

# Chapter 03

## 아두이노 인공지능 드론 살펴보기

이번 장에서는 아두이노 인공지능 드론을 구성하는 시리얼 출력과 입력, LED, 버튼, DC 모터, 네오픽셀, RGB LCD에 대한 테스트를 수행하도록 합니다.

## 01 문자열 보내기

앞에서 우리는 `Serial.println` 함수를 사용하여 다음과 같은 메시지를 PC로 보냈습니다.

```
Hello PC^^. I'm an ESP32~  
Hello PC^^. I'm an ESP32~  
Hello PC^^. I'm an ESP32~  
Hello PC^^. I'm an ESP32~  
Hello PC^^. I'm an ESP32~
```

아두이노가 여러분에게 인사를 한 것이죠. `Serial.println` 함수는 아주 유용한 함수입니다. 아두이노의 상태가 어떤지 우리에게 알려주는 주인공이 바로 `Serial.println` 함수입니다. 여러분은 앞으로 아두이노 스케치를 작성하다 버튼이나 센서의 값을 알고 싶은 경우가 있을 수 있습니다. 이 때 필요한 함수가 바로 `Serial.println` 함수입니다.

여러분은 아두이노 스케치를 통해 메시지를 출력할 때 다음 세 함수를 주로 사용하게 됩니다.

```
Serial.begin(speed)  
Serial.println(val)  
Serial.print(val)
```

### Serial.begin

`Serial.begin`은 PC로 메시지를 보낼 때 데이터의 속도를 설정하는 함수입니다.

**Serial.begin(speed);**

❶

❶ PC로 메시지를 보낼 때 데이터 속도

(300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200)

`Serial.begin` 함수는 PC로 메시지를 보낼 때 데이터의 속도를 설정합니다. `speed` 인자를 통해 설정할 수 있는 속도 값은 다음과 같습니다.

```
300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200
```

우리 책에서는 주로 115200을 사용합니다. 115200bps는 초당 115200 비트를 보내는 속도입니다. 시리얼 포트를 통해 문자 하나를 보내는데 10비트가 필요합니다. 그러므로 1초에

115200/10 = 11520 문자를 보내는 속도입니다. 11520 문자는 A4 용지 기준 5~6페이지 정도의 양입니다. 비트는 0 또는 1을 담을 수 있는 데이터 저장의 가장 작은 단위입니다.

PC로 메시지를 보내기 위해서는 Serial.print 또는 Serial.println 함수를 사용합니다.

## Serial.println

Serial.println은 PC로 메시지를 보내는 함수입니다.

**Serial.println(val);**

❶

❶ PC로 보낼 메시지로 정수, 실수, 문자열을 보낼 수 있다.

Serial.println 함수는 메시지 출력 후, 커서 위치를 다음 줄 첫 번째 칸으로 옮기는 엔터 키 입력 효과를 줍니다. val 인자를 통해 보낼 수 있는 값은 문자열, 정수 값, 실수 값을 보낼 수 있습니다.

Serial.print 함수는 메시지만 출력하고, 커서의 위치는 옮기지 않습니다.

다음과 같이 형식(format) 인자를 이용하여 메시지 형식을 좀 더 자세하게 줄 수도 있습니다.

Serial.println(val, format)

Serial.print(val, format)

format 인자의 경우 val 인자가 정수일 경우엔 진법(DEC, HEX, OCT, BIN)을 설정할 수 있으며, 실수인 경우엔 소수점 이하 표시할 자리수를 설정할 수 있습니다. 뒤에서 예제를 통해 사용법을 살펴봅니다.

\*\*\* 우리는 이 함수들을 이용하여 문자열과 숫자를 출력하게 됩니다. 숫자의 경우는 정수와 실수로 나눌 수 있습니다. 정수의 경우는 주로 10진수와 16진수로 표현할 수 있으며, 아두이노의 경우엔 2진수 표현도 가능합니다. 실수의 경우는 자릿수를 얼마나 나타낼지를 결정할 수 있습니다.

\*\*\* 아두이노 ESP32의 경우 C에서 사용하는 printf 함수를 지원합니다.

## 01 여러 형식의 자료 내보내기

여기서는 Serial.println 함수를 이용하여 문자열, 숫자, 문자를 출력해봅니다.

1. 다음과 같이 예제를 작성합니다.

311.ino

```
01 void setup() {  
02     Serial.begin(115200);  
03  
04     Serial.println("Hello PC^^. I'm an ESP32~");  
05     Serial.println(78);  
06     Serial.println(1.23456);  
07     Serial.println('N');  
08 }  
09  
10 void loop() {  
11  
12 }
```

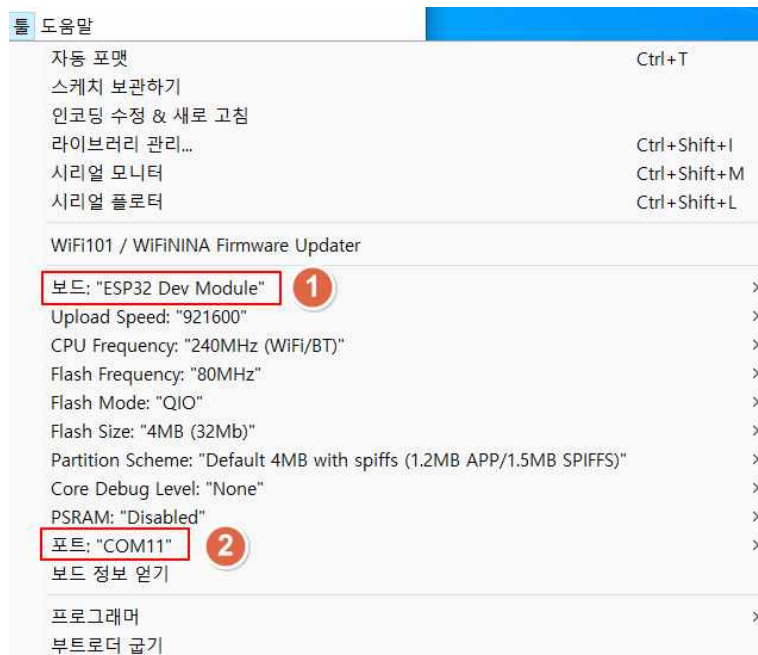
04 : 문자열을 출력합니다.

05 : 정수 78을 10진수 문자열로 변환하여 출력합니다.

06 : 실수 1.23456을 10진 실수 문자열로 변환하여 출력합니다.

07 : 문자 N을 문자열로 변환하여 출력합니다.

2. [툴] 메뉴를 이용하여 보드, 포트를 다음과 같이 선택합니다.



3. 업로드를 수행합니다.



4. [시리얼 모니터] 버튼을 클릭합니다.

5. 시리얼 모니터 창이 뜨면, 우측 하단에서 통신 속도를 115200으로 맞춰줍니다.

새 줄

115200 보드레이트

출력 지우기

6. 다음에 표시된 [EN] 버튼을 눌러 재부팅을 수행합니다.



7. 출력결과를 확인합니다.

```

Hello PC^^. I'm an ESP32~
78
1.23
N

```

1.23456 실수의 경우 기본적으로 소수점 아래 두 자리만 출력하는 것을 볼 수 있습니다.

