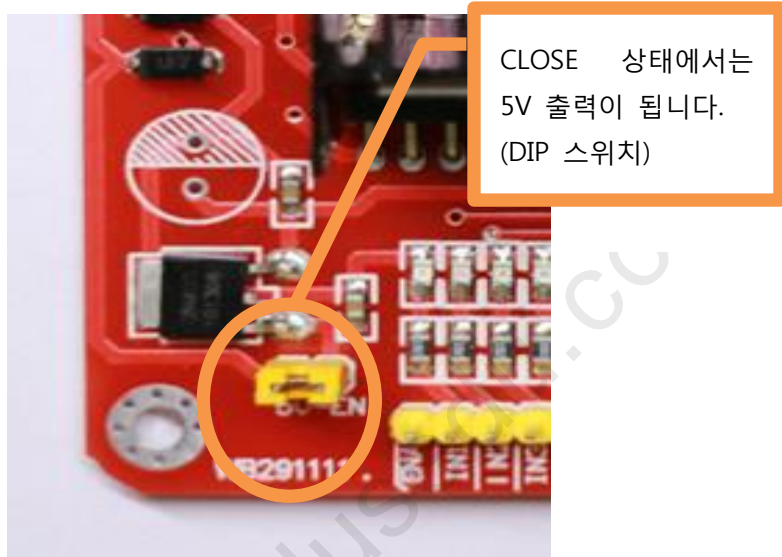
만약, 5V_EN 을 사용(CLOSE 상태)하는 상태인 경우 L298N 의 5V 포트와 우노 R3 (센서 실드 V4)의 VIN 포트에 연결하고 우노 R3 에는 별도의 전원 입력(9V 배터리)을 하지 않아도 됩니다.

사용시 적절히 선택하시기 바랍니다.

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

7.3 5V_EN 포트 사용 설명

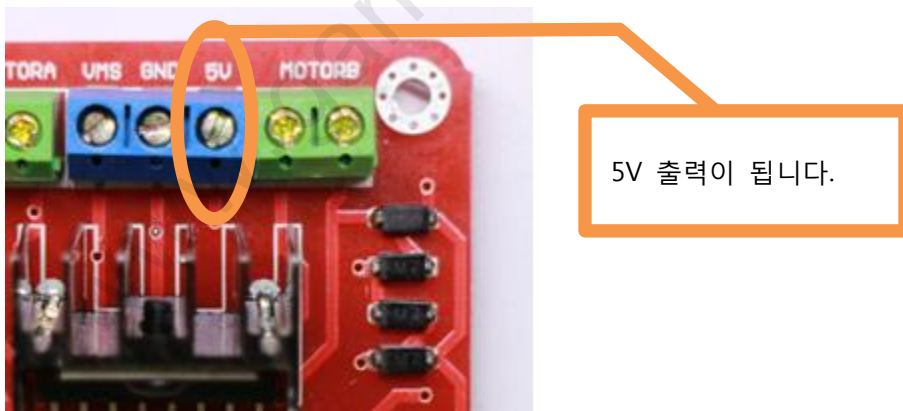
5V_EN 포트의 스위치가 닫혀 있는 상태입니다.



CLOSE 상태에서는
5V 출력이 됩니다.
(DIP 스위치)

5V_EN 포트 위치

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



5V 출력이 됩니다.

반대로 오픈 상태이면 5V 포트에 전원 출력이 안됩니다.
본 키트에서는 5V_EN 포트 닫혀진 상태로 사용하면 됩니다.

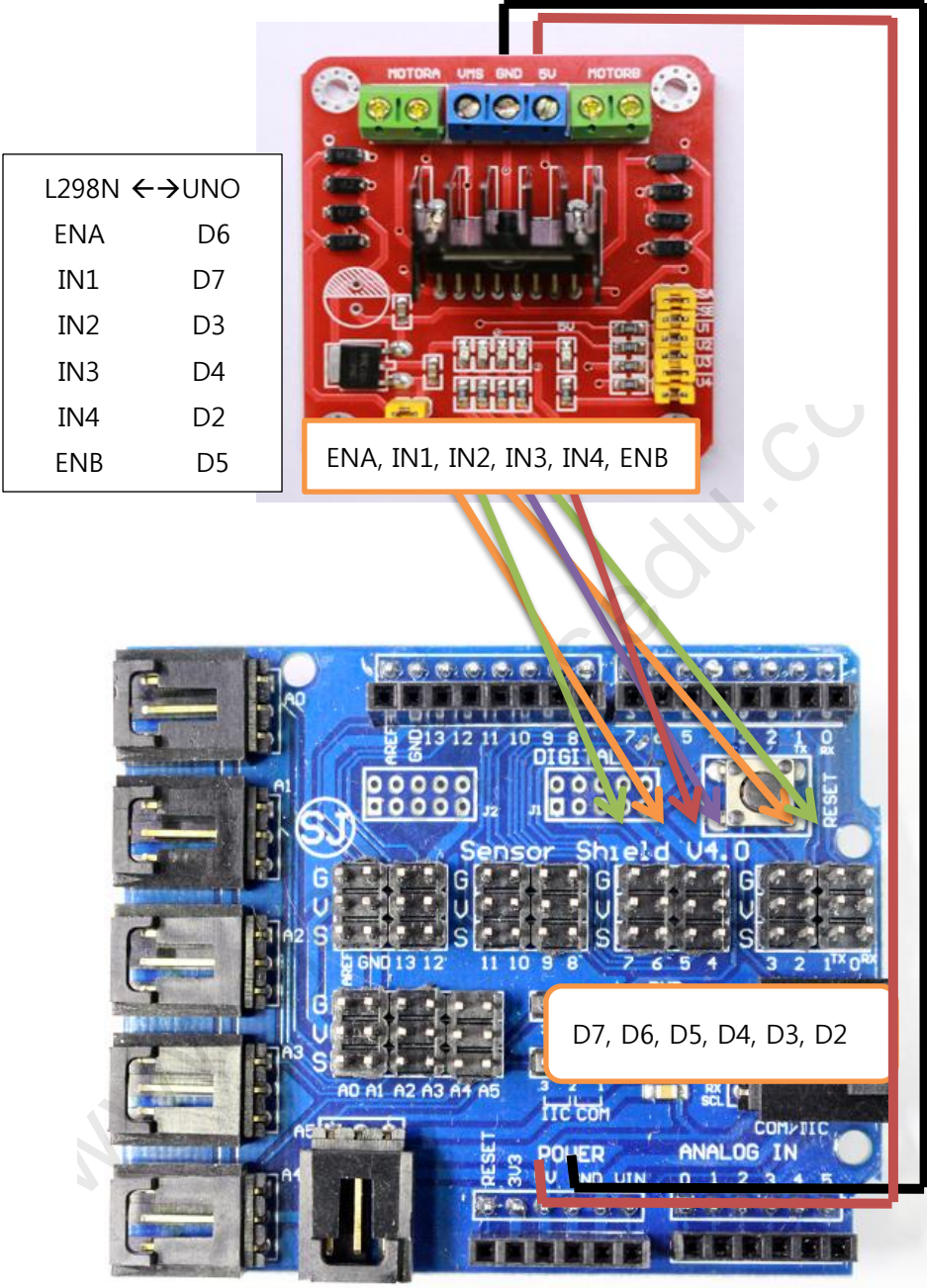
7.4 아두이노 우노 R3 & L298N 제어 포트 연결

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

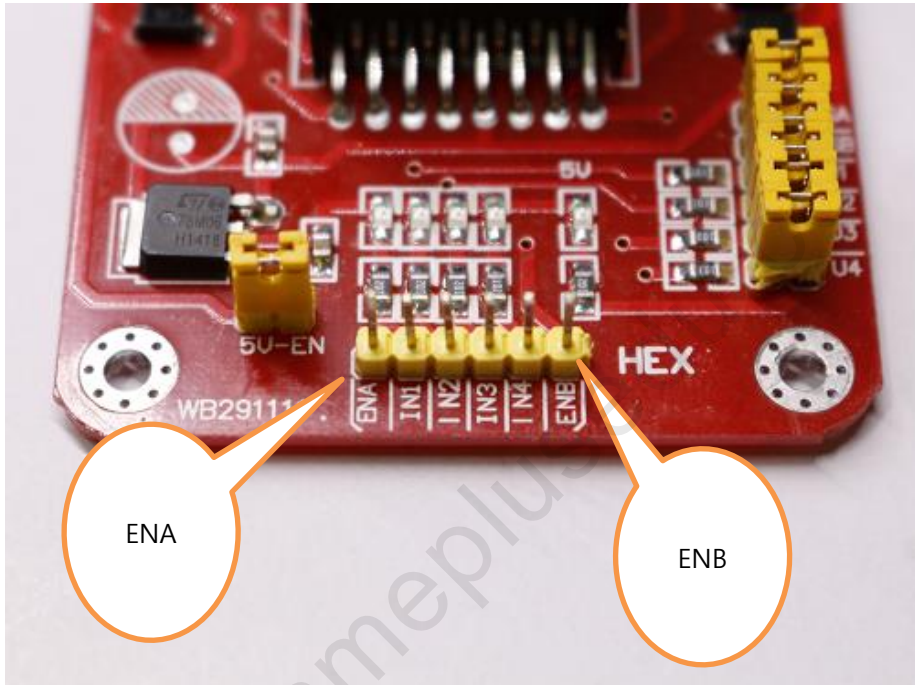
4WD 스마트 로봇 자동차를 만들기 위한 초음파 센서, 서보모터, 적외선 수신모듈, L298N 제어 보드와 연결됩니다. 연결하기 편한 GVS 포트를 사용하여도 되며, 기존 아두이노 핀배열 형태의 Female 헤더도 지원됩니다.



L298N 센서 실드 연결 설명도

7.5 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 위해서 입니다. 아두이노의 PWM 기능의 포트는 analogWrite 함수를 사용하여 전압 조절이 가능합니다. 본 키트에 사용되는 DC 모터는 3V~6V 사용 가능합니다. (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 ENB 는 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 왼쪽 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.

8 라인 추적 주행 (LINE TRACKING)

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.

