3장 직렬 통신

3.1 직렬 통신 개요

직렬 통신(serial communication)은 기기들 사이에 데이터를 주고받는 방법 중 하나인데 병렬 통신 (parallel communication) 방식에 비해서 사용되는 통신선의 개수가 적다는 장점이 있어서 널리 사용되고 있다. 아두이노는 UART (universal asynchronous receiver and transmitter), SPI(Serial Peripheral Interface), I2C (Inter-Integrated Circuit 또는 TWI - Two Wire Interface - 라고도 불린다) 방식의 직렬 통신을 지원하는데 이중 UART는 주로 아두이노와 PC간의 통신에 사용된다.

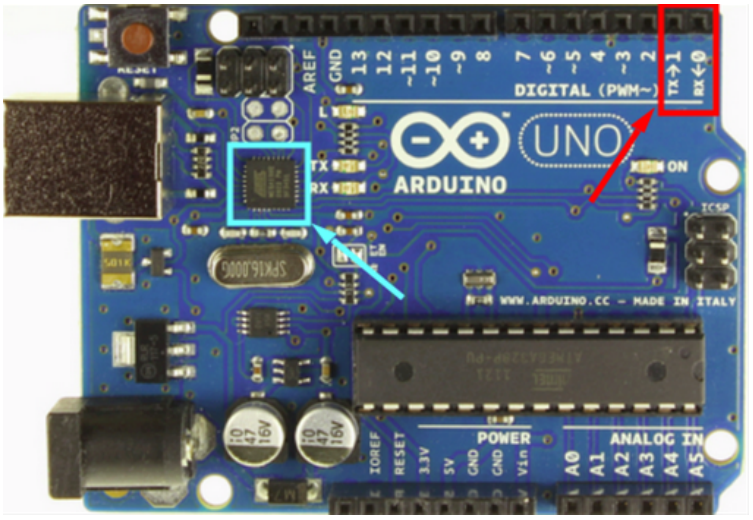
[표 3.1.1] 아두이노가 지원하는 직렬 통신 방식

통신 방식	사용	아두이노 라이브러리	신호선	비고
UART	사용자 프로그램 업로드 PC 와의 통신	Serial 클래스	TX, RX	1:1 통신
SPI	주변 기기와의 통신	SPI.h	MISO, MOSI, SCK, SS	1:n 통신
I2C	주변 기기와의 통신	Wire.h	SDL,SCL	1:n 통신

아두이노는 하나 이상의 UART 통신기가 내장되어 있으며 프로그램을 업로드할 때 바로 이 UART 통신을 이용한다. 또한 아두이노 Serial 클래스와 아두이노 IDE에 내장된 터미널(terminal, 주고 받는 데이터를 확인할 수 있는 프로그램)을 이용하면 손쉽게 PC와의 통신을 수행할 수 있다.

아두이노 우노의 경우 0번과 1번핀이 UART 통신에 사용된다. 이 핀들은 내부적으로 USB통신을 담당하는 칩과 연결되어서 USB신호로 변환된 후 PC에 전달된다. 반대로 PC에서 보내지는 USB신호는 이 칩에서 시리얼 통신 신호로 변환되어 아두이노의 AVR에 전달된다. 따라서 만약 아두이노가 PC와의 통신을 수행하고 있다면 이 핀들을 다른 용도로 사용하면 안된다. 그리고 통신을 수행할 때에는 TX, RX라고 마킹된 LED가 깜빡인다.

일단 사용자는 아두이노와 PC간에 USB케이블로 연결하면 UART 통신 실습을 할 준비가 끝나게 된다.



[그림 3.1.1] UART 통신에 사용되는 두 개의 핀(빨간색)과 usb변환 칩(하늘색).