**용어설명: C의 자료형** C에서 일반적으로 사용하는 자료 형은 int, double, char \*, char입니다. int는 정수 값을 담을 수 있는 자료 형을, double은 실수 값을 담을 수 있는 자료 형을, char \*는 문자열의 첫 문자의 주소를 담을 수 있는 자료 형을, char는 한 문자를 담을 수 있는 자료 형을 나타냅니다. 정수의 경우엔 10진수와 16진수 두 종류가 있습니다. 10진수의 경우엔 개수나 번호 등에 사용되며, 16진수는 메모리 주소 값이나 특정한 비트의 값을 나타낼 때 사용합니다. 10진수는 주로 사칙연산자나 비교연산자와 같이 사용되며, 16진수는 주로 비트연산자와 같이 사용됩니다.

## 02 여러 형식의 숫자 내보내기

여기서는 Serial.println 함수를 이용하여 10진수와 16진수 정수를 출력해 봅니다. 또, 10진 실수의 소수점 이하 출력을 조절해 봅니다.

1. 다음과 같이 예제를 작성합니다.

312.ino

```
01 void setup() {  
02     Serial.begin(115200);  
03  
04     Serial.println(78, DEC);  
05     Serial.println(78, HEX);  
06     Serial.println(78, BIN);  
07  
08     Serial.println(1.23456, 0);  
09     Serial.println(1.23456, 2);  
10     Serial.println(1.23456, 4);  
11 }  
12  
13 void loop() {  
14  
15 }
```

04 : 정수 78을 10진수 문자열로 변환하여 출력합니다.

05 : HEX 형식은 정수를 16진수 문자열로 변환하는 형식입니다. 여기서는 정수 78을 16진수 문자열로 변환하여 출력합니다.

06: 정수 78을 2진수 문자열로 변환하여 출력합니다.

08 : 여기서는 실수 1.23456을 소수점 이하 0개까지 10진 실수 문자열로 변환하여 출력합니다.

09 : 여기서는 실수 1.23456을 소수점 이하 2개까지 10진 실수 문자열로 변환하여 출력합니다.

10 : 여기서는 실수 1.23456을 소수점 이하 4개까지 10진 실수 문자열로 변환하여 출력합니다.

2. 업로드를 수행합니다.



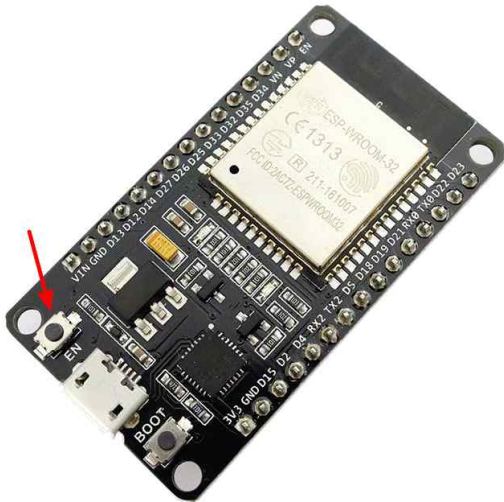
3. [시리얼 모니터] 버튼을 클릭합니다.



4. 시리얼 모니터 창이 뜨면, 우측 하단에서 통신 속도를 115200으로 맞춰줍니다.

새 줄 ▼	115200 보드레이트 ▼	출력 지우기
-------	----------------	--------

5. 다음에 표시된 [EN] 버튼을 눌러 재부팅을 수행합니다.



6. 출력결과를 확인합니다.

```
78
4E
1001110
1
1.23
1.2346
```

## 02 LED 켜고 끄기

여러분은 다음과 같이 유튜브 등에서 아두이노를 이용하여 LED를 깜빡이는 동영상을 본 적이 있나요?



LED를 깜빡이게 하는 주인공이 바로 `digitalWrite` 함수입니다.

여러분은 아두이노 스케치를 통해 LED를 제어할 때 다음 세 함수를 주로 사용하게 됩니다.

```
pinMode(pin, mode)
digitalWrite(pin, value)
delay(ms)
```

### pinMode

`pinMode`란 특정 핀을 출력 또는 입력 모드로 설정하는 명령어입니다.

**`pinMode(pin, mode);`**

①      ②

① 설정하고자 하는 핀 번호

② 설정하고자 하는 모드로 입력일 때는 INPUT, 출력일 때는 OUTPUT

`pinMode` 함수는 특정한 핀을 출력으로 사용할지 입력으로 사용할지를 설정합니다. `pin` 인자로는 보드 상에 나와 있는 숫자를 사용합니다. 본 책에서 사용하는 아두이노 ESP32의 경우 앞에서 살펴본 그림에서 화살표가 가리키는 주황 색깔의 핀에 해당하는 숫자를 사용합니다. `mode` 인자로는 OUTPUT, INPUT, INPUT\_PULLUP을 사용할 수 있습니다. LED를 켜기 위해서는 0 또는 1을 LED로 쓰는 개념이기 때문에 OUTPUT으로 설정합니다. 버튼의 경우 버튼의 값을 읽는 개념이기 때문에 INPUT으로 설정합니다. 버튼의 경우 외부에 저항을 이용하여 회로를 구성하는데, 외부에 저항을 사용하지 않고 아두이노의 칩 내부에 있는 저항을 이용할 경우엔 INPUT\_PULLUP으로 설정합니다.

### digitalWrite



digitalWrite란 특정 핀을 HIGH 또는 LOW로 설정하는 명령어입니다.

```
digitalWrite(pin, value);
```

①      ②

① 제어하고자 하는 핀 번호

② HIGH 또는 LOW

digitalWrite 함수는 디지털 핀으로 HIGH(=1) 또는 LOW(=0) 값을 씁니다. 아두이노 ESP32의 주황 색깔의 핀의 경우 pinMode 함수를 통해 해당 핀이 OUTPUT으로 설정되었을 때, HIGH 값의 경우엔 해당 핀이 3.3V로 설정되며, LOW 값의 경우엔 0V로 설정됩니다.

## delay

delay란 인자로 주어진 시간만큼 프로그램의 진행을 멈춥니다.

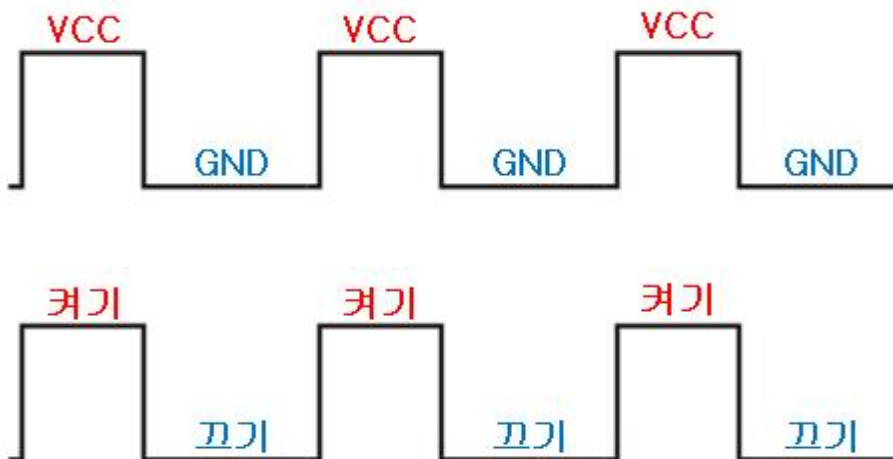
```
delay(ms);
```

①

① 멈춰야할 밀리초(ms : unsigned long 형)

\*\*\* unsigned long은 변수형의 한 종류로 아두이노 스케치에서 0~4,294,967,295 ( $2^{32} - 1$ ) 범위의 0과 양의 정수 값을 갖습니다.

여기서는 digitalWrite 함수를 이용하여 LED를 켜보고 꺼보는 예제를 수행해 봅니다. 또 반복적으로 켜고 끄는 주기를 짧게 해가며 아래 그림과 같은 사각 파형에 대해서도 알아보도록 합니다.