라인트레이서 TCRT5000	절연 테이프	듀폰케이블 F to F
		

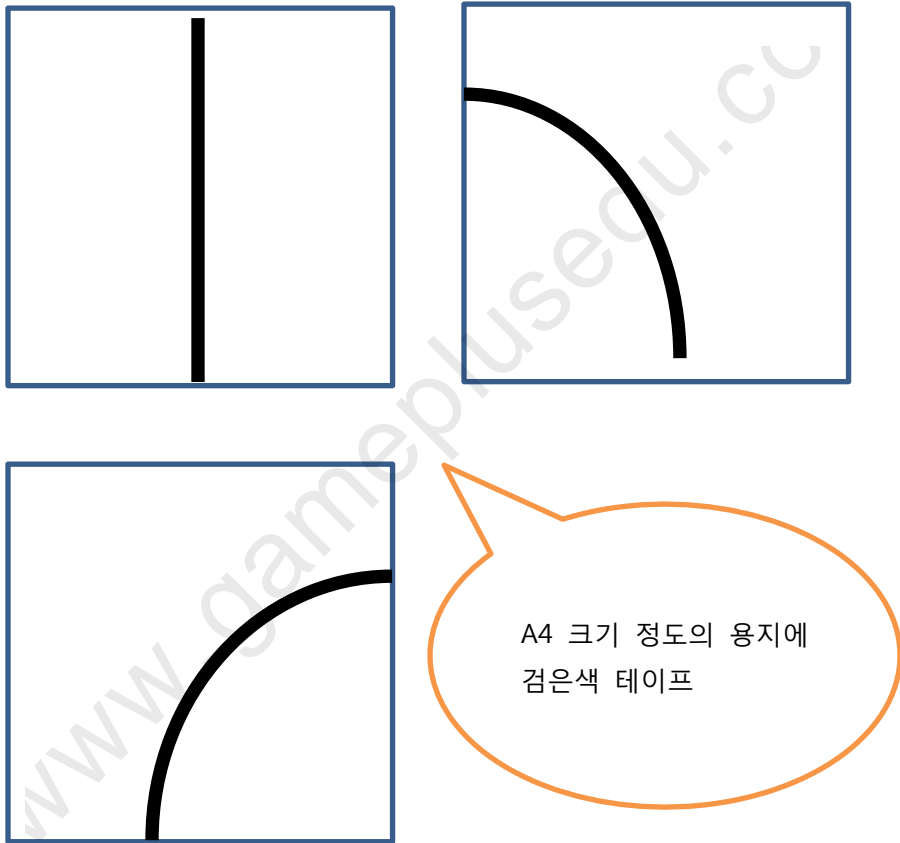
8.1 트랙 라인 설치

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.

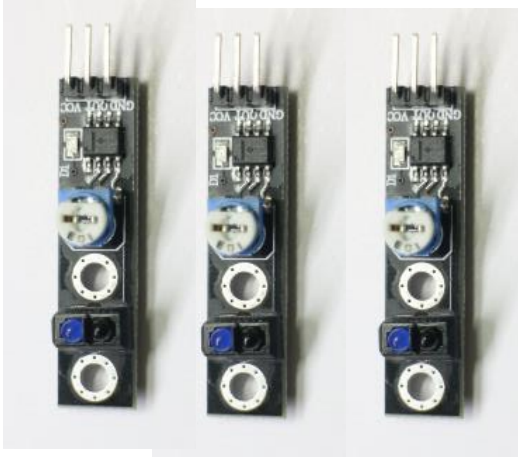
최적의 테스트 환경은 넓은 공간에 흰색 바탕에 검은색 유도선을 부착합니다.



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

8.2 라인 트레이서 모듈 기초 정보

라인 추적 모듈 TCRT5000 모듈 3 개를 사용합니다.



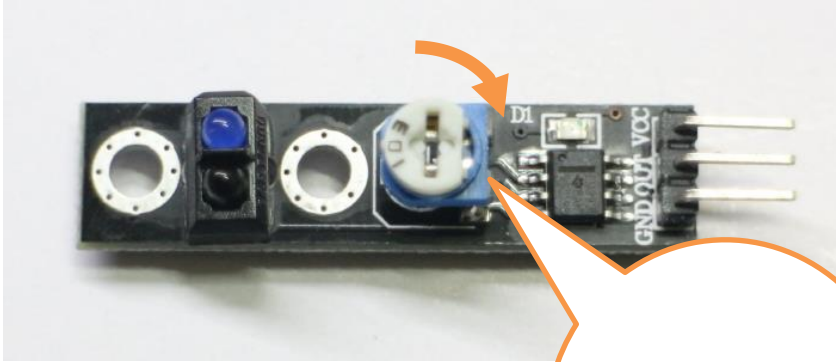
프로그래밍 하는 방식에 따라 3 개 이상 사용할 수도 있고, 2 개도 사용 가능합니다. 본 키트에 적용되는 예제는 3 개의 모듈을 사용하는 방식으로 기술 되어 있습니다.

8.2.1 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.

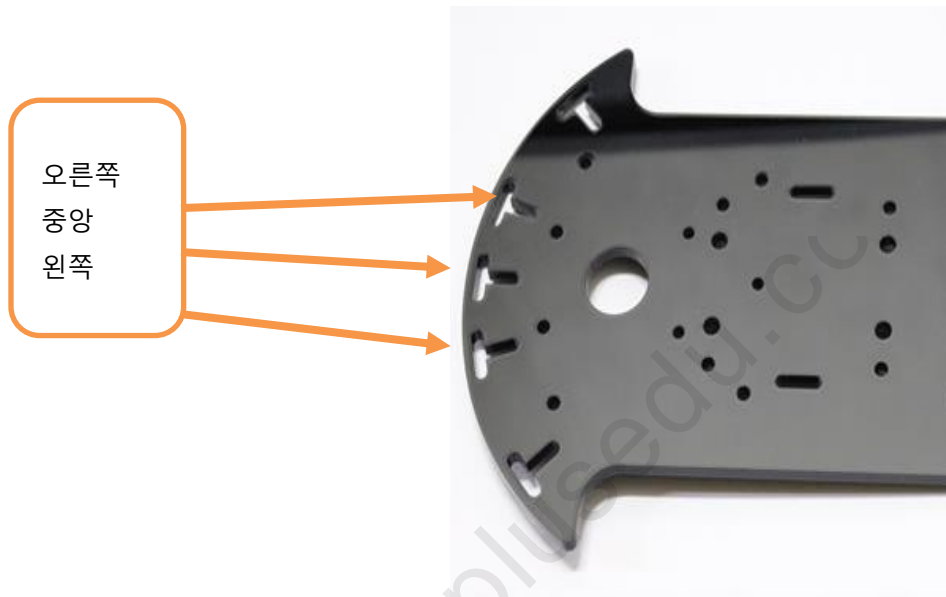


가변저항을 약간
돌려서 거리 조
절을 합니다.



8.3 라인 트레이서 모듈 장착

장착 위치는 하단 새시의 앞쪽 밑으로 장착을 하게 됩니다.

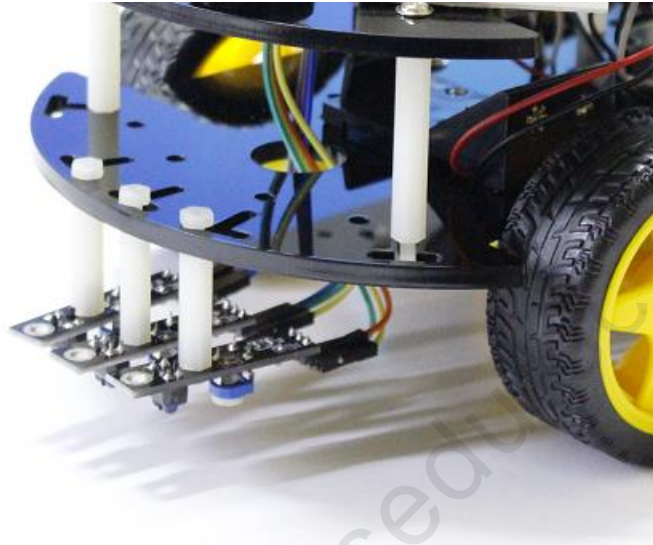


라인트레이서 고정 기둥 위치

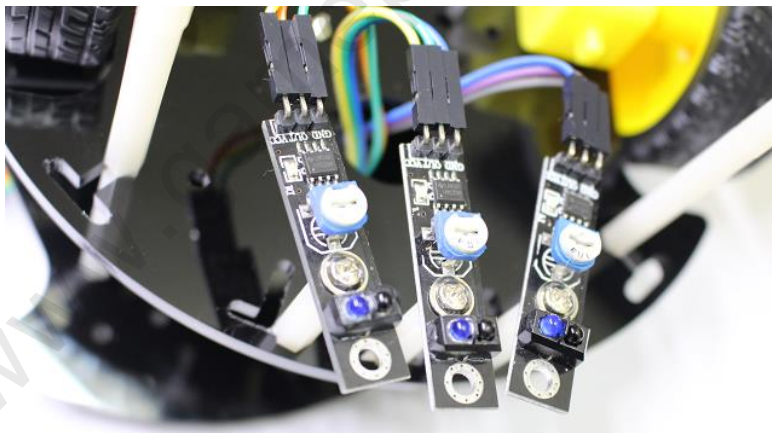


라인 트레이서 모듈 고정 시 사용되는 부품

하단 새시에 장착된 그림입니다.



라인트레이서 모듈 장착된 모습



장착 후 바닥에서 본 모습

8.4 라인 추적 운행 기초 원리

라인 추적 운행을 하기 위해 3 개 정도의 TCRT5000 모듈을 붙입니다. 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.