3.2 첫 번째 예제

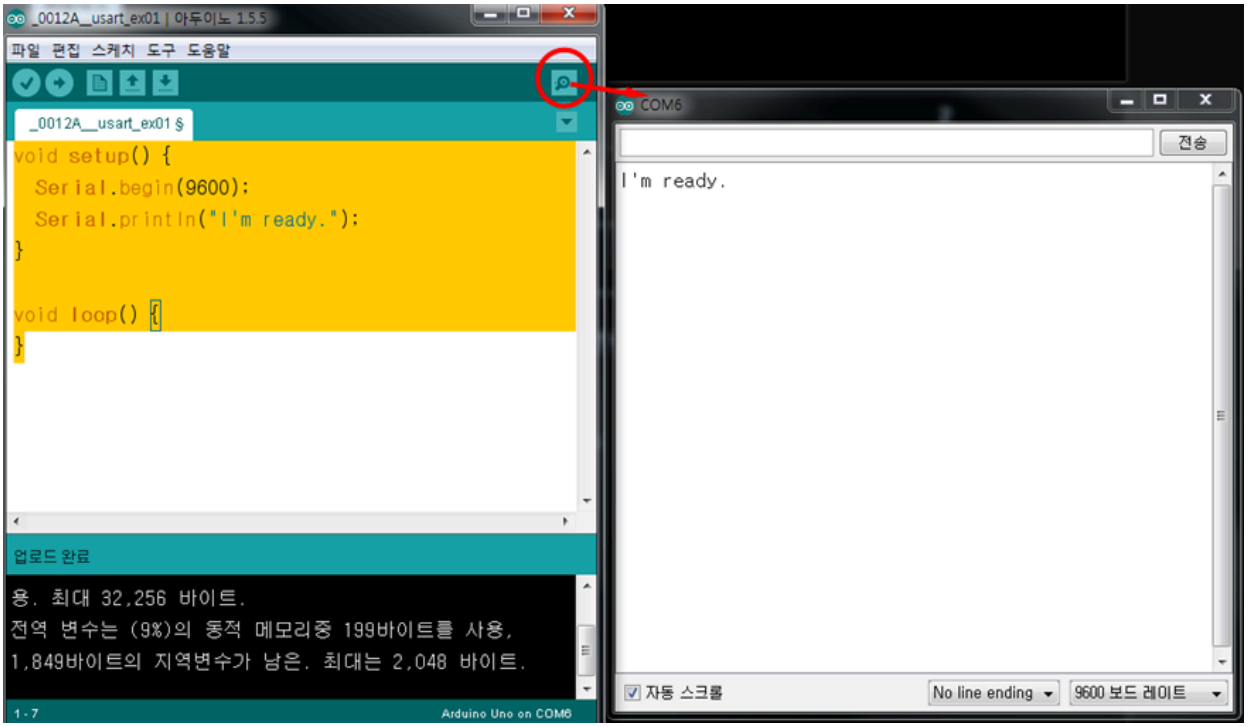
첫 번째 예제로 아두이노에서 PC로 간단한 데이터를 전송하는 예를 해보도록 하겠다. 단순히 아두이노에게 전원을 인가하면 “I’m ready.” 라는 메세지를 PC에 전송하는 예이다.

```
void setup() {
    Serial.begin(9600);
    Serial.print("I'm ready.");
}
void loop() {
}
```

이 예제를 다운로드 한 후 터미널을 켜고 리셋버튼을 누르면 터미널에 “I’m ready.”라는 문자열이 찍히는 것을 확인할 수 있다. 터미널 창에 대해서 다음에 주의한다.

- 터미널 실행 버튼을 누르면 아두이노에 자동으로 리셋신호가 걸려서 프로그램이 처음부터 수행된다.
- 프로그램 업로드 도중에는 터미널을 실행시키지 못 한다.
- 만약 터미털 창이 열린 상태에서 업로드를 수행하면 창이 닫힌다.

그 이유는 프로그램을 업로드하는 데에도 바로 이 UART 통신이 사용되기 때문이다.



[그림 3.2.1] 예제 수행 결과

UART와 관련된 아두이노의 라이브러리는 **Serial**클래스에 다 모여 있다. 일단 이 예제에서 보면 setup()함수내에서 두 개의 함수가 호출되었다.

- void **Serial.begin**(long baud_rate) 함수
  - USART 통신을 초기화 시킨다.
  - 통신 속도(baud rate, 초당 전송되는 비트)를 입력으로 받는다.
  - 9600, 19200, 57600, 115200 등 여러 baud rate를 지원한다.
  - 반드시 터미널의 통신속도는 설정한 속도와 같게 맞추어 주어야 한다.
- long **Serial.print**(val) 함수
  - 입력값을 ASCII값으로 변환하여 PC쪽으로 출력한다.
  - 전송된 데이터의 바이트 수를 리턴한다. (잘 사용되지 않음)
  - 비동기 통신 방식이므로 데이터가 전송되기 전에 리턴된다.
  - 입력 변수 val은 여러 데이터 타입이 가능하다. 예를 들면