## 01 LED 켜고 끄기 확인하기

먼저 digitalWrite 함수를 이용하여 LED를 주기적으로 켜고 꺼봅니다. 여기서는 아두이노 ESP32의 2번 핀에 연결된 LED를 주기적으로 켜고 꺼 봅니다.



1. 다음과 같이 예제를 작성합니다.

321.ino

```
01 const int LED = 2;
02
03 void setup() {
04     pinMode(LED, OUTPUT);
05 }
06
07 void loop() {
08     digitalWrite(LED, HIGH);
09     delay(500);
10     digitalWrite(LED, LOW);
11     delay(500);
12 }
```

01 : LED 상수에 2번 핀을 할당합니다.

04 : pinMode 함수를 이용하여 LED를 출력으로 설정하고 있습니다. pinMode 함수는 digitalWrite 함수를 이용하여 HIGH, LOW 값을 쓰고자 할 때 사용하는 함수입니다.

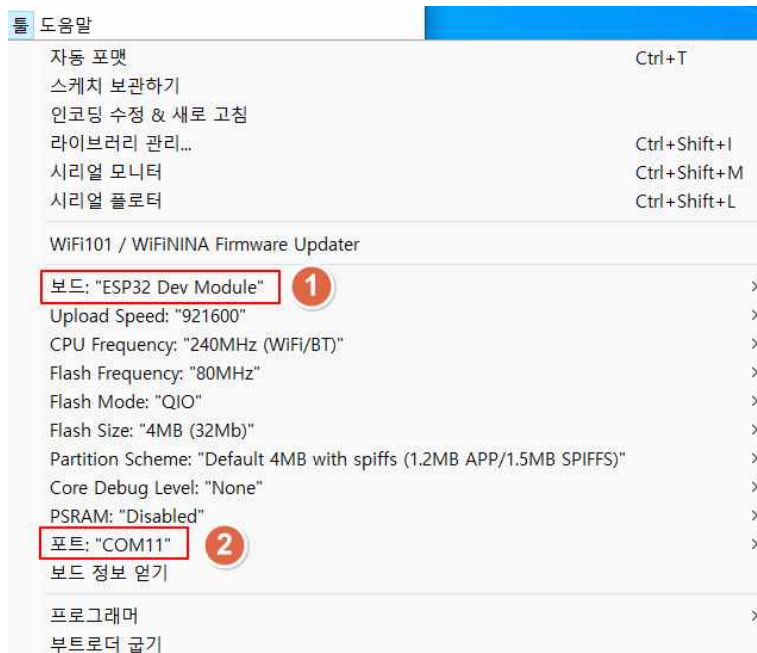
08 : digitalWrite 함수를 이용하여 LED에 HIGH 값을 씁니다. 그러면 LED는 켜지게 됩니다.

09 : 500 밀리초간 지연을 줍니다. delay 함수는 아두이노가 아무것도 수행하지 않고 일정시간을 기다리게 하는 함수입니다. 함수의 인자로 주어지는 500은 밀리 초 단위입니다. 여기서는 500 밀리 초 동안 아두이노가 아무것도 수행하지 않습니다.

10 : digitalWrite 함수를 이용하여 LED에 LOW 값을 씁니다. 그러면 LED는 꺼지게 됩니다.

11 : 500 밀리초간 지연을 줍니다.

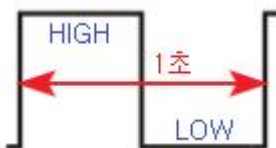
2. [툴] 메뉴를 이용하여 보드, 포트를 다음과 같이 선택합니다.



3. 업로드를 수행합니다.



4. LED의 동작을 확인합니다. 1초 주기로 LED가 켜졌다 꺼졌다 하는 것을 확인합니다. 즉, 1Hz의 주파수로 LED가 점멸하는 것을 확인합니다.



LED의 점등은 LED(=2) 핀을 통해 나오는 HIGH 값에 의해 발생합니다. LED의 소등은 LED 핀을 통해 나오는 LOW 값에 의해 발생합니다. 즉, LED 핀으로는 위 그림과 같이 HIGH값과 LOW 값이 1초 주기로 나오게 되며, 이 값들에 의해 LED는 점멸을 반복하게 됩니다. 그리고 이 경우 여러분은 LED가 점멸 하는 것을 느낄 수 있습니다.

\*\*\* Hz : 같은 동작이 1초에 1 번씩 반복될 때 우리는 1Hz로 동작한다고 합니다. 같은 동작이 1초에 2 번씩 반복될 때 우리는 2Hz로 동작한다고 합니다.

## 02 LED 켜고 끄기 간격 줄여보기

여기서는 digitalWrite 함수를 이용하여 아래와 같은 사각 파형에 대한 주파수와 상하비의 개념을 이해해 보도록 합니다.



주파수란 1초간 반복되는 사각 파형의 개수를 의미하며, 상하 비란 사각 파형의 HIGH 값과 LOW 값의 비를 의미합니다.

이제 LED의 점멸 간격을 줄여보도록 합니다. 그러면 여러분은 좀 더 조밀하게 LED가 점멸하는 것을 느낄 것입니다.

1. 다음과 같이 이전 예제를 수정합니다.

322.ino

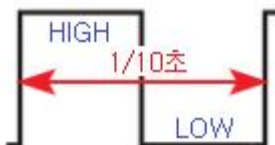
```
01 const int LED = 2;
02
03 void setup() {
04     pinMode(LED, OUTPUT);
05 }
06
07 void loop() {
08     digitalWrite(LED, HIGH);
09     delay(50);
10     digitalWrite(LED, LOW);
11     delay(50);
12 }
```

09, 11 : 500을 50으로 변경합니다.

2. 업로드를 수행합니다.



3. LED의 동작을 확인합니다. 이 예제의 경우 LED는 초당 10번 점멸하게 됩니다. 즉, 10Hz의 주파수로 점멸하게 됩니다.



그림과 같은 파형이 초당 10개가 생성됩니다. 이 경우에도 여러분은 반복적으로 LED가 점멸하는 것을 느낄 것입니다. 그러나 그 간격은 더 조밀하게 느껴질 것입니다.

### 03 LED 켜고 끄기를 밝기로 느껴보기

LED의 점멸 간격을 더 줄여보도록 합니다. 여기서 여러분은 LED의 점멸을 느끼지 못하게 될 것입니다. 오히려 LED가 일정한 밝기로 켜져 있다고 느낄 것입니다.

1. 다음과 같이 예제를 수정합니다.

323.ino

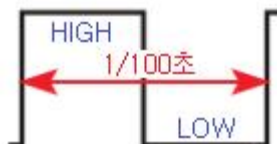
```
01 const int LED = 2;
02
03 void setup() {
04     pinMode(LED, OUTPUT);
05 }
06
07 void loop() {
08     digitalWrite(LED, HIGH);
09     delay(5);
10     digitalWrite(LED, LOW);
11     delay(5);
12 }
```

09, 11 : 50을 5로 변경합니다.

2. 업로드를 수행합니다.



3. LED의 동작을 확인합니다. 이 예제의 경우 LED는 초당 100번 점멸 하게 됩니다. 즉, 100Hz의 주파수로 점멸하게 됩니다.



그림과 같은 파형이 초당 100개가 생성됩니다. 이제 여러분은 LED가 점멸하는 것을 느끼지 못할 것입니다. 오히려 LED가 일정하게 켜져 있다고 느낄 것입니다.