8.5 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

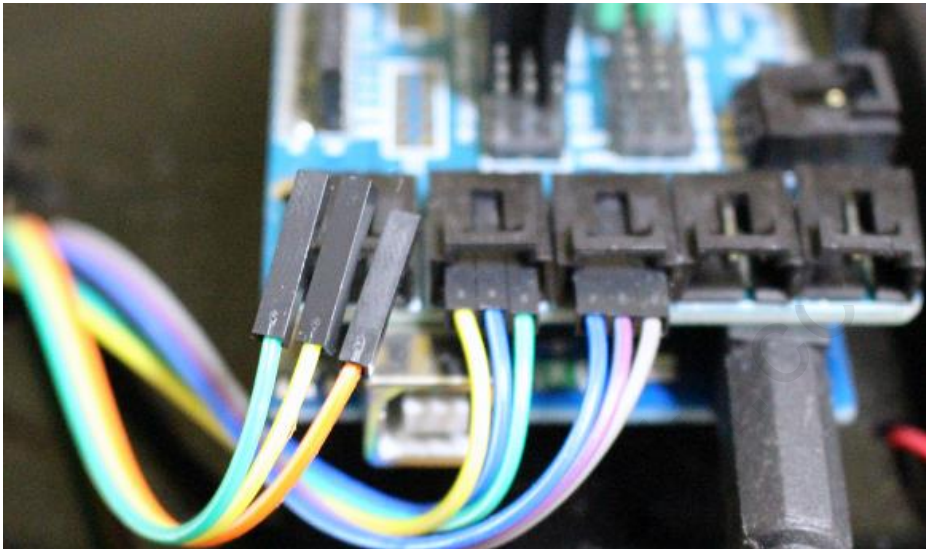
중앙 모듈

TCRT5000 모듈 중앙	아두이노
VCC	5V
OUT	A1
GND	GND

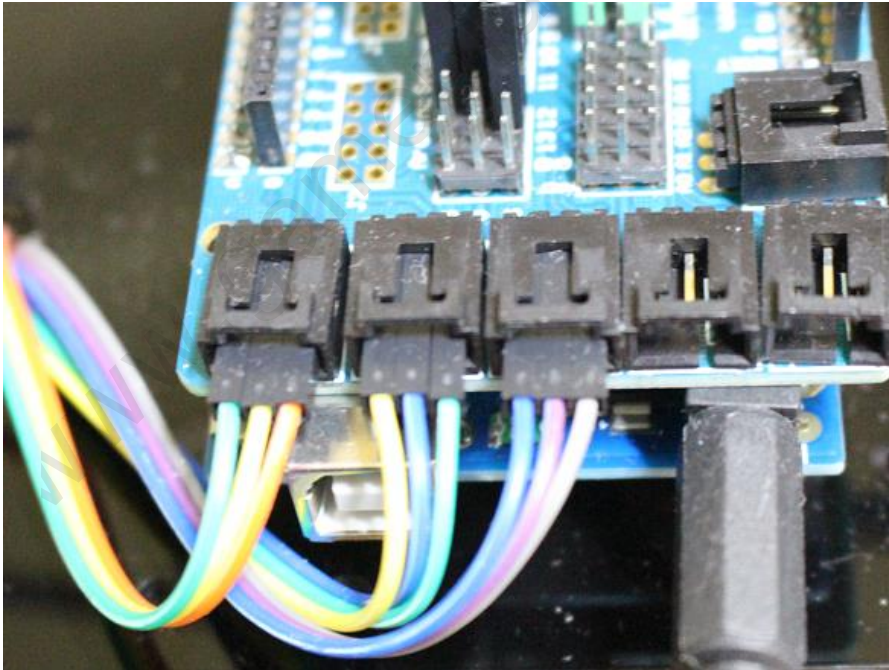
오른쪽 모듈

TCRT5000 모듈 오른쪽	아두이노
VCC	5V
OUT	A2
GND	GND

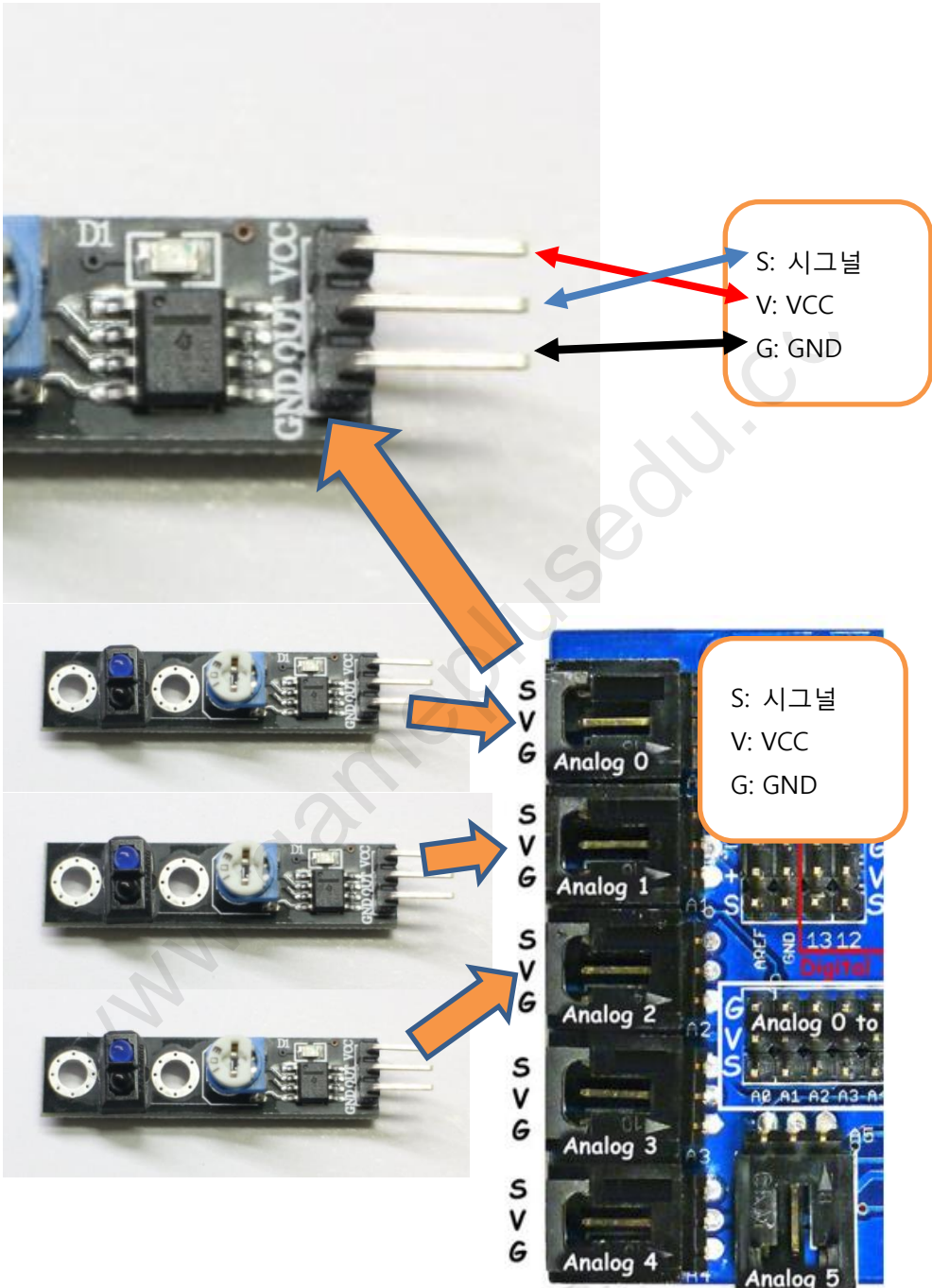
본 키트는 뒤풍 케이블 2.54pitch 제공되므로 3 가닥씩 잘라서 사용합니다.
Female To Female 형태의 케이블입니다.



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.



8.6 라인 추적 운행 코드

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

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

```
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.