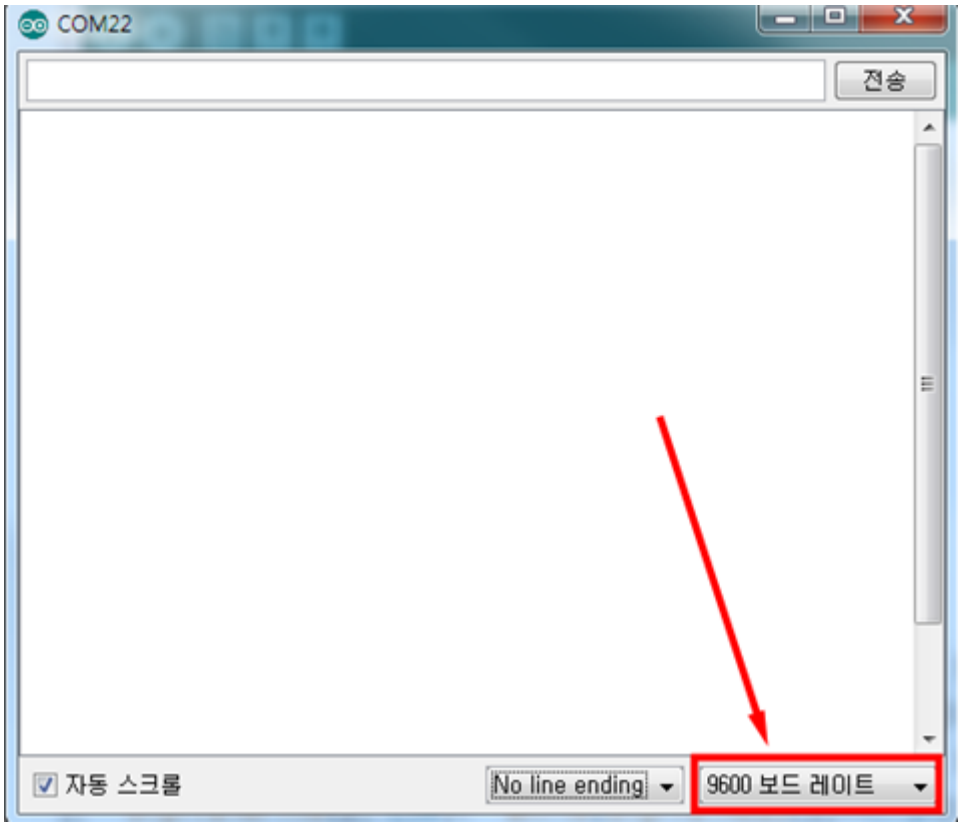
```
Serial.print(78); //gives "78"
Serial.print(1.23456); //gives "1.23"
Serial.print('N'); //gives "N"
Serial.print("Hello world."); //gives "Hello world."
```

- 두 번째 인수로 출력 형식을 지정할 수 있다. 예를 들면

```
Serial.print(78, BIN); // 이진수로 전송 : "1001110"
Serial.print(78, OCT); // 팔진수로 전송: "116"
Serial.print(78, DEC); // 10진수로 전송(기본값) : "78"
Serial.print(78, HEX); // 16진수로 전송 : "4E"
Serial.print(1.23456, 0); // 소수점 0번째 자리 까지 전송 : "1"
Serial.print(1.23456, 2); // 소수점 두 번째 자리 까지 전송 : "1.23"
Serial.print(1.23456, 4); // 소수점 네 번째 자리 까지 전송 : "1.2346”
```

첫 번째 예제는 setup() 함수 내에서 UART를 초기화 하고 문자열 하나를 보내는 아주 간단한 예제이다.

한 가지 주의할 것은 **Serial.setup( ) 함수로 설정한 속도로 터미널의 통신 속도로 같게 맞추어 주어야만 정상적으로 통신이 수행된다**는 것이다. 터미널의 하단에 속도를 설정할 수 있는 콤보박스가 있으며 그것을 이용하여 통신 속도를 지정해 줄 수 있다.



[그림] 터미널의 통신 속도 지정

보레이트**(baud rate)**란 시리얼 통신의 속도를 나타내는 숫자로서 단위는 초당 전송되는 비트 수이다. 따라서 9600이라면 초당 9600비트를 전송할 수 있다. 1바이트는 8비트이므로

이는 초당 1200바이트, 즉 1.2kbyte 를 전송할 수 있는 속도이다. 9600외에도 19200, 57600, 115200 등 다양한 속도를 지정해 줄 수 있으며 시리얼 통신으로 지정해 줄 수 있는 최대 속도는 약 1메가 bps 정도이다. 보레이트가 너무 크면 통신이 불안정해질 수 있으니 적당한 속도로 설정해야 하며 많 사용되는 속도는 9600 혹 115200 이다.