일반적으로 이러한 파형이 초당 50개 이상이 되면, 즉, 50Hz 이상의 주파수로 LED 점멸을

반복하면 우리는 그것을 느끼기 어렵습니다.

## 04 LED 어둡게 하기

이제 delay 함수를 조절하여 LED의 밝기를 어둡게 해 봅시다. 이전 예제의 경우 LED는 100Hz의 속도로 50%는 점등을, 50%는 소등을 반복하였습니다. 그리고 이 경우 우리는 LED의 밝기를 평균값인 50%의 밝기로 느꼈습니다. 만약 LED에 대해 10%는 점등을, 90%는 소등을 반복한다면 우리는 LED의 밝기를 어떻게 느낄까요? 평균 10%의 밝기로 느끼게 되지 않을까요? 예제를 통해 확인해 보도록 합니다.

1. 다음과 같이 예제를 수정합니다.

324.ino

```
01 const int LED = 2;
02
03 void setup() {
04     pinMode(LED, OUTPUT);
05 }
06
07 void loop() {
08     digitalWrite(LED, HIGH);
09     delay(1);
10     digitalWrite(LED, LOW);
11     delay(9);
12 }
```

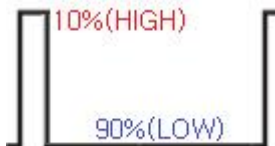
09 : 5를 1로 변경합니다.

11 : 5를 9로 변경합니다.

2. 업로드를 수행합니다.



3. LED의 동작을 확인합니다. 이 예제의 경우도 LED는 초당 100번 점멸 하게 됩니다. 즉, 100Hz의 주파수로 점멸하게 됩니다. 그러나 10%는 점등 상태로, 90%는 소등 상태로 있게 됩니다. 그래서 우리는 LED의 밝기가 이전 예제보다 낮다고 느끼게 됩니다.



그림에서 LED는 실제로 10%만 점등 상태이지만 100Hz의 주파수로 점멸하기 때문에 우리는 10%의 평균 밝기로 느끼게 됩니다. 10%는 HIGH 값에 의해 켜져 있고 90%는

LOW 값에 의해 꺼져있으며, 이 경우 (HIGH:LOW)=(1:9)가 되게 됩니다. 즉, 상하비가 1:9가 됩니다.

## 05 LED 밝게 하기

이제 반대로 LED의 밝기를 밝게 해 봅니다.

1. 다음과 같이 예제를 수정합니다.

325.ino

```
01 const int LED = 2;
02
03 void setup() {
04     pinMode(LED, OUTPUT);
05 }
06
07 void loop() {
08     digitalWrite(LED, HIGH);
09     delay(9);
10     digitalWrite(LED, LOW);
11     delay(1);
12 }
```

09 : 1를 9로 변경합니다.

11 : 9를 1로 변경합니다.

2. 업로드를 수행합니다.



3. LED의 동작을 확인합니다. 이 예제의 경우도 LED는 초당 100번 점멸 하게 됩니다. 즉, 100Hz의 주파수로 점멸하게 됩니다. 그러나 90%는 점등 상태로, 10%는 소등 상태로 있게 됩니다. 그래서 우리는 LED가 이전 예제에 비해 아주 밝다고 느끼게 됩니다.



그림에서 LED는 실제로 90%만 점등 상태이지만 100Hz의 주파수로 점멸하기 때문에 우리는 90%의 평균 밝기로 느끼게 됩니다. 90%는 HIGH 값에 의해 켜져 있고 10%는 LOW 값에 의해 꺼져 있으며, 이 경우 (HIGH:LOW)=(9:1)이 되게 됩니다. 즉, 상하비가 9:1이 됩니다.

상하비가 8:2가 되면 우리는 LED가 80%의 밝기로 켜져 있다고 느끼게 됩니다. 9:1에 해당되는 부분을 차례대로 다음과 같이 바꾸어 볼 수 있습니다.

0:10, 1:9, 2:8, 3:7 ... 10:0
------------------------------

우리는 HIGH와 LOW의 상하 비에 따라 LED의 밝기를 조절할 수 있습니다.

## 06 LED 밝기 조절해 보기

여기서는 1초 간격으로 다음의 상하비로 LED의 밝기를 조절해 보도록 합니다.

0:10, 1:9, 2:8, 3:7 ... 10:0
------------------------------

즉, HIGH의 개수는 0부터 10까지 차례로 늘어나며, 반대로 LOW의 개수는 10부터 0까지 차례로 줄게 됩니다.

### 0.01초 간격으로 LED 밝기 11 단계 조절해보기

먼저 0.01초 간격으로 LED의 밝기를 11단계로 조절해 봅니다.

1. 다음과 같이 예제를 수정합니다.

326.ino

```
01 const int LED = 2;
02
03 void setup() {
04     pinMode(LED, OUTPUT);
05 }
06
07 void loop() {
08     for(int t_high=0;t_high<=10;t_high++) {
09         digitalWrite(LED, HIGH);
10         delay(t_high);
11         digitalWrite(LED, LOW);
12         delay(10-t_high);
13     }
14 }
```

08 : t\_high 변수를 0부터 10까지 1씩 증가시켜가면서, 중괄호 안쪽(8줄~13줄)의 동작을 수행합니다.

09, 10 : LED를 켜고 t\_high 시간만큼 기다립니다.

11, 12 : LED를 끄고 (10-t\_high) 시간만큼 기다립니다.



10, 12 :  $t\_high + (10 - t\_high) = 10$ 이 되어 for문을 한 번 도는 데는 10밀리 초 정도가 되며 for문 전체를 도는 데는 110밀리 초 정도가 됩니다.

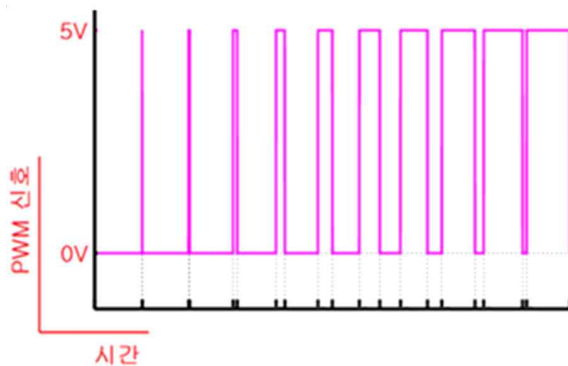
2. 업로드를 수행합니다.



3. LED의 동작을 확인합니다. 10밀리 초 간격으로 다음의 비율로 LED가 깜빡입니다.

0%, 10% 20%, 30%, ... 100%

아래와 같은 형태의 파형이 반복되면서 LED의 밝기가 변합니다.



이 예제의 경우 밝기의 변화가 너무 빨라 밝기가 변하는 것을 느끼기 어렵습니다. 깜빡임으로 느낄 수 있습니다.

## 0.1초 간격으로 LED 밝기 11 단계 조절해보기

다음은 0.1초 간격으로 LED의 밝기를 11단계로 조절해 봅니다.

1. 다음과 같이 예제를 수정합니다.

326\_2.ino

```
01 const int LED = 2;
02
03 void setup() {
04     pinMode(LED, OUTPUT);
05 }
06
07 void loop() {
08     for(int t_high=0;t_high<=10;t_high++) {
09         int cnt =0;
```

```

10         while(true) {
11             digitalWrite(LED, HIGH);
12             delay(t_high);
13             digitalWrite(LED, LOW);
14             delay(10-t_high);
15
16             cnt++;
17             if(cnt==10) break;
18         }
19     }
20 }

```

08 : for 문을 사용하여 t\_high 변수 값을 0부터 10까지 주기적으로 변경하고 있습니다. t\_high 변수 값은 16, 18 번째 줄에서 사용되며, LED를 통해 HIGH, LOW 값이 나가는 시간 값을 가집니다.