```

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 실행 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 설정해야 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 1 회 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();
```

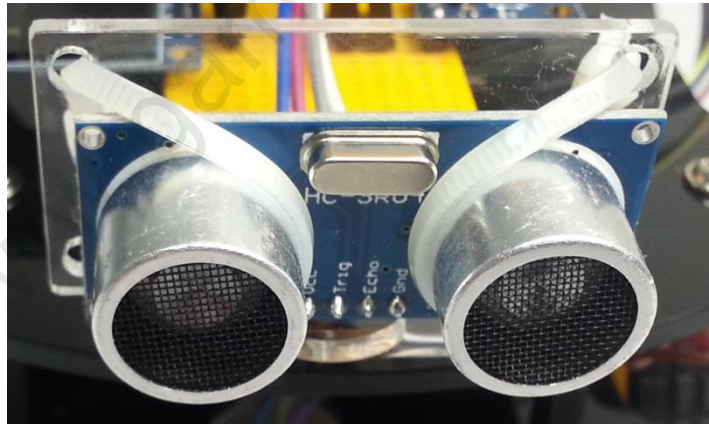
```
}
```

9 초음파 센서 프로그래밍

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



초음파 센서 모듈

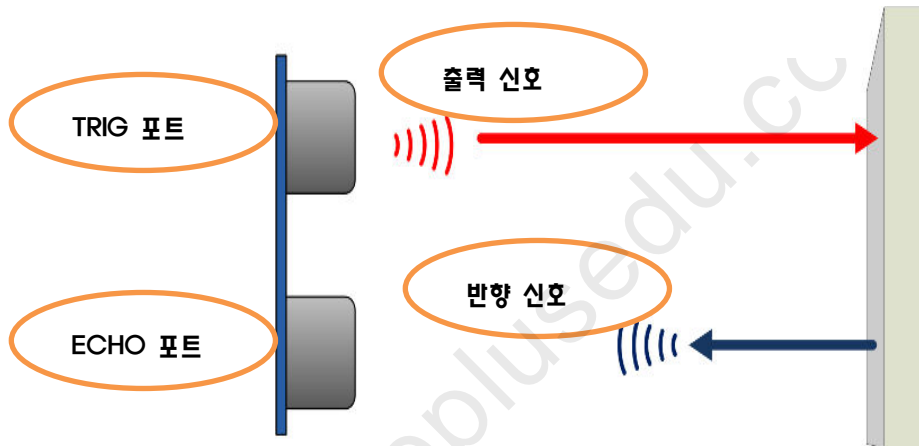


초음파 센서 모듈 거치대 사용 모습

9.1 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 잴 수 있습니다.

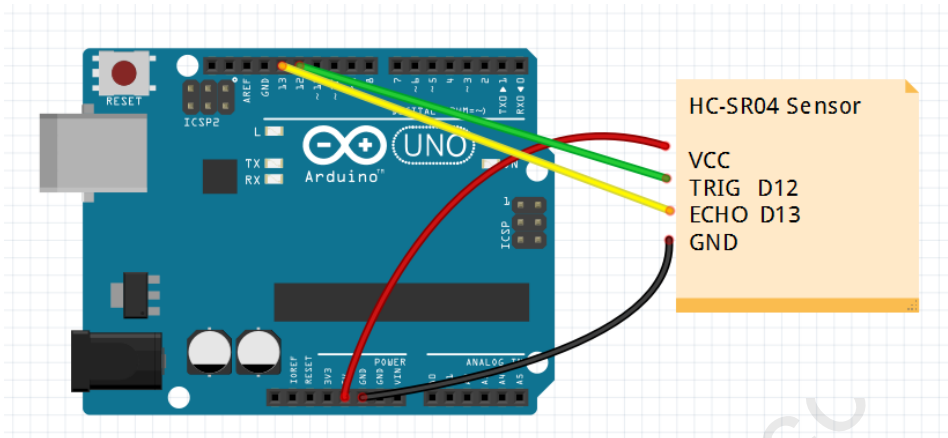
측정거리: 2cm ~ 5m (4m), 측정각도: 15 도, 측정 해상도: 3mm



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

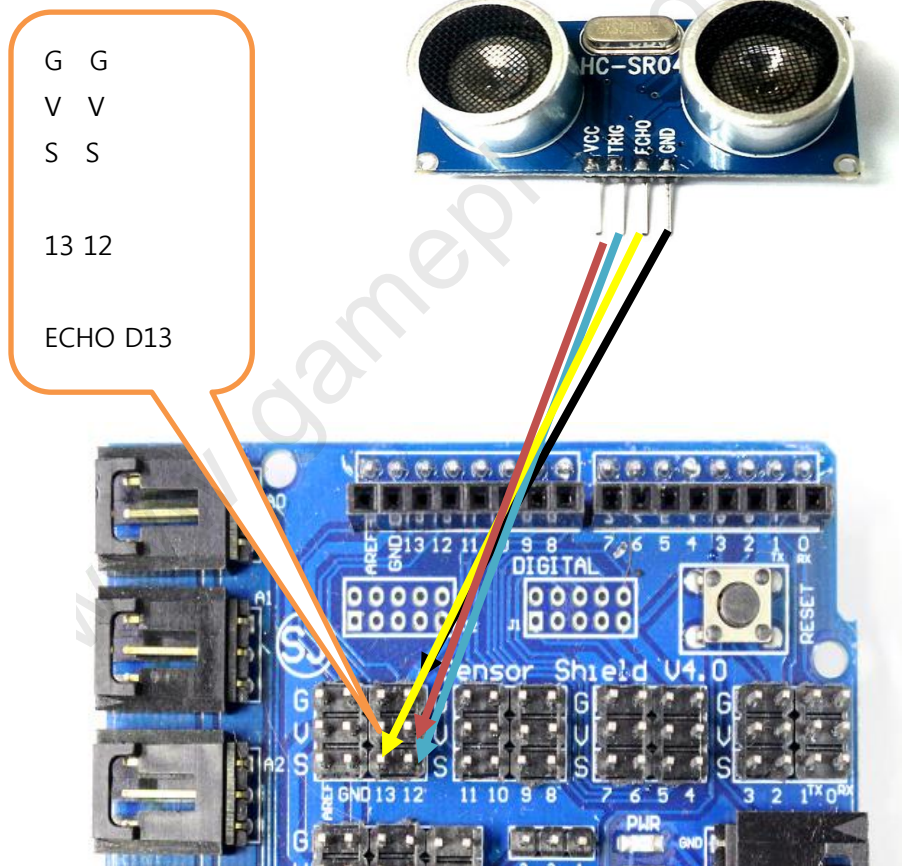
$$\text{Distance} = \text{time} / 29 / 2$$

HC-SR04 는 약 4 미터 정도의 거리까지 측정 가능합니다.

9.2 초음파 센서 연결하기

키트에 포함된 뒤풍 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.

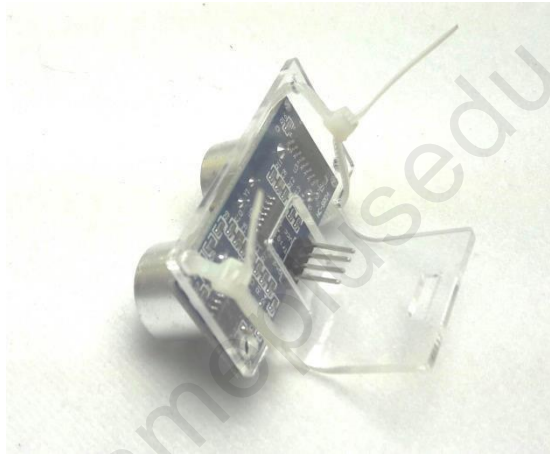


초음파 센서 연결

9.3 초음파 센서를 이용한 거리 측정 구현

앞에서 설명한 방식으로 초음파 센서를 이용하여 장애물까지 거리를 측정해 봅니다. 초음파 센서 모듈의 Trig, Echo 핀에 대한 제어를 코드에서 처리합니다.

케이블 타이를 이용하여 센서 거치대에 초음파 센서(HC-SR04)를 고정시키고 스마트 자동차 상단 앞쪽에 장착한 다음 센서실드에 듀폰 케이블로 연결합니다.



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();
    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.

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

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.






    return microseconds / 29 / 2; // 시간에 대한 값을 센티미터로 변환
}

```

10 장애물 회피 자율주행

스마트 자동차의 자율주행은 초음파 거리센서, 전자나침판을 활용하여 장애물을 회피하면서 정해진 방향으로 스스로 주행하는 기능입니다. 센서에서 수집된 자료를 활용하고 정밀한 프로그램 처리를 통해 자율주행 자동차의 원리를 익혀 봅니다.

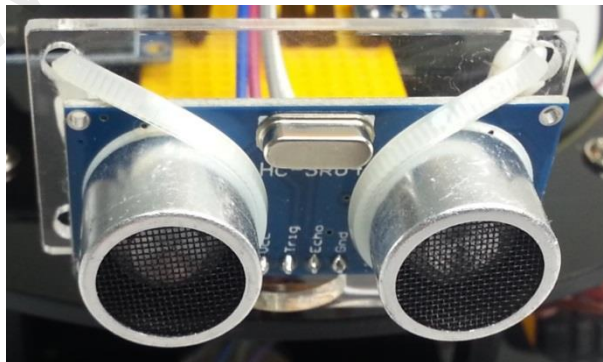
- 장애물 감지를 위한 초음파 센서 3 개 장착
- 주행방향 설정을 위한 전자나침판 장착
- 주행방향 표시 OLED 디스플레이 장착

HC-SR04	센서 거치대	케이블타이
		
OLED 0.96" I2C 4PIN	GY 282	
		

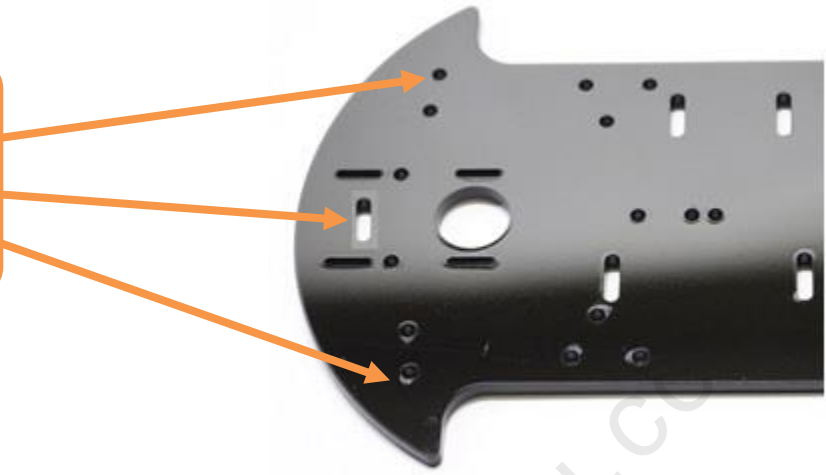
10.1 장애물 감지 초음파 센서 장착

초음파 센서를 활용하여 장애물을 감지합니다. 전면, 좌우 3 개의 센서를 장착하고 근접한 장애물을 감지하는데 HC-SR04 는 훌륭한 기능을 할 수 있습니다. 다음을 참고하여 스마트 자동차에 센서를 장착합니다.

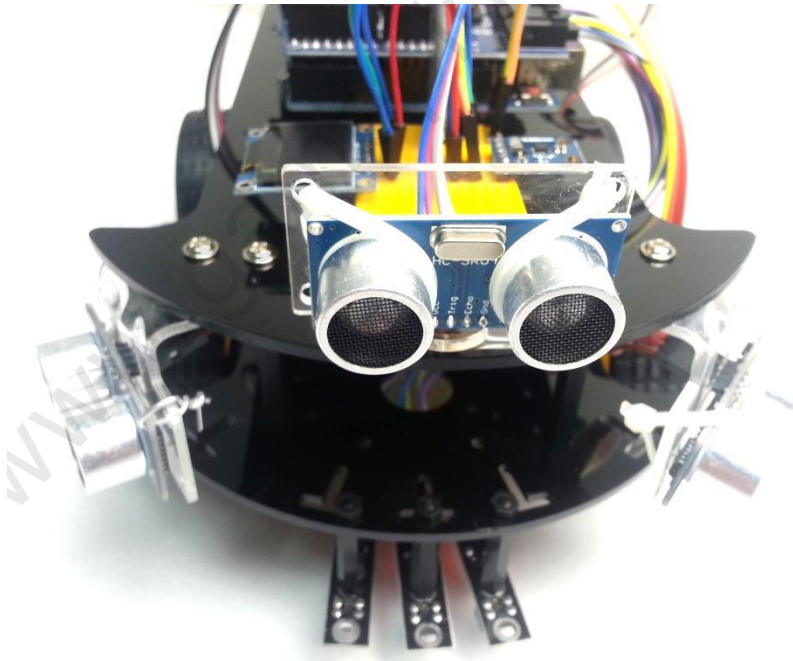
- 초음파 센서 핀 펴기
- 아크릴 보호막 투명비닐 제거
- 케이블타이로 초음파 센서를 거치대에 고정
- 3 개의 초음파 센서(HC-SR04)를 좌/우/중앙에 장착
- 좌-T(14)/E(15), 우-T(8)/E(9), 중앙-T(12)/E(13) 배선 연결

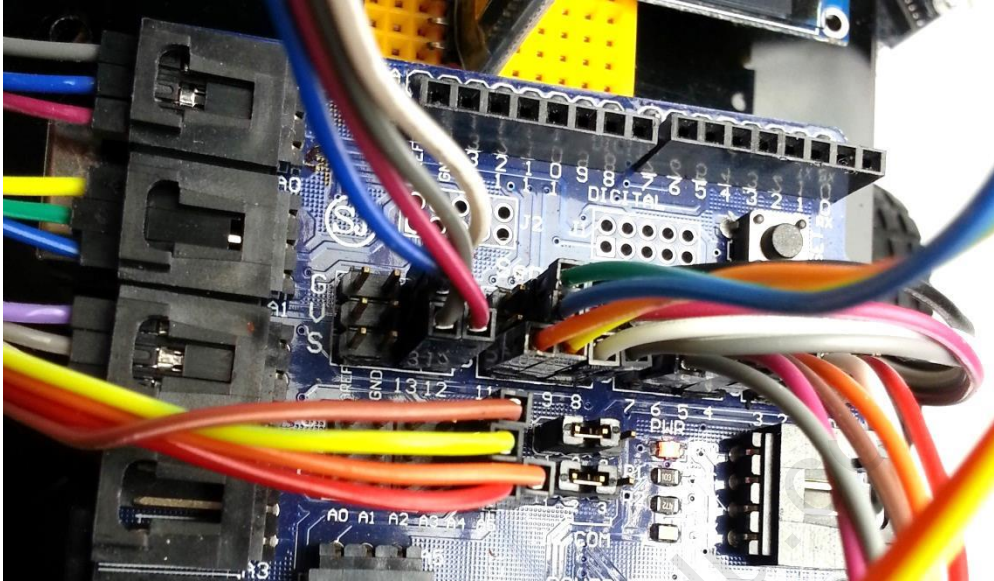


오른쪽
중앙
왼쪽



중앙은 센서 거치대를 세우고 좌/우는 매달아서 볼트나사로 고정





아두이노 아날로그핀은 디지털핀으로도 사용할 수 있습니다.