※ 아두이노(C++) 프로그램에서 식별자(identifier, 즉 이름)는 다음과 같은 관행이 있다.

- 클래스 이름은 대문자로 시작한다.
- 변수나 함수(클래스 멤버 함수)는 소문자로 시작한다.
- 상수는 전체를 대문자로 사용한다.

예를 들어 Serial 은 대문자로 시작했으니 클래스 이름이다. print()는 Seiral의 멤버 함수이므로 소문자로 시작한다. pinMode(), digitalWrite(), digitalRead() 도 (전역) 함수이므로 소문자로 시작한 것을 알 수 있다. INPUT, OUTPUT, HIGH, LOW 등은 전체가 대문자이므로 #define 전처리문에 의해서 정의된 상수이다.

3.3 두 번째 예제

두 번째 예제로 아두이노 디지털핀에 연결된 버튼을 누르면 메세지를 PC쪽에 보내는 예제를 작성해 보자. 택스위치가 8번 핀에 연결되어 있다고 가정하고 그 버튼이 눌려졌다면 “pressing ”라는 문자열이 터미널에 표시되도록 해보자. Serial.print() 함수를 이용하면 매우 간단하게 작성할 수 있다.

```
#define SW 8
void setup() {
    Serial.begin(115200);
    pinMode(SW, INPUT_PULLUP);
}
void loop() {
    if (digitalRead(SW) == LOW) {
        Serial.print("pressing ");
    }
}
```

이 예제를 실행시키면 버튼을 누르고 있으면 “pressing ”라는 문자열이 연속적으로 PC로 전송되게 된다.

3.4 세 번째 예제

세 번째 예제로 이번에는 PC에서 문자 하나를 받아서 그것이 ‘0’이면 LED를 끄고 ‘1’이면 LED를 켜는 프로그램을 작성해 보자. 이 경우 아두이노는 데이터가 사용자로부터 들어올 때까지 대기 상태로 있다가 데이터가 읽히면 거기에 맞는 동작을 수행해야 한다.

```
#define LED 13
void setup() {
    pinMode(LED, OUTPUT);
    Serial.begin(115200);
}
void loop() {
    if ( Serial.available() ) {
        char command = Serial.read();
        if (command == '0') {
            digitalWrite(LED, LOW);
            Serial.println("LED off.");
        } else if (command == '1') {
            digitalWrite(LED, HIGH);
            Serial.println("LED on.");
        } else {
            Serial.print("Wrong command :");
            Serial.println(command);
        }
    }
}
```

이 예제에서는 다음과 같은 함수들이 사용되었다.

- int Serial.available() 함수
  - 전송되어 내부버퍼(64바이트)에 저장된 데이터의 개수를 반환한다.
  - 입력인수는 없다.
- int Serial.read() 함수
  - 전송되어 내부 버퍼(64바이트)에 저장된 데이터 중 가장 첫 번째 데이터(ASCII코드)를 읽어서 반환한다.
  - 이 함수가 수행되면 내부 버퍼의 크기는 하나가 줄어든다.
  - 내부 버퍼가 비었다면 -1을 반환하다.
- int Serial.println() 함수

- 출력 문자열의 끝에 줄바꿈 기호 '\r\n' 가 자동으로 붙는다는 점 외에는 Serial.print()함수와 동일한 동작을 수행한다.

이 예제에서는 USRT 통신기의 버퍼(buffer)의 개념을 이해할 필요가 있다.

- 버퍼란 전송된 데이터가 일시적으로 저장되는 내부 메모리이다
- 전송된 데이터는 받은 순서대로 버퍼에 저장된다.

아두이노 우노의 시리얼 통신부의 내부 버퍼의 크기는 64바이트이다. 전송되어 저장된 데이터를 사용하려면 내부 버퍼에서 이 데이터를 읽어내야 하는데 이때 사용되는 함수가 Serial.read()함수이다. 가장 먼저 전송된 데이터 하나를 읽어낸 후 그 데이터는 버퍼에서 삭제되며, 만약 버퍼가 비었다면 -1을 반환한다. 따라서 버퍼에 읽어낼 데이터가 있는지 없는지를 먼저 검사하는 것이 일반적이고 이때 사용되는 함수가 Serial.available()이다. 이 함수는 버퍼에 저장된 데이터의 개수를 반환하며 만약 버퍼가 비어있다면 0을 반환한다.

```
if ( Serial.available() ) { 명령어들 }
```

또한 if() 문은 조건을 검사하는 명령어다. 만약 조건문이 참이라면 {명령어들}이 수행되는데 **C++**에서 조건식은 **0**값만을 거짓으로 그 외의 값들은 참으로 간주한다. 따라서 0이 아니라면 명령어들이 수행되고 이로부터 버퍼에 데이터가 하나라도 있다면 명령어들이 수행된다는 것을 알 수 있다.