10 : 조건이 없는 while 문을 수행합니다. while 문을 나오는 조건은 21 번째 줄에 있으며, 1초 간격으로 나오게 됩니다.

09 : cnt 변수 생성 후, 0으로 초기화합니다.

16 : cnt 변수를 하나씩 증가시킵니다.

17 : cnt 변수가 10이 되면 break 문을 수행하여 while 문을 벗어납니다.

08 : cnt 변수를 선언한 후, 0으로 초기화합니다.

09 : 무한루프를 돌면서

15 : cnt 값을 하나씩 증가시킵니다.

16 : cnt 값이 10이 되면 내부 while 문을 나옵니다.

이렇게 하면 09~17줄을 cnt값이 0에서 9까지 10회 반복하게됩니다. 그러면  $0.001 * t\_high$  값을 유지하는 시간을 10밀리초(0.01초)에서 100밀리초(0.1초)로 늘릴 수 있습니다. for 문을 수행하는 시간도 110밀리초(0.11초)에서 1100밀리초(1.1초)로 늘릴 수 있으며, 우리는 LED 밝기의 변화를 느낄 수 있습니다.

2. 업로드를 수행합니다.



3. LED의 동작을 확인합니다. 1.1 초 주기로 다음의 비율로 LED가 밝아집니다.

0%, 10% 20%, 30%, ... 100%

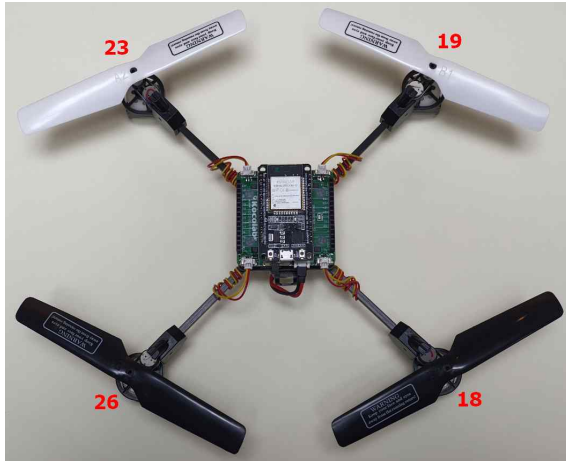
## 07 모터 회전 정지 반복해 보기

여기서는 digitalWrite 함수를 이용하여 모터를 돌렸다 멈췄다를 반복해 봅니다.

주의! 여기서 수행할 예제들의 경우엔 ESP32 아두이노 드론 기준으로 작성되었습니다. ESP32 아두이노 드론이 아닌 경우에는 수행하지 않도록 합니다. 모터의 용량에 따라 전선이 타는 경우도 있으니 주의하도록 합니다.

주의! 모터가 돌면 위험하니 독자 여러분이나 주변 사람들이 다치지 않도록 주의합니다.

여기서는 17 번 모터의 속도를 조절해 보도록 합니다. 본 책에서 다루고 있는 ESP32 아두이노 드론의 모터는 다음과 같이 핀 배치가 되어 있습니다.



16 17 25 26 변경

1. 다음과 같이 예제를 작성합니다.

2510.ino

```
01 const int fan_pin = 17;
02
03 void setup() {
04     pinMode(fan_pin, OUTPUT);
05 }
06
07 unsigned int howMany = 5;
08 void loop() {
09     if(howMany>0) {
10         howMany--;
11
12         digitalWrite(fan_pin, HIGH);
13         delay(100);
14         digitalWrite(fan_pin, LOW);
15         delay(900);
16     }
17 }
```

1 : fan\_pin 상수에 17 번 모터 핀을 할당하고 있습니다.

4 : pinMode 함수를 이용하여 fan\_pin을 출력으로 설정하고 있습니다. pinMode 함수는

digitalWrite 함수나 digitalRead 함수를 이용하여 HIGH, LOW 값을 쓰거나 읽고자 할 때 사용하는 함수입니다.

7 : howMany 변수를 선언한 후, 5로 초기화합니다. howMany 변수는 9, 10줄에서 사용하여 9~16줄의 수행 횟수를 결정합니다.

9 : howMany 변수 값이 0보다 크면

10 : howMany 변수 값을 1 감소시킵니다.

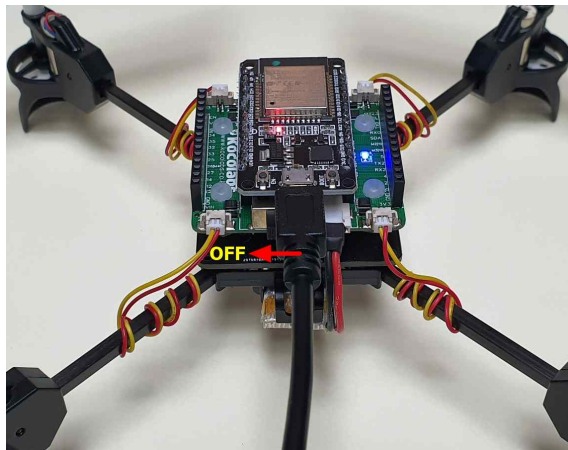
12 : digitalWrite 함수를 이용하여 fan\_pin에 HIGH 값을 쓰고 있습니다. 그러면 17 번 모터는 최고 속도로 돌게 됩니다.

13 : 0.1초간 지연을 줍니다.

14 : digitalWrite 함수를 이용하여 fan\_pin에 LOW 값을 쓰고 있습니다. 그러면 17 번 모터는 멈추게 됩니다.

15 : 0.9초간 지연을 줍니다.

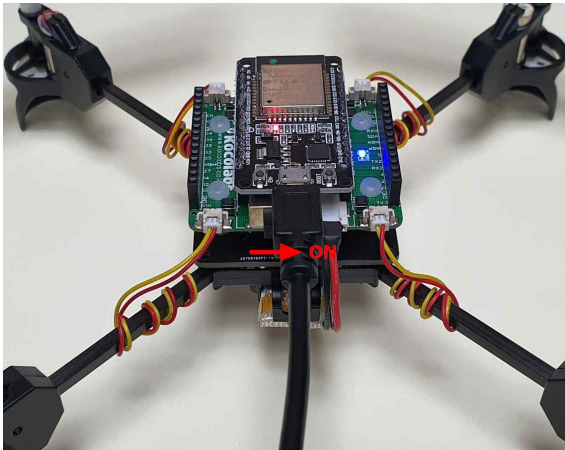
2. 드론의 전원을 끕니다.



3. 업로드를 수행합니다.



4. 17 번 핀에 연결된 프로펠러가 회전 시 손에 닿지 않도록 드론을 주의해서 잡은 후, 전원을 켭니다.



1초 주기로 모터가 돌았다 멈추었다 하는 것을 확인합니다. 즉, 1Hz의 주파수로 모터가 회전하고 정지하는 확인합니다.



모터의 회전은 19번 핀을 통해 나오는 HIGH 값에 의해 발생합니다. 모터의 정지는 19번 핀을 통해 나오는 LOW 값에 의해 발생합니다. 즉, 19번 핀으로는 위 그림과 같이 HIGH값과 LOW 값이 1초 주기로 나오게 되며, 이 값들에 의해 모터는 회전과 정지를 반복하게 됩니다. 그리고 이 경우 여러분은 모터가 돌았다 멈추었다 하는 것을 느낄 수 있습니다.