- A0 = D14, , A5 = D19

초음파센서를 연결할 디지털 핀이 부족하므로 좌측 초음파센서는 아날로그 핀 A0(D14), A1(D15)을 사용합니다.

초음파(좌)	센서실드	초음파(우)	센서실드	초음파(앞)	센서실드
VCC	VCC	VCC	VCC	VCC	VCC
Trig	A0(D14)	Trig	D8	Trig	D12
Echo	A1(D15)	Echo	D9	Echo	D13
GND	GND	GND	GND	GND	GND

10.2 방향 탐지를 위한 전자나침반 모듈 장착

특정 방향을 목표지점으로 자율주행에 적용하기 위해 전자나침반을 스마트 자동차에 추가해 봅니다. 본 키트에서 전자나침반은 HMC5983 칩을 장착한 GY 282 모듈을 사용합니다.



HMC5983 은 지구자기장을 측정하는 센서로 실제 북쪽방향과 약간 차이가 있으므로 측정된 값을 지역에 따라 보정하여 사용합니다. 또한 자석이나 금속물질에 영향을 받으므로 이점을 고려해야 합니다.

■ 주의 ■

HMC5983 모듈의 I2C 주소가 0x1E 입니다. HMC5883L 정품모듈과 호환되나 일부 다른 칩을 사용하는 모듈은 I2C 주소가 달라 예제 소스를 사용할 수 없으니 i2c 주소 스캔 코드로 꼭 확인하기 바랍니다.

전자나침반 모듈과 센서실드 연결은 아래 표를 참고하여 구성합니다.

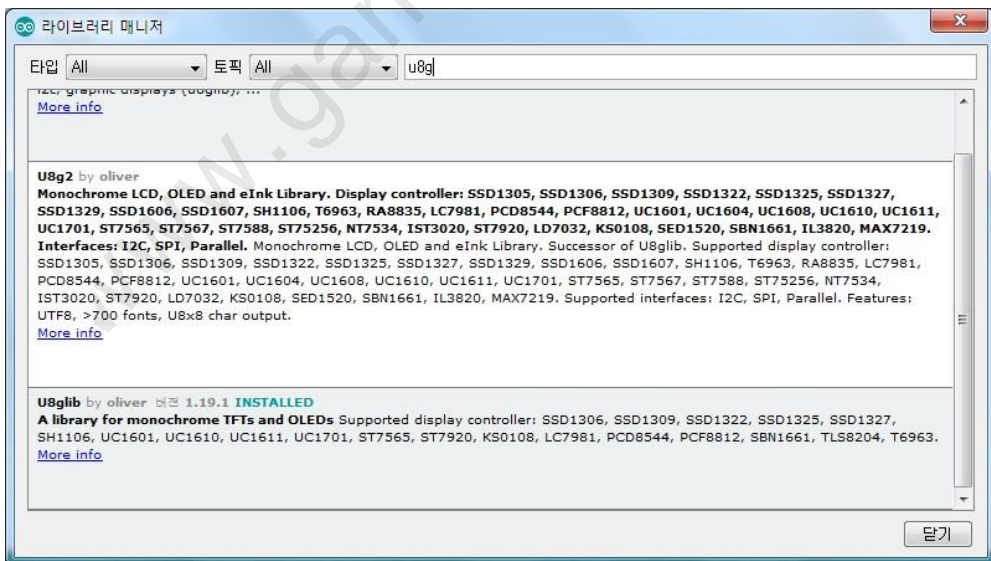
GY-282	센서 실드
VCC	VCC
GDN	GND
SCL	A5(SCL)
SDA	A4(SDA)
DROY	

전자나침반의 방향 각도를 디스플레이로 표시하기 위해 OLED 모듈을 추가합니다.. I2C 인터페이스로 연결되는 디스플레이에 초기설정된 목표방향과 이동 중 방향을 표시하여 사용자에게 상황정보를 제공하도록 예제 코드가 준비되어 있습니다.



0.96 Inch 128X64 Blue OLED Display

위 OLED 디스플레이는 U8glib 라이브러리를 설치해야 정상적으로 작동합니다. 아두이노 IDE 프로그램 메뉴에서 [스케치] > [라이브러리 포함하기] > [라이브러리 관리]를 선택합니다. 라이브러리 매니저에서 U82lib 를 검색하여 설치합니다.



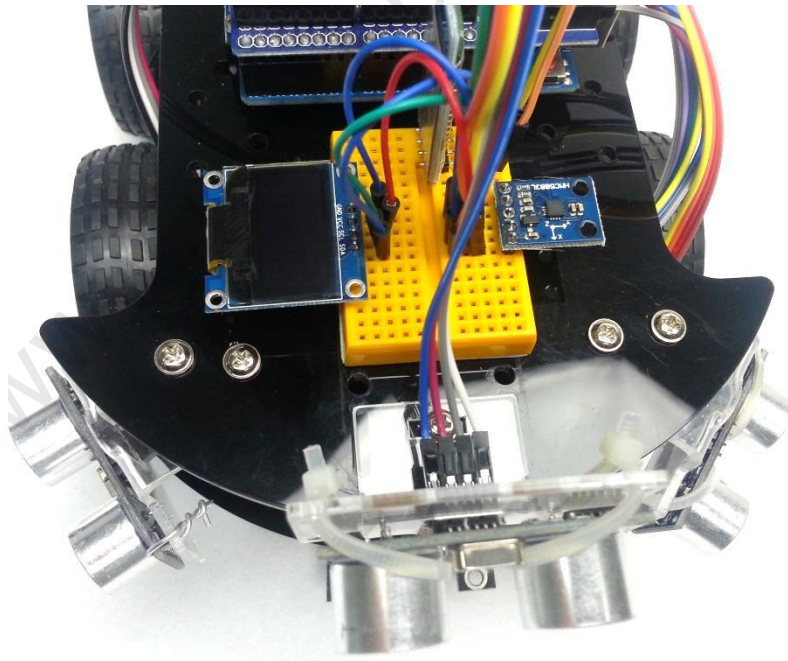
OLED 디스플레이와 센서실드 연결은 아래 표를 참고하여 구성합니다.

OLED 디스플레이	센서 실드
GND	GND
VCC	VCC
SCL	A5(SCL)
SDA	A4(SDA)

OLED 디스플레이는 라이브러리에서 제공하는 예제 코드를 이용하여 정상 작동 여부를 확인할 수 있습니다.

- [파일] > [예제] > [U8glib] > [GraphicTest]

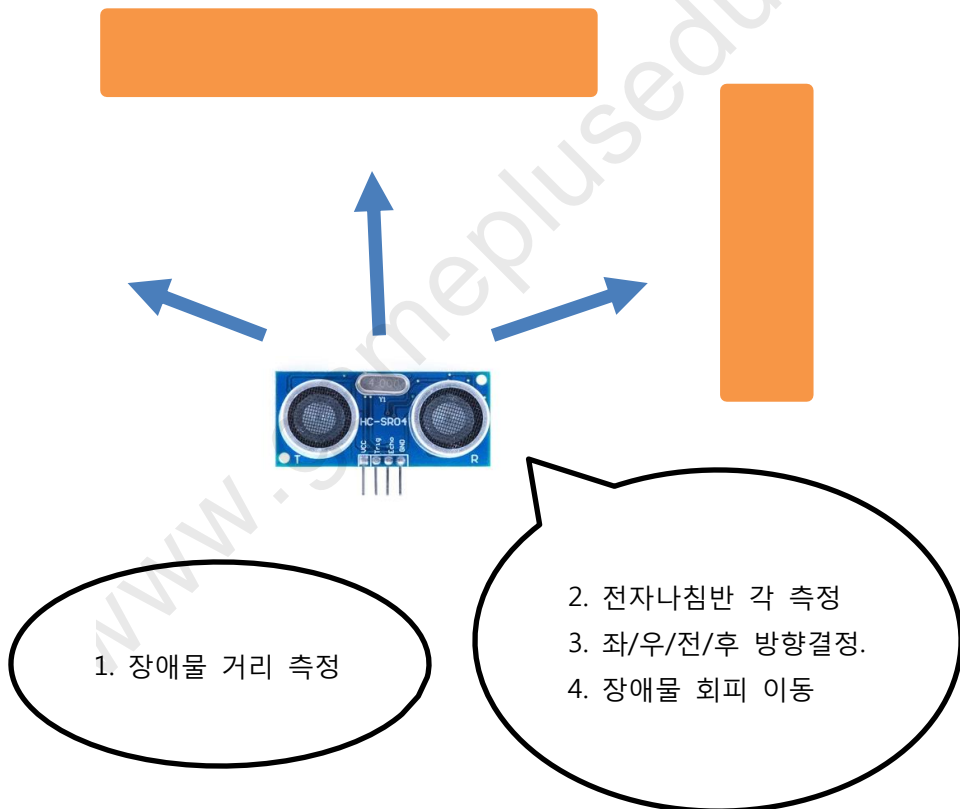
본 키트에서는 미니브레드보드를 정면 초음파센서와 아두이노 보드 사이에 장착하여 그림과 같이 구성하였습니다.



10.3 장애물 감지 방향 전환

스마트 자동차는 3 개의 초음파센서를 이용하여 좌/우/앞 위치의 장애물을 감지 하면서 설정된 방향을 향해 이동합니다. 다음의 여러 요소에 따라 자율주행 성능을 체크하고 소스코드를 업데이트 하여 환경에 맞게 개선해 보세요.