이런 사항들을 이해하였다면 이 예제를 이해하는데 어려움이 없을 것이다. 데이터가 전송되어 오면 그것을 읽어들여서 '1'이면 LED를 켜고, '0'이면 끈다. 그 이외의 데이터에 대해서는 잘못된 명령이라는 문자열을 출력한다.

3.5 네 번째 예제

네 번째로 11번 핀에 부저가 연결되었다고 가정하고 명령어를 하나 받아서 다음과 같은 동작을 수행하는 예제를 작성해 보도록 하겠다.

- '0' : LED를 끈다.
- '1' : LED를 켠다.
- '2' : 부저를 짧게 한 번 울린다. (삑)
- '3' : 부저를 짧게 두 번 울린다.(삐삑)

- 그 외의 명령어들은 오류 메세지를 출력한다.

이 예제의 경우는 처리해야 될 경우의 수가 많기 때문에 if-else 명령보다는 switch-case 명령이 더 효율적이다. 프로그램을 작성하면 다음과 같다.

```
#define LED 7
#define BUZ 11
void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BUZ, OUTPUT);
  Serial.begin(9600);
}
void loop() {
  if ( Serial.available() ) {
    char command = Serial.read();
    switch (command)  {
      case '1':
        digitalWrite(LED, HIGH);
        Serial.println("LED on.");//line
        break;
      case '0':
        digitalWrite(LED, LOW);
        Serial.println("LED off.");
        break;
      case '2':
        beep(50);
        Serial.println("beep.");
        break;
      case '3':
        beep(50);
        delay(100);
        beep(50);
        Serial.println("beep.");
        break;
      default:
        Serial.println("wrong command.");
        break;
    }//switch
  }//if
}// loop()

void beep(int on_time) {
  digitalWrite(BUZ, HIGH);
  delay(on_time);
  digitalWrite(BUZ, LOW);
}
```

3.6 초음파 센서로 거리 측정하기

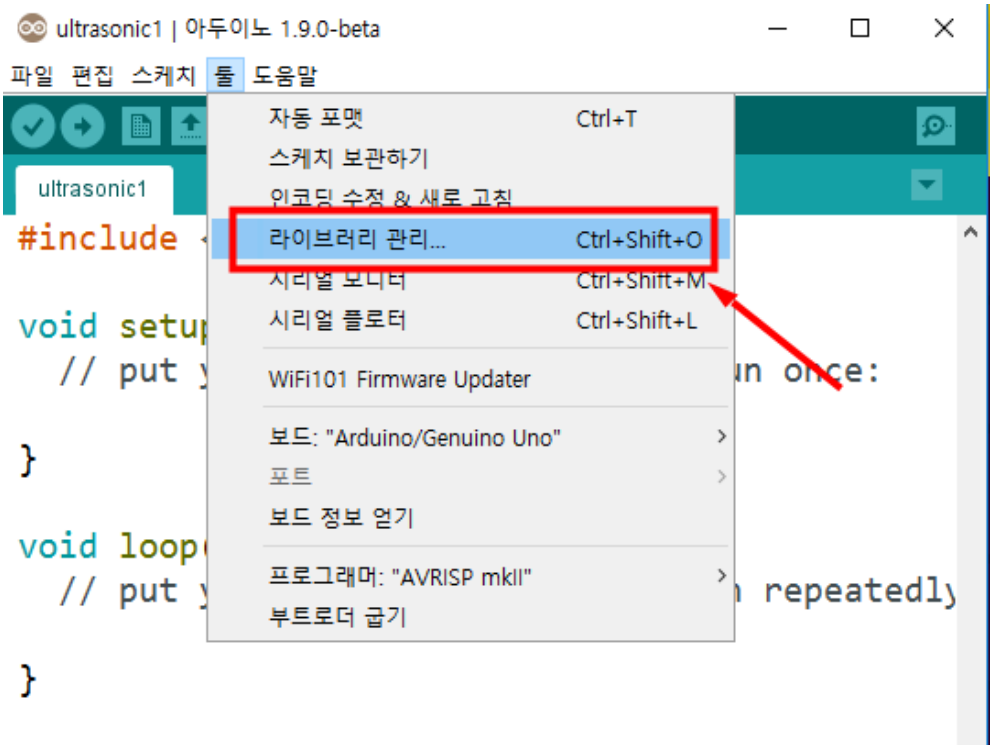
장애물과의 거리를 측정하는 데 초음파 센서가 많이 사용된다. 초음파 센서 모듈은 초음파를 발생시켜서 물체에 반사되어 돌아오는 시간을 측정해서 장애물과의 거리를 구할 수 있는 장치이다. 시중에서 쉽게 구할 수 있는 저가형 초음파 센서 모듈(HC-SR04)는 다음 그림과 같이 4핀 인터페이스를 갖는다.