\*\*\* 모터를 재구동하고 싶다면 다음에 표시된 [EN] 버튼을 눌러 재부팅을 수행합니다.



모터 회전 정지 간격 줄여보기

이제 모터의 회전 정지 간격을 줄여보도록 합니다. 그러면 여러분은 좀 더 조밀하게 모터가 돌다 멈추는 것을 느낄 것입니다.

1. 이전 예제를 다음과 같이 수정합니다.

2510\_2.ino

```
01 const int fan_pin = 17;
02
03 void setup() {
04     pinMode(fan_pin, OUTPUT);
05 }
06
07 unsigned int howMany = 50;
08 void loop() {
09     if(howMany>0) {
10         howMany--;
11
12         digitalWrite(fan_pin, HIGH);
13         delay(10);
14         digitalWrite(fan_pin, LOW);
15         delay(90);
16     }
17 }
```

7 : howMany 변수 값을 50으로 변경합니다.

13 : 100을 10으로 변경합니다.

15 : 900을 90으로 변경합니다.

2. 업로드를 수행합니다.



\*\*\* 업로드가 안 될 경우 USB 단자를 다시 연결해 줍니다.

이 예제의 경우 모터는 초당 10번 돌았다 멈추었다 하게 됩니다. 즉, 10Hz의 주파수로 돌게 됩니다.



그림과 같은 파형이 초당 10개가 생성됩니다. 이 경우에도 여러분은 반복적으로 모터가 돌다 멈추는 것을 느낄 것입니다. 그러나 그 간격은 더 조밀하게 느껴질 것입니다.

## 반복적인 모터 회전 정지를 일정한 회전으로 느껴보기

모터의 회전 정지 간격을 더 줄여보도록 합니다. 여기서 여러분은 모터의 회전 정지를 느끼지 못하게 될 것입니다. 오히려 모터가 일정하게 회전하고 있다고 느낄 것입니다.

1. 이전 예제를 다음과 같이 수정합니다.

2510\_3.ino

```
01 const int fan_pin = 17;
02
03 void setup() {
04     pinMode(fan_pin, OUTPUT);
05 }
06
07 unsigned int howMany = 500;
08 void loop() {
09     if(howMany>0) {
10         howMany--;
11
12         digitalWrite(fan_pin, HIGH);
13         delay(1);
14         digitalWrite(fan_pin, LOW);
15         delay(9);
16     }
17 }
```

7 : howMany 변수 값을 500으로 변경합니다.

13 : 10을 1로 변경합니다.

15 : 90을 9로 변경합니다.

2. 업로드를 수행합니다.



이 예제의 경우 모터는 초당 100번 돌았다 멈추었다 하게 됩니다. 즉, 100Hz의 주파수로 돌게 됩니다.



그림과 같은 파형이 초당 100개가 생성됩니다. 이제 여러분은 모터가 돌아 멈추는 것을 느끼지 못할 것입니다. 오히려 모터가 일정한 속도로 회전한다고 느낄 것입니다.

일반적으로 이러한 파형이 초당 50개 이상이 되면, 즉, 50Hz 이상의 주파수로 모터가 돌고 멈추고를 반복하면 우리는 그것을 느끼기 어렵습니다.



## 03 모터 속도 조절 : ledcWrite

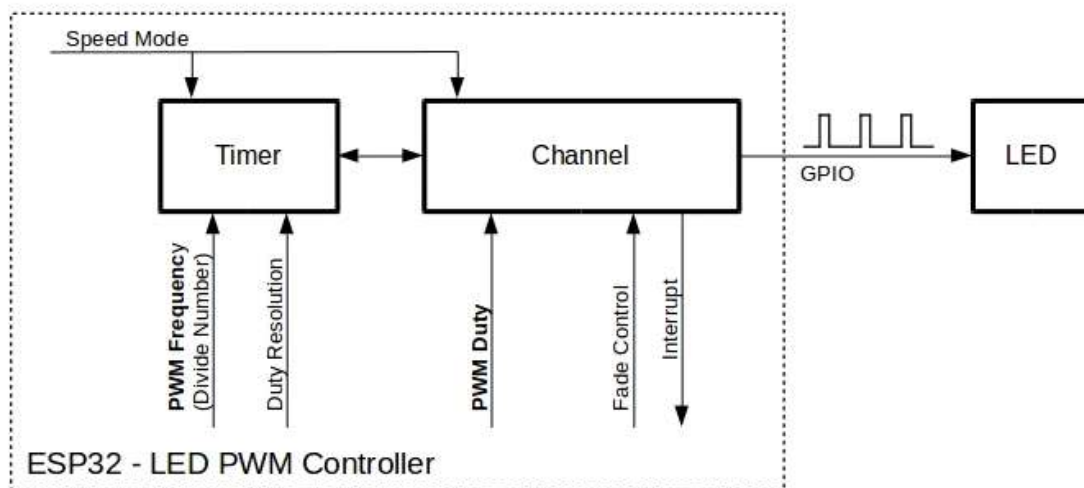
이전 예제에서 우리는 100Hz의 속도로 1:9의 상하비로 모터를 돌려 보았습니다. 17 번 핀에 ledcWrite 함수를 사용할 경우 더 조밀한 상하비로 모터의 속도를 조절할 수 있습니다. 예를 들어, 0~1023 사이의 HIGH 값으로 모터의 속도를 조절할 수 있습니다.

ESP32의 경우 analogWrite 함수 대신에 ledcWrite 함수를 지원합니다. ledcWrite 함수는 사각파형에 대한 세밀한 제어 기능을 제공합니다.

ESP32의 경우 ledcWrite 함수는 모든 핀에 대해 아래와 같은 형태의 사각 파형을 내보내며, 특히 상하비를 결정하는 역할을 합니다. 상하비는 한 주기당 3V 비율을 의미합니다.

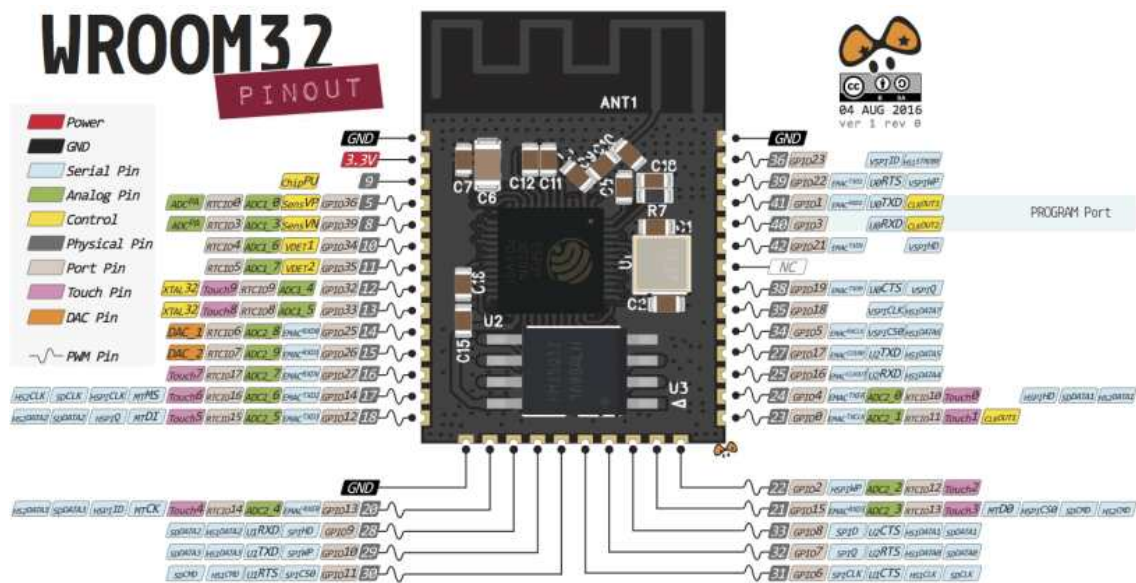


다음 그림은 ESP32 내부의 LED PWM Controller 모듈입니다.



ledcWrite 함수는 내부 LED PWM Controller 모듈에 명령을 주어 해당 핀으로 일정한 모양의 사각파형을 내보내게 합니다. LED PWM Controller 모듈은 새로운 ledcWrite 명령을 받을 때까지 해당 핀으로 똑같은 사각 파형을 내보냅니다.