- 모터속도(0~255)
- 장애물 감지거리(0~2 미터)
- 장애물 회피 회전 각도
- 현재방향과 목표방향



10.4 장애물 회피 자율주행 구현

장애물 회피 자율주행의 모든 기능을 구현한 코드는 여러 개 파일로 구성됩니다. 아래 메인 소스 파일을 아두이노 IDE 에서 열어서 컴파일하고 업로드 하세요.

예제 소스폴더에는 전자나침반 기능에 필요한 소스파일을 포함하고 있습니다. HMC5883L 관련 소스는 방향 각도를 구하기위해 라이브러리처럼 사용합니다. 별도의 설명은 지원하지 않으니 참고만하기 바랍니다.

```
/******  
 * Smart Car V3  
 * - Auto driving  
 * - Object Avoidance(장애물 회피)  
 * - HC-SR04 : 초음파센서 3 개  
 * - HMC5883L : 자기장 센서  
 * - Note : 모터속도, 장애물거리, 회전각도 등을 환경에 맞게 설정  
*****/  
  
////////////////////////////////////  
// Ultrasonic sensor (front/left/right)  
////////////////////////////////////  
#define Sonic_Front_TRIG 12 // front Trigger  
#define Sonic_Front_ECHO 13 // front Echo  
#define Sonic_Right_TRIG 8 // right Trigger  
#define Sonic_Right_ECHO 9 // right Echo  
#define Sonic_Left_TRIG 14 // left Trigger  
#define Sonic_Left_ECHO 15 // left Echo  
  
#define MAX_DISTANCE_SONIC 200  
  
// HC-SR04 Ultrasonic sensor(0~200cm)  
int getDistance_UltraSonic(int pin_Trig, int pin_Echo)  
{
```

```

int centimeter;
long duration;

digitalWrite(pin_Trig, HIGH);
delayMicroseconds(10);
digitalWrite(pin_Trig, LOW);
delayMicroseconds(10);
duration = pulseIn(pin_Echo, HIGH);
centimeter = (int)(duration/29/2);    // microsecondsToCentimeters

if (centimeter > MAX_DISTANCE_SONIC)
    centimeter = MAX_DISTANCE_SONIC;

return centimeter;
}

/////////////////////////////////////////////////////////////////
// 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

/////////////////////////////////////////////////////////////////
// Car direction
//
#define CAR_DIR_FORWARD  0  // forward
#define CAR_DIR_BACKWARD 1  // backward
#define CAR_DIR_LEFT     2  // left turn
#define CAR_DIR_RIGHT    3  // right turn
#define CAR_DIR_STOP     4  // stop

/////////////////////////////////////////////////////////////////

```

```

// Car Speed : 0 ~ 255
// it depends on the moter and battery power
// Obstacle distance may be linked to car speed
////////////////////////////////////
#define CAR_SPEED_DEFAULT 100 //120
#define CAR_SPEED_FAST 150
#define CAR_SPEED_SLOW 50

////////////////////////////////////
// Default direction and speed
//
int g_carDirection = CAR_DIR_STOP;
int g_carSpeed_L;
int g_carSpeed_R;

////////////////////////////////////
// Note : confirm HIGH/LOW for correct movement
////////////////////////////////////
void car_forward()
{
    digitalWrite(EN1, HIGH);
    digitalWrite(EN2, LOW);
    analogWrite(ENA, g_carSpeed_R);

    digitalWrite(EN3, HIGH);
    digitalWrite(EN4, LOW);
    analogWrite(ENB, g_carSpeed_L);
    Serial.println("*****car_forward**");
}

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

    digitalWrite(EN3, LOW);

```

```

digitalWrite(EN4, HIGH);
analogWrite(ENB, g_carSpeed_L);
Serial.println("*****car_back**");
}

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

digitalWrite(EN3, HIGH);
digitalWrite(EN4, LOW);
analogWrite(ENB, g_carSpeed_L);
Serial.println("*****car_left**");
}

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

digitalWrite(EN3, LOW);
digitalWrite(EN4, HIGH);
analogWrite(ENB, g_carSpeed_L);
Serial.println("*****car_right**");
}

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

////////////////////
// Execute car moving

```



```

////////////////////////////////////
void update_Car()
{
    switch ( g_carDirection ) {
        case CAR_DIR_FORWARD:
            car_forward();
            break;
        case CAR_DIR_BACKWARD:
            car_backward();
            break;
        case CAR_DIR_LEFT:
            car_left();
            break;
        case CAR_DIR_RIGHT:
            car_right();
            break;
        case CAR_DIR_STOP:
            car_stop();
            break;
        default :
            ;
    }
    return;
}

// option for obstacles
bool isLeftObstacle = false;
bool isRightObstacle = false;
bool isFrontObstacle = false;
bool isAlmostBump = false;

// option for the previous actions
bool isBack = false;    // check the prev BACK operation
int countLeftRight = 0; // left(-) right(+)

// distance of obstacles
int cmLeftObstacle = 0;

```

```

int cmRightObstacle = 0;
int cmFrontObstacle = 0;

// Obstacle distance
#define DISTANCE_OBSTACLE_FRONT    (15+CAR_SPEED_DEFAULT/10)
#define DISTANCE_OBSTACLE_LEFT    (CAR_SPEED_DEFAULT/10)
#define DISTANCE_OBSTACLE_RIGHT   (CAR_SPEED_DEFAULT/10)
#define DISTANCE_ALMOST_BUMP_FRONT (CAR_SPEED_DEFAULT/10)

void searchObstacles()
{
    // reset obstacle options
    isFrontObstacle = isLeftObstacle = isRightObstacle = isAlmostBump = false;

    // HC-SR04 : front
    cmFrontObstacle = getDistance_UltraSonic(Sonic_Front_TRIG, Sonic_Front_ECHO);
    if (cmFrontObstacle < DISTANCE_OBSTACLE_FRONT) {
        isFrontObstacle = true;
        if (cmFrontObstacle < DISTANCE_ALMOST_BUMP_FRONT)
            isAlmostBump = true; // must go back at once
    }

    if (isBack== true) { // check backward moving
        isFrontObstacle = true;
        isBack = false; // reset by default
    }

    // HC-SR04 : right/left
    cmRightObstacle = getDistance_UltraSonic(Sonic_Right_TRIG, Sonic_Right_ECHO); //
right sensor
    if (cmRightObstacle < DISTANCE_OBSTACLE_RIGHT)
        isRightObstacle = true;

    cmLeftObstacle = getDistance_UltraSonic(Sonic_Left_TRIG, Sonic_Left_ECHO); // left
sensor
    if (cmLeftObstacle < DISTANCE_OBSTACLE_LEFT)

```

```

    isLeftObstacle = true;

    Serial.println("*** SearchObstacles() ***");
    Serial.println(" Left-----Front-----Right");
    Serial.print(cmLeftObstacle);
    Serial.print(" ----- ");
    Serial.print(cmFrontObstacle);
    Serial.print(" ----- ");
    Serial.println(cmRightObstacle);

    return;
}

////////////////////////////////////
// compass setting
// targetDegree - target direction to move
// Note : setup() 에서 현재 방향을 목표방향으로 설정함
////////////////////////////////////
int targetDegree = 0;    // 0 - north (0 ~ 360)
int currentDegree = 0;

void setup() {

    // for debug
    Serial.begin(9600);
    delay(100);
    Serial.println("Auto Driving(Obstacle avoidance) >> Start");

    setupHMC5883L(); // initialize compass sensor(HMC5883L)

    pinMode(ENA, OUTPUT);
    pinMode(ENB, OUTPUT);
    pinMode(EN1, OUTPUT);
    pinMode(EN2, OUTPUT);
    pinMode(EN3, OUTPUT);
    pinMode(EN4, OUTPUT);

```

```

pinMode(Sonic_Front_TRIG,OUTPUT);
pinMode(Sonic_Front_ECHO,INPUT);
pinMode(Sonic_Right_TRIG,OUTPUT);
pinMode(Sonic_Right_ECHO,INPUT);
pinMode(Sonic_Left_TRIG,OUTPUT);
pinMode(Sonic_Left_ECHO,INPUT);

// boot notification on OLED
drawMessage("Setup !!");
delay(2000);
// set target direction at first
targetDegree = getHeadingDegree();
drawLED("Target : ", targetDegree);
delay(3000);
}

////////////////////////////////////
// 장애물과 모터 성능에따라 delay time 으로 속도 조절
////////////////////////////////////
#define DELAY_SHORT 100
#define DELAY_NORMAL 200
#define DELAY_LONG 300

int countForward = 0; // counting continuous goForward() for direction checking

void loop() {

searchObstacles();

currentDegree = getHeadingDegree();

if (isFrontObstacle == false) { // no front obstacle
if (isLeftObstacle == true && isRightObstacle == true) {
Serial.println("Left-----<.....>-----Right -> goBack");
goBackward();
isBack = true;
delay(DELAY_SHORT);
}
}
}

```

```

}
else if (isLeftObstacle == true) { // left obstacle (no front)
    Serial.println("Left-----<.....>-----<.....> -> turnRight");
    turnRight();
    delay(DELAY_SHORT);
    countLeftRight++;
    goForward();
    delay(DELAY_NORMAL);
}
else if (isRightObstacle == true) { // right obstacle (no front)
    Serial.println("<.....>-----<.....>-----Right -> turnLeft");
    turnLeft();
    delay(DELAY_SHORT);
    countLeftRight--;
    goForward();
    delay(DELAY_NORMAL);
}
else { // no front/left/right obstacles
    //////////////////////////////////////
    // check target direction
    //////////////////////////////////////
    if ((countForward > 4)) { // 4 : count limit based on obstacle condition
        // rotate for target direction
        if (currentDegree > (targetDegree+30)) {
            turnLeft();
            delay(DELAY_SHORT);
        } else if (currentDegree < (targetDegree-30)) {
            turnRight();
            delay(DELAY_SHORT);
        }
        else {}

        countForward = 0; // initialize
    } //
    else {
        countForward++;
    }
}