- Vcc(5 V)/GND는 전원핀
- Trig (trigger)는 초음파를 발생시키는 펄스 신호를 내보내는 핀
- Echo는 반사파가 감지되었음을 알려주는 신호선

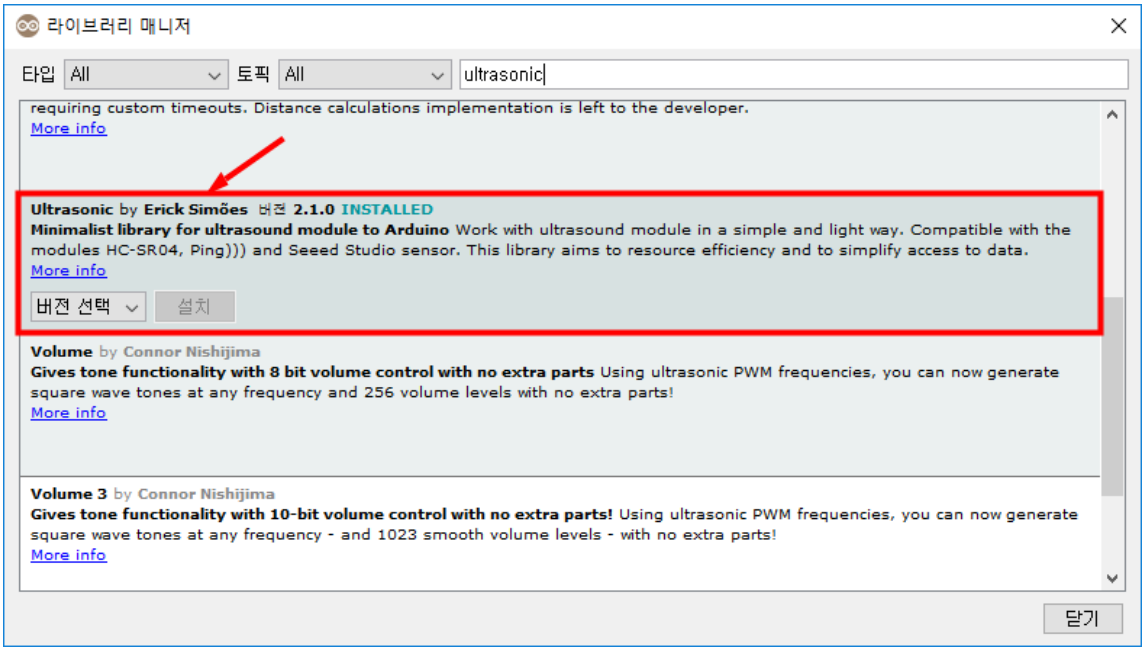
따라서 이 모듈을 사용하려면 아두이노의 디지털 핀 두 개가 필요하다.