다음 그림에서 물결 표시된 핀은 사각 파형을 내보낼 수 있는 핀입니다.



여러분은 아두이노 스케치를 통해 LED의 밝기를 조절하거나 모터의 속도를 조절할 때 다음 세 함수를 사용하게 됩니다.

```
ledcAttachPin(pin, channel)
ledcSetup(channel, frequency, resolution)
ledcWrite(channel, dutycycle)
```

## ledcAttachPin

ledcAttachPin이란 특정 핀으로 사각파형을 내보낼 채널을 연결하는 명령어입니다.

**ledcAttachPin**(pin, channel);

①      ②

① 제어하고자 하는 핀 번호입니다. 정수형을 씁니다.

② 사각 파형을 내보낼 채널 번호입니다. 0~15 사이의 값을 쓸 수 있습니다. 정수형을 씁니다.

## ledcSetup

ledcSetup이란 특정 채널에 주파수, 듀티 사이클 정밀도를 설정하는 명령어입니다.

**ledcSetup**(channel, frequency, resolution);

①      ②      ③

① 사각 파형을 내보낼 채널 번호입니다. 정수형을 씁니다.

② 사각 파형의 주파수입니다. 정수형을 씁니다.

③ 사각 파형의 듀티사이클의 정밀도입니다. 예를 들어, resolution 값이 10 이면 듀티사이클 값은 0~1023 ( $2^{10} - 1$ )사이의 값을 갖습니다.

## ledcWrite

ledcWrite란 특정 채널에 듀티사이클을 적용하는 명령어입니다.

**ledcWrite(channel, dutycycle);**

①

②

① 사각 파형을 내보낼 채널 번호입니다. 0~16 사이의 값을 쓸 수 있습니다. 정수형을 씁니다.

② 듀티 사이클 값입니다. ledcSetup 함수에서 설정한 해상도 인자에 따라 결정됩니다. 예를 들어, resolution 값이 10 이면 듀티사이클 값은 0~1023( $2^{10} - 1$ )사이의 값을 갖습니다.

## 01 ledcWrite 함수로 모터 회전 정지 반복해 보기

먼저 ledWrite 함수로 모터를 돌렸다 멈췄다를 반복해 봅니다.

1. 다음과 같이 예제를 작성합니다.

262.ino

```
01 const int fan_pin = 17;
02 const int fan_channel = 1;
03 const int fan_freq = 1;
04 const int fan_resolution = 10;
05
06 void setup() {
07     ledcAttachPin(fan_pin, fan_channel);
08     ledcSetup(fan_channel, fan_freq, fan_resolution);
09
10     ledcWrite(fan_channel, 100);
11
12     delay(5000);
13
14     ledcWrite(fan_channel, 0);
15 }
16
17 void loop() {
18
19 }
```

1 : fan\_pin 정수 상수를 선언한 후, 19로 설정합니다.

2 : fan\_channel 정수 상수를 선언한 후, 1로 설정합니다.

3 : fan\_freq 정수 상수를 선언한 후, 1로 설정하여 주파수를 1로 맞추어 줍니다.

4 : fan\_resolution 정수 상수를 선언한 후, 10으로 설정합니다.

7 : ledcAttachPin 함수를 호출하여 핀에 채널을 연결합니다.

8 : ledcSetup 함수를 호출하여 fan\_channel에 주파수와 듀티 사이클 해상도를

설정합니다.

10 : ledcWrite 함수를 호출하여 fan\_channel에 100의 HIGH 값을 줍니다. 이렇게 하면 10%의 속도로 모터가 회전합니다.

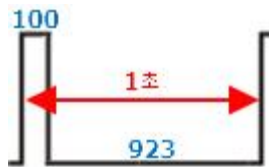
12 : 5초간 지연을 줍니다.

14 : ledcWrite 함수를 호출하여 fan\_channel에 0의 HIGH 값을 줍니다. 이렇게 하면 모터의 회전이 멈춥니다.

2. 업로드를 수행합니다.



1초 주기로 모터가 돌았다 멈추었다 하는 것을 확인합니다. 즉, 1Hz의 주파수로 모터가 회전하고 정지하는 확인합니다.



## 모터 회전 정지 간격 줄여보기

이제 모터의 회전 정지 간격을 줄여보도록 합니다. 그러면 여러분은 좀 더 조밀하게 모터가 돌다 멈추는 것을 느낄 것입니다.

1. 이전 예제를 다음과 같이 수정합니다.

262\_2.ino

```
01 const int fan_pin = 17;
02 const int fan_channel = 1;
03 const int fan_freq = 10;
04 const int fan_resolution = 10;
05
06 void setup() {
07     ledcAttachPin(fan_pin, fan_channel);
08     ledcSetup(fan_channel, fan_freq, fan_resolution);
09
10     ledcWrite(fan_channel, 100);
11
12     delay(5000);
13
14     ledcWrite(fan_channel, 0);
15 }
```

```

16
17 void loop() {
18
19 }

```

3 : fan\_freq 정수 상수 값을 10으로 설정하여 주파수를 10으로 맞추어 주고 있습니다. 이 경우 주파수는 10Hz가 됩니다.

2. 업로드를 수행합니다.



\*\*\* 업로드가 수행이 안 될 경우, ESP32를 분리한 후, 업로드를 수행해 봅니다.

이 예제의 경우 모터는 초당 10번 돌았다 멈추었다 하게 됩니다. 즉, 10Hz의 주파수로 돌게 됩니다.



## 반복적인 모터 회전 정지를 일정한 회전으로 느껴보기

모터의 회전 정지 간격을 더 줄여보도록 합니다. 여기서 여러분은 모터의 회전 정지를 느끼지 못하게 될 것입니다. 오히려 모터가 일정하게 회전하고 있다고 느낄 것입니다.

1. 이전 예제를 다음과 같이 수정합니다.

262\_3.ino

```

01 const int fan_pin = 17;
02 const int fan_channel = 1;
03 const int fan_freq = 100;
04 const int fan_resolution = 10;
05
06 void setup() {
07     ledcAttachPin(fan_pin, fan_channel);
08     ledcSetup(fan_channel, fan_freq, fan_resolution);
09
10     ledcWrite(fan_channel, 100);
11

```

```

12     delay(5000);
13
14     ledcWrite(fan_channel, 0);
15 }
16
17 void loop() {
18
19 }

```

3 : fan\_freq 정수 상수 값을 100으로 설정하여 주파수를 100으로 맞추어 주고 있습니다. 이 경우 주파수는 100Hz가 됩니다.

2. 업로드를 수행합니다.



이 예제의 경우 모터는 초당 100번 돌았다 멈추었다 하게 됩니다. 즉, 100Hz의 주파수로 돌게 됩니다.



그림과 같은 파형이 초당 100개가 생성됩니다. 이제 여러분은 모터가 돌다 멈추는 것을 느끼지 못할 것입니다. 오히려 모터가 일정한 속도로 회전한다고 느낄 것입니다.