```

```

Serial.println("<.....>-----<.....>-----<.....> -> goForward");
goForward();
delay(DELAY_NORMAL);
}
}
else { // front obstacle : +-----+ +-----+ +-----+ +-----+
// LR obstacle : |<left |<both>| right>| nothing
countForward = 0;

if (isAlmostBump==true || (isLeftObstacle==true && isRightObstacle==true)) {
Serial.println("Left-----Front-----Right -> goBackward");
goBackward();
delay(DELAY_NORMAL);
isBack = true;
}
else if (isLeftObstacle == true) { // front/left obstacle
Serial.println("Left-----Front-----[.....] -> turnRight");
turnRight();
delay(DELAY_NORMAL);
countLeftRight++;
goForward();
delay(DELAY_NORMAL);
}
else if (isRightObstacle == true) { // front/right obstacle
Serial.println("[.....]-----Front-----Right -> turnLeft");
turnLeft();
delay(DELAY_NORMAL);
countLeftRight--;
goForward();
delay(DELAY_NORMAL);
}
else { // front obstacle (no left/right)
//////////
// check target direction
//////////
if (currentDegree > targetDegree) {

```

```

Serial.println("[.....]-----Front-----[.....] -> turnLeft");
turnLeft();
delay(DELAY_NORMAL);
countLeftRight--;
goForward();
delay(DELAY_NORMAL);
}
else { // if(currentDegree < targetDegree) {
Serial.println("[.....]-----Front-----[.....] -> turnRight");
turnRight();
delay(DELAY_NORMAL);
countLeftRight++;
goForward();
delay(DELAY_NORMAL);
}
} // no left/right obstacle
} // front obstacle

////////////////////////////////////
// debug : possible to check out direction with delay()
////////////////////////////////////
//delay(500);
}

void goForward()
{
g_carDirection = CAR_DIR_FORWARD;

g_carSpeed_L = CAR_SPEED_DEFAULT;
g_carSpeed_R = CAR_SPEED_DEFAULT;

update_Car();
delay(100);
}

void goBackward()
{

```

```
g_carDirection = CAR_DIR_BACKWARD;

g_carSpeed_L = CAR_SPEED_DEFAULT;
g_carSpeed_R = CAR_SPEED_DEFAULT;

update_Car();
delay(500);
}

void turnLeft()
{
g_carDirection = CAR_DIR_LEFT;

g_carSpeed_L = CAR_SPEED_DEFAULT;
g_carSpeed_R = CAR_SPEED_DEFAULT;

update_Car();
delay(500);
}

void turnRight()
{
g_carDirection = CAR_DIR_RIGHT;

g_carSpeed_L = CAR_SPEED_DEFAULT;
g_carSpeed_R = CAR_SPEED_DEFAULT;

update_Car();
delay(500);
}
```


11 아두이노 우노 R3

아두이노란?

아두이노는 오픈 소스를 기반으로 한 단일 보드 마이크로컨트롤러로 완성된 보드(상품)와 관련 개발 도구 및 환경을 말합니다. 2005년 이탈리아에서 하드웨어에 익숙지 않은 학생들이 자신들의 디자인 작품을 손쉽게 제어할 수 있도록 하기 위해 고안된 아두이노는 처음에 AVR을 기반으로 만들어졌으며, 아트멜 AVR 계열의 보드가 현재 가장 많이 판매되고 있습니다. ARM 계열의 Cortex-M0(Arduino M0 Pro)과 Cortex-M3(Arduino Due)를 이용한 제품도 존재합니다.

아두이노는 다수의 스위치나 센서로부터 값을 받아들여, LED나 모터와 같은 외부 전자 장치들을 통제함으로써 환경과 상호작용이 가능한 물건을 만들어 낼 수 있습니다. 임베디드 시스템 중의 하나로 쉽게 개발할 수 있는 환경을 이용하여, 장치를 제어할 수 있습니다.

아두이노 통합 개발 환경(IDE)을 제공하며, 소프트웨어 개발과 실행코드 업로드도 제공합니다. 또한 어도비 플래시, 프로세싱, Max/MSP와 같은 소프트웨어와 연동할 수 있습니다.

아두이노 우노

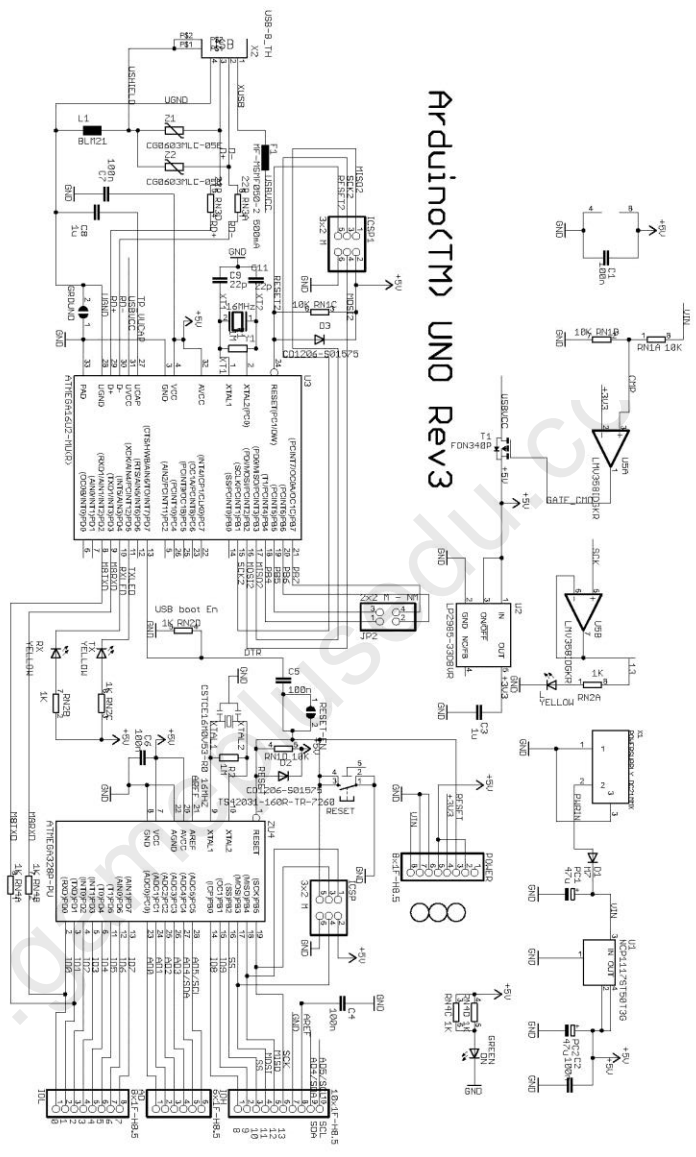
아두이노 우노는 ATmega328P에 기반한 마이크로컨트롤러 보드입니다. 보드에는 14개의 디지털 입력/출력 핀(이 중 6개는 PWM 출력으로 사용될 수 있음), 6개의 아날로그 입력, 16Mhz 석영 크리스탈, USB 연결, 파워 잭, ICSP 헤더와 리셋 버튼이 있습니다.

11.1 개요

아두이노 우노 R3는 ATmega328P에 기반한 마이크로컨트롤러 보드이며, 32KB 플래시 메모리 8비트 마이크로컨트롤러와 2KB 램으로 구성되어 있습니다. USB 케이블로 컴퓨터와 연결할 수도 있고, AC-to-DC 어댑터나 배터리를 이용하여 전원을 공급할 수도 있습니다.

11.2 기술 사양

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB
Flash Memory for Bootloader	0.5 KB
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz



Arduino™ UNO Rev3

Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS. Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information. ARDUINO is a registered trademark.

Use of the ARDUINO name must be compliant with <http://www.arduino.cc/en/Main/Policy>

11.3 아두이노 소프트웨어

오픈소스 아두이노 소프트웨어(IDE)는 코드 작성이 쉽고, 보드에 업로드하기도 용이합니다. Windows, Mac OS X, 그리고 Linux 에서 작동합니다. 환경은 자바로 써있고, 처리 소프트웨어와 다른 오픈소스 소프트웨어에 기반하고 있습니다. 또한 어떤 아두이노 보드와도 사용 가능합니다.

아두이노 소프트웨어는 코드 작성을 위한 텍스트 에디터, 메시지 영역, 텍스트 콘솔, 공통 기능과 메뉴 툴바로 이루어져 있습니다. 아두이노 하드웨어, Genuino 하드웨어와의 연결을 형성하고 프로그램을 업로드하고 통신이 가능하게 합니다.

스케치 작성

아두이노 소프트웨어를 이용하여 작성된 프로그램을 스케치(sketches)라고 부릅니다. 스케치는 텍스트 에디터에서 작성되고 .ino 라는 파일 확장자의 형태로 저장됩니다. 에디터에서는 자르기/붙여넣기 그리고 찾기/바꾸기가 가능합니다. 메시지 영역에서는 저장과 내보내기의 피드백이 가능하며 오류를 보여줍니다. 콘솔은 아두이노 소프트웨어가 출력한 텍스트(오류 메시지 등의 정보)를 보여줍니다. 우측 아래의 창에는 보드와 시리얼 포트가 표시됩니다. 툴바 버튼들을 통해 검증, 업로딩, 작성, 열기, 스케치 저장, 시리얼 모니터를 할 수 있습니다.

메뉴

[스케치(Sketch)]

- 확인/컴파일(Verify/Compile)
 - 컴파일 과정에서의 오류를 확인합니다. 그리고 코드와 변수들이 사용하는 메모리 크기를 콘솔 영역에 보여줍니다.
- 업로드(Upload)
 - 코드를 컴파일하고 포트를 이용하여 보드에 바이너리 파일을 업로드합니다.
- 프로그래머를 이용해 업로드(Upload Using Programmer)
 - 보드의 부트로더에 덮어씁니다.
- 컴파일된 바이너리 보내기(Export Compiled Binary)
 - 다른 툴을 이용하는 보드에 보내거나 아카이빙하기 위한 .hex 파일을 저장합니다.
- 스케치 폴더 보이기(Show Sketch Folder)

- 현재 스케치 폴더를 엽니다.
- 라이브러리 포함하기(Include Library)
 - 코드 상단에 #include 문을 삽입하여 현재 스케치에 라이브러리를 추가합니다.
- 파일 추가...(Add File...)
 - 스케치에 소스 파일을 추가합니다.