[그림 3.6.1] 초음파센서 모듈 HC-SR04

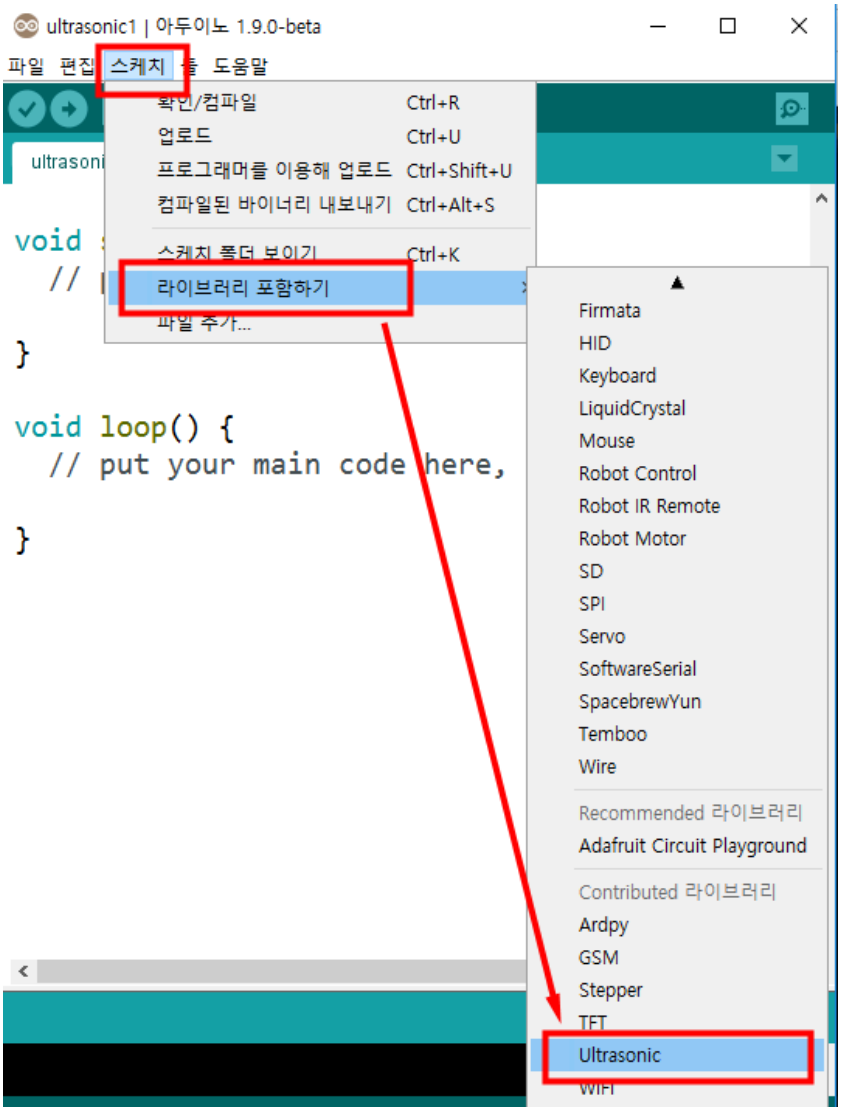
이것을 이용하여 거리를 측정하려면 전용 아두이노 라이브러리를 사용하는 것이 편리하다. 미리 작성된 먼저 아두이노 IDE에서 툴>라이브러리 관리 메뉴를 선택한다.



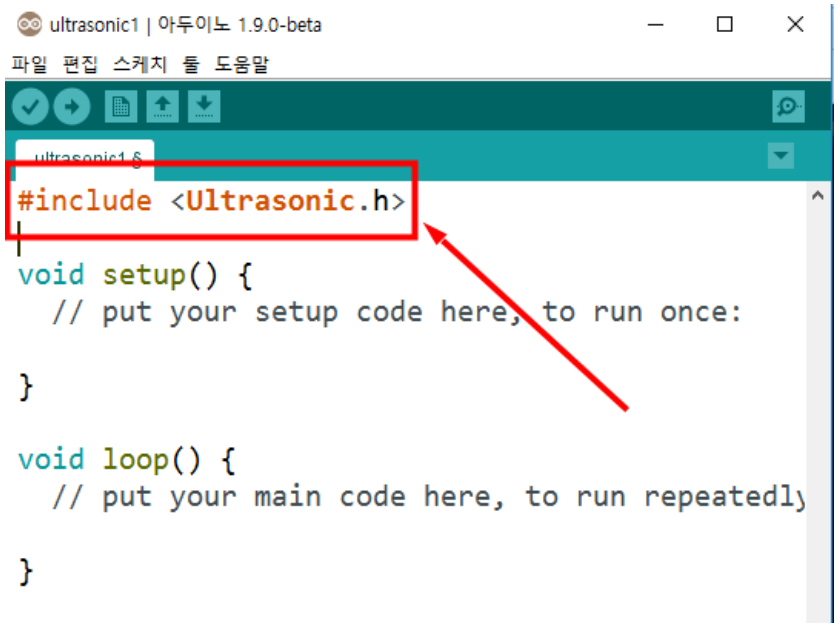
그러면 다음과 같은 '라이브러리 매니저'창이 뜬다. 여기 검색창에서 ultrasonic 으로 검색하여 'Ultrasonic by Erick Simoes' 라는 라이브러리를 선택한 후 설치한다.



설치된 이후에는 다음과 같이 아두이노 IDE에서 스케치>라이브러리 포함하기 메뉴에 Ultrasonic 항목이 나타난다.



이것을 선택하면 다음과 같이 편집창에 Ultrasonic.h 헤더파일이 인클루드 된다.



이 라이브러리에는 **Ultrasonic 클래스**가 정의되어 있는데 자세한설명과 최신 버젼은 여기에서 확인할 수 있다. 클래스 객체는 다음과 같이 만든다.

```
Ultrasonic us(Trig, Echo [, TIME_OUT_IN_MS] );
```

만약 Trig핀이 2번, Echo핀이 3번에 연결되어 있다면 다음과 같이 객체를 생성하면 된다.

```
Ultrasonic us(2, 3);
```

생성된 객체 us의 멤버 함수는 두 개인데 distanceRead()와 setTimeout() 이다. 이 중에서 distanceRead() 함수는 측정된거리를 cm단위의 정수(unsigned int)로 반환해 준다. 만약 인치단위로 반환받고 싶다면 첫 번째 인자로 INC 라는 상수를 주면 된다.

- us.distanceRead() : 측정 거리를 cm단위로 반환
- us.distanceRead(INC) : 측정 거리를 inch 단위로 반환

setTimeout() 멤버함수는 장애물이 감지되지 않을 때의 음파를 기다리는 최대 시간을 설정하는데 사용되며 기본값을 사용하는 것으로 충분하므로 자주 사용되지 않는다.

3.6.1 첫 번째 예제

첫 번째 예제로 측정된 거리를 시리얼 통신으로 받아서 시리얼 모니터에 표시하도록 하는 프로그램을 작성해 보자. Ultrasonic 라이브러리를 이용하면 매우 쉽게 프로그램을 작성할 수 있다.

```
#include <Ultrasonic.h>

Ultrasonic us(2,3);

void setup() {
  Serial.begin(115200);
}

void loop() {
  int dist = us.distanceRead(); //cm(정수)
  Serial.print(dist);
  Serial.println(" cm");
  delay(100);
}
```

3.6.2 두 번째 예제

아두이노에는 다음 표와 같은 시간과 관련된 함수들이 있다.

[표] 아두이노의 시간과 관련된 함수들

반환값	함수명	기능
ulong	millis()	프로그램 시작 후 경과시간(밀리초) 반환
ulong	micros()	프로그램 시작 후 경과시간(마이크로초) 반환
void	delay(uint ms)	주어진 밀리초만큼 지연
void	delayMicroseconds(uint us)	주어진 마이크로초만큼 지연

이 중 micros()함수를 이용하여 초음파 센서로 거리를 측정하는 데 얼마만큼의 시간이 걸리는 지 확인해 보는 프로그램을 작성해보자.

```
  unsigned long timeStart = micros();
  int dist = us.distanceRead(); //cm(정수)
  Serial.print(micros()-timeStart);
```

전체 프로그램은 다음과 같다.

```
#include <Ultrasonic.h>

Ultrasonic us(2,3);

void setup() {
  Serial.begin(115200);
}

void loop() {
  unsigned long timeStart = micros();
  int dist = us.distanceRead(); //cm(정수)
  Serial.print(micros()-timeStart);
  Serial.print("us, ");
  Serial.print(dist);
  Serial.println("cm");
  delay(100);
}
```

3.6.3 세 번째 예제

여기에서는 초음파센서와 부저를 조합하여 거리에 따라서 부저음이 울리는 간격이 달라지도록 하는 프록램을 작성해 보자.

```
#include <Ultrasonic.h>
#define BUZ 11
Ultrasonic us(2, 3);

void setup() {
  pinMode(BUZ, 11);
  Serial.begin(115200);
}

void loop() {
  digitalWrite(BUZ, HIGH);
  delay(50);
  digitalWrite(BUZ, LOW);
  int n = 5;
  for (int k = 0; k < n; k++) {
    int dist = us.distanceRead(); //cm
    Serial.println(dist);
    if (dist>=25 || dist==0) {
      n = 5;
    } else if (dist>=20) {
      n = 4;
    } else if (dist>=15) {
```

```
      n = 3;
    } else if (dist>=10) {
      n = 2;
    } else if (dist>=5){
      n = 1;
    } else {
      break;
    }
    delay(50);
  }
}
```