[툴(Tools)]

- 자동 포맷(Auto Format)
 - 코드를 깔끔하게 정리합니다.
- 스케치 보관하기(Archive Sketch)
 - .zip 형식으로 현재의 스케치를 아카이빙합니다. 아카이브는 현재 스케치와 같은 폴더에 생성됩니다.
- 인코딩 수정 & 새로고침(Fix Encoding & Reload)
 - 에디터 문자표(char map)와 운영체제 문자표 사이의 불일치들을 수정합니다.
- 시리얼 모니터(Serial Monitor)
 - 시리얼 모니터 창을 열고, 현재 선택된 포트에 연결된 보드와의 데이터 교환을 초기화합니다. 시리얼 포트 연결(opening)을 통한 리셋이 가능한 경우, 보드를 리셋(reset)합니다.
- 보드(Board)
 - 이용하고 있는 보드를 선택합니다.
- 포트(Port)
 - 포트를 선택합니다.
- 프로그래머(Programmer)
 - 온보드 USB 시리얼 연결을 이용하지 않고 보드나 칩을 프로그래밍할 때, 하드웨어 프로그래머를 선택하기 위해 사용합니다. 보통은 이용하지 않지만, 새로운 마이크로컨트롤러에 부트로더를 구웠을 때 이 메뉴를 사용합니다.
- 부트로더 굽기(Burn Bootloader)
 - 아두이노 보드 마이크로컨트롤러 부트로더를 구울 때 사용합니다. 보통은 필요하지 않지만, 새로운 ATmega 마이크로컨트롤러(부트로

더가 없는)를 구매했을 때 유용합니다. 부트로더를 굽기 전에 정확한 보드를 선택했는지를 확인하세요.

스케치북

아두이노 소프트웨어는 스케치들을 담아 놓는 곳인 스케치북이란 개념을 사용합니다. 스케치북의 스케치들은 [파일->스케치북]을 통해 열 수 있습니다. 처음 아두이노 소프트웨어를 실행했을 때는 자동으로 스케치북 폴더가 만들어집니다. 스케치북의 위치는 [파일->환경설정]에서 볼 수 있습니다.

시리얼 모니터

아두이노 보드가 보내오는 시리얼 데이터를 표시합니다. 보드로 데이터를 보내기 위해서는 텍스트를 입력하고 "전송" 버튼이나 엔터를 누르면 됩니다. 스케치의 Serial.begin 에 넘겨진 레이트와 일치하도록 보드 레이트를 선택할 수 있습니다. 윈도우, 맥, 리눅스에서는 시리얼 모니터를 통해 연결할 때 보드가 리셋됩니다.

12 스마트폰과 자동차 연동

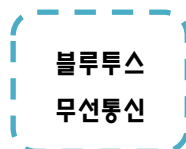
12.1 블루투스란

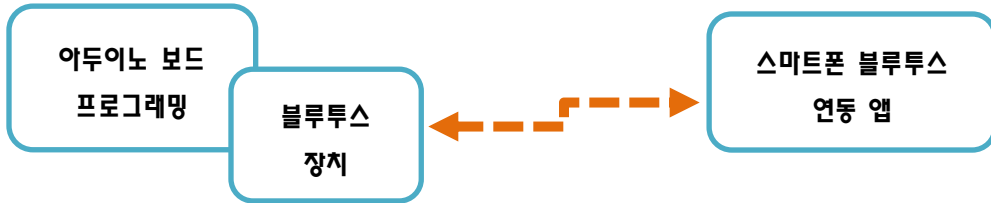
블루투스란 근거리 무선 통신 기술 규약을 의미합니다. 현재까지는 휴대용 기기, 근거리에서 사용되는 독립된 장치 등에 많이 사용됩니다. 블루투스의 출발은 휴대폰 제조사의 에릭슨, 노키아에서 개발된 만큼, 휴대용 기기에 많이 사용되고 있습니다. 블루투스의 출발은 소형 기기와의 통신을 위한 용도이므로 저전력을 사용하는 무선 통신 규약입니다. 스마트폰(휴대폰), 노트북, 태블릿 PC, 이어폰, 헤드폰, GPS 단말기, 키보드, 카메라, MP3 등에 사용됩니다.

블루투스의 사용 및 응용 범위는 계속 확대 추세이며, 현재의 블루투스는 4.0(~4.1)버전의 규약까지 나와있습니다. 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.



12.2 아두이노와 스마트폰의 블루투스 통신

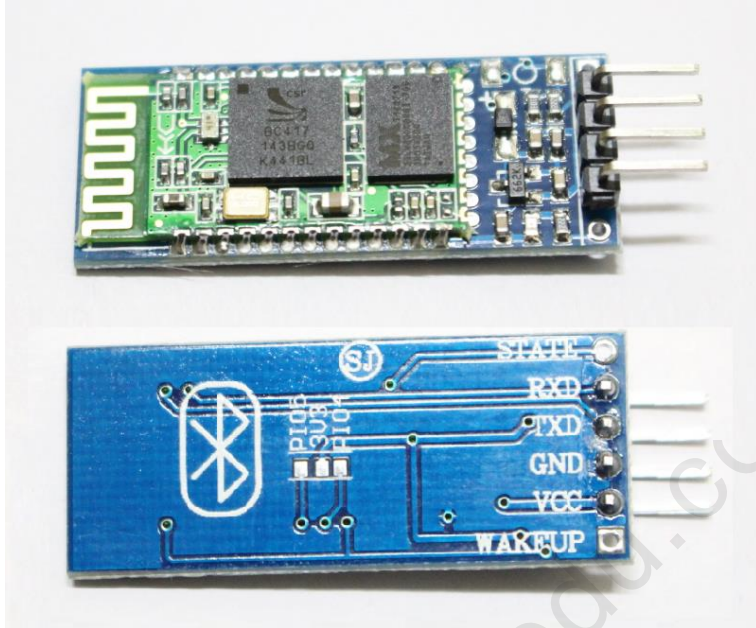




블루투스 통신은 위에서 언급된 12.1 설명대로 말 그대로 근거리 통신망에 사용되는 무선통신 규약입니다. 게다가 사용하기 편하게 시리얼 통신이 가능한 칩셋 구조로 생산되므로, 시리얼 통신 가능한 MCU, PC 등의 대부분의 장치에서 사용할 수 있습니다. 시리얼 통신은 최소 송/수신 2 개의 포트만 있으면 사용가능한 구조입니다. 아두이노 보드도 안정적인 시리얼 통신이 가능한 기기이므로 블루투스 모듈을 사용하면, 안정적인 근거리 무선 시리얼 통신이 가능합니다.

12.3 블루투스 HC-06 슬레이브 모듈

스마트 로봇 자동차의 근거리 통신 제어를 담당할 부품으로 블루투스 HC-06 슬레이브 모듈을 사용합니다. 아두이노 보드에 연결하여 사용가능한 블루투스 모듈은 HC-06 슬레이브 모듈과 HC-05 마스터 모듈이 있습니다. HC-05 블루투스 모듈은 마스터, 슬레이브 기능이 모두 지원됩니다.



블루투스 HC-06 슬레이브 모듈

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

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 으로 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 스위프트 다운 레벨 컨버터 저항 연결 구성을 할 필요는 없습니다.

12.4 아두이노와 블루투스 모듈 연결

HC-06 모듈(또는 HC-05 모듈)은 블루투스 무선 통신을 통해 시리얼 데이터를 주고 받을 수 있습니다. 블루투스 마스터/슬레이브 1:1 페어링이 된 상태인 경우 기기와의 통신 데이터는 모두 내부 시리얼 포트에 통신 가능하도록 구성되어 있습니다. 즉, HC-06(또는 HC-05) 모듈을 사용하기 위해서는 상대방 기기에서도 시리얼 포트 통신이 가능해야 합니다.

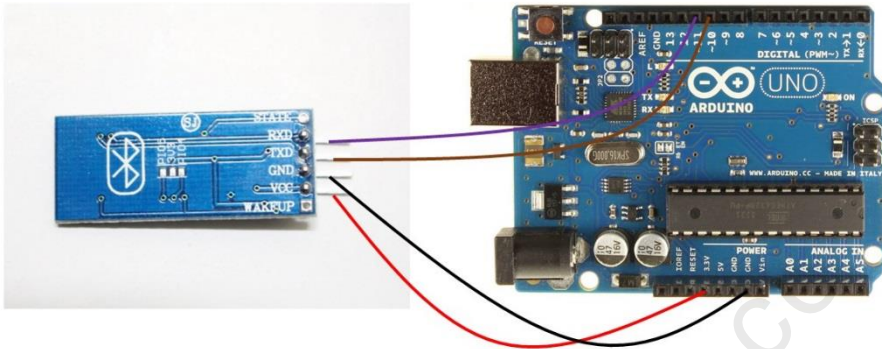
12.4.1 아두이노 하드웨어 시리얼 포트

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. ATmega328P MCU 에서는 하드웨어 시리얼 통신 포트가 1 개만 지원됩니다.

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

블루투스 HC-06 모듈에 sms RX (수신 포트), TX(송신 포트)가 있습니다. D0, D1 연결하여 사용할 수 있습니다. 블루투스 모듈의 RX/TX, D0, D1 연결일 때는 아두이노 IDE 스케치에서의 프로그램 업로드가 안됩니다. 펌웨어 업로드 후 단독으

로 아두이노 사용할 경우에는 블루투스 모듈을 D0, D1 연결하여 사용할 수 있지만 다소 사용하기 불편합니다.



HC-06 블루투스 모듈	아두이노 우노 R3
VCC	POWER 3.3V
GND	POWER GND
TXD	DIGITAL 10
RXD	DIGITAL 11

12.4.2 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) D10, 송신(TX) D11 포트를 사용하는 경우에는 블루투스 모듈의 TX, RX 포트를 연결하면 됩니다.

12.4.3 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 달라집니다. 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 변경되는 동안에는 소프트웨어 시리얼 통신이 불가 합니다.

최근의 아두이노 소프트웨어 시리얼 2 라이브러리를 사용하는 경우에는 하드웨어 시리얼 포트처럼 다른 포트에 영향을 받지 않고 사용가능 하다고 합니다. 필요한 경우에는 테스트해 보시기 바랍니다.

12.5 블루투스 시리얼 데이터 연동 기초

블루투스로 데이터를 주고 받기 위해서는 블루투스 기기(모듈) 페어링(Pairing)되어 있어야 합니다. 페어링이란 용어 그대로 짝짓기입니다. 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 있는 기능입니다. 물론 반대로 슬레이브 기능도 가지고 있습니다. 즉, 다른 기기에서의 페어링 요청을 받아들여 사용될 수도 있습니다.

블루투스 마스터 모듈의 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 실행과 리모트 컨트롤 코드를 구현한 상태인 경우 거의 비슷하게 수정하여 변경할 수 있습니다.

블루투스 통신에 의한 실행 제어도 쉽게 적용시킬 수 있습니다. 시리얼 통신 기반으로 명령어 전송 및 처리 부분을 구현해 주면 됩니다.

www.gameplusedu.cc

감사합니다

<http://www.gameplusedu.com>