## 모터 회전 부드럽게 만들기

주파수를 늘리면 모터의 회전이 더 부드러워집니다. 여기서는 주파수를 늘려 모터 회전을 좀 더 부드럽게 만들어 봅니다.

1. 이전 예제를 다음과 같이 수정합니다.

262\_4.ino

```

01 const int fan_pin = 17;
02 const int fan_channel = 1;
03 const int fan_freq = 1000;
04 const int fan_resolution = 10;
05
06 void setup() {

```

```

07    ledcAttachPin(fan_pin, fan_channel);
08    ledcSetup(fan_channel, fan_freq, fan_resolution);
09
10    ledcWrite(fan_channel, 100);
11
12    delay(5000);
13
14    ledcWrite(fan_channel, 0);
15 }
16
17 void loop() {
18
19 }

```

3 : fan\_freq 정수 상수 값을 1000으로 설정하여 주파수를 1000으로 맞추어 주고 있습니다. 이 경우 주파수는 1000Hz가 됩니다.

2. 업로드를 수행합니다.



이 예제의 경우 모터는 초당 1000번 돌았다 멈추었다 하게 됩니다. 즉, 1000Hz의 주파수로 돌게 됩니다.



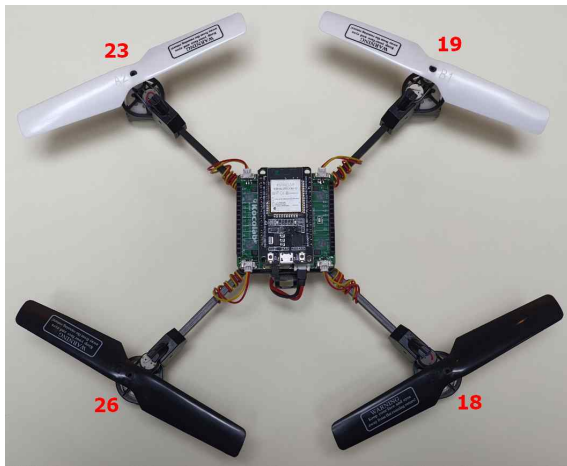
모터의 회전이 훨씬 부드러운 것을 느낄 수 있습니다.

### 03 모터 멜로디 조절해 보기

다음은 소리에 따른 주파수 표를 나타냅니다. 예를 들어 4 옥타브에서 도 음에 대한 주파수는 262 Hz가 됩니다. 즉, 1초에 262 개의 사각 파형을 만들어 내면 도 음이 나게 됩니다. 레는 294 Hz, 미는 330 Hz, 파는 349 Hz, 솔은 392 Hz, 라는 440 Hz, 시는 494 Hz, 5 옥타브의 도는 523 Hz가 됩니다.

Frequency in hertz (semitones above or below middle C)												
Octave → Note ↓	0	1	2	3	4	5	6	7	8	9	10	
C	16.352 (-48)	32.703 (-36)	65.406 (-24)	130.81 (-12)	261.63 (±0)	523.25 (+12)	1046.5 (+24)	2093.0 (+36)	4186.0 (+48)	8372.0 (+60)	16744.0 (+72)	
C#/D♭	17.324 (-47)	34.648 (-35)	69.296 (-23)	138.59 (-11)	277.18 (+1)	554.37 (+13)	1108.7 (+25)	2217.5 (+37)	4434.9 (+49)	8869.8 (+61)	17739.7 (+73)	
D	18.354 (-46)	36.708 (-34)	73.416 (-22)	146.83 (-10)	293.66 (+2)	587.33 (+14)	1174.7 (+26)	2349.3 (+38)	4698.6 (+50)	9397.3 (+62)	18794.5 (+74)	
D#/E♭	19.445 (-45)	38.891 (-33)	77.782 (-21)	155.56 (-9)	311.13 (+3)	622.25 (+15)	1244.5 (+27)	2489.0 (+39)	4978.0 (+51)	9956.1 (+63)	19912.1 (+75)	
E	20.602 (-44)	41.203 (-32)	82.407 (-20)	164.81 (-8)	329.63 (+4)	659.26 (+16)	1318.5 (+28)	2637.0 (+40)	5274.0 (+52)	10548.1 (+64)	21096.2 (+76)	
F	21.827 (-43)	43.654 (-31)	87.307 (-19)	174.61 (-7)	349.23 (+5)	698.46 (+17)	1396.9 (+29)	2793.8 (+41)	5587.7 (+53)	11175.3 (+65)	22350.6 (+77)	
F#/G♭	23.125 (-42)	46.249 (-30)	92.499 (-18)	185.00 (-6)	369.99 (+6)	739.99 (+18)	1480.0 (+30)	2960.0 (+42)	5919.9 (+54)	11839.8 (+66)	23679.6 (+78)	
G	24.500 (-41)	48.999 (-29)	97.999 (-17)	196.00 (-5)	392.00 (+7)	783.99 (+19)	1568.0 (+31)	3136.0 (+43)	6271.9 (+55)	12543.9 (+67)	25087.7 (+79)	
A♭/G#	25.957 (-40)	51.913 (-28)	103.83 (-16)	207.65 (-4)	415.30 (+8)	830.61 (+20)	1661.2 (+32)	3322.4 (+44)	6644.9 (+56)	13289.8 (+68)	26579.5 (+80)	
A	27.500 (-39)	55.000 (-27)	110.00 (-15)	220.00 (-3)	440.00 (+9)	880.00 (+21)	1760.0 (+33)	3520.0 (+45)	7040.0 (+57)	14080.0 (+69)	28160.0 (+81)	
B♭/A#	29.135 (-38)	58.270 (-26)	116.54 (-14)	233.08 (-2)	466.16 (+10)	932.33 (+22)	1864.7 (+34)	3729.3 (+46)	7458.6 (+58)	14917.2 (+70)	29834.5 (+82)	
B	30.868 (-37)	61.735 (-25)	123.47 (-13)	246.94 (-1)	493.88 (+11)	987.77 (+23)	1975.5 (+35)	3951.1 (+47)	7902.1 (+59)	15804.3 (+71)	31608.5 (+83)	

여기서는 모터를 이용하여 멜로디를 생성해 보도록 하겠습니다. 드론의 19번 모터를 이용해 멜로디를 생성해 보도록 합니다.



1. 다음과 같이 예제를 작성합니다.

263.ino

```

01 const int fan_pin = 17;
02 const int fan_channel = 1;
03 const int fan_resolution = 10;
04
05 const int melody[] = {
06     262, 294, 330, 349, 393, 440, 494, 523,
07 };
08
09 void setup() {
10     ledcAttachPin(fan_pin, fan_channel);
11
12     for(int note=0;note<8;note++) {
13         ledcSetup(fan_channel, melody[note], fan_resolution);
14         ledcWrite(fan_channel, 10);

```



```

15         delay(500);
16
17         ledcWrite(fan_channel, 0);
18         delay(50);
19     }
20
21     ledcWrite(fan_channel, 0);
22 }
23
24 void loop() {
25
26 }

```

5~7 : 4 옥타브의 도, 레, 미, 파, 솔, 라, 시, 도에 해당하는 주파수를 값으로 갖는 melody 배열 상수를 선언합니다.

10 : ledcAttachPin 함수를 호출하여 핀에 채널을 연결합니다.

13 : ledcSetup 함수를 호출하여 fan\_channel에 주파수와 듀티 사이클 해상도를 설정합니다.

14 : ledcWrite 함수를 호출하여 fan\_channel에 10의 HIGH 값을 줍니다. 이렇게 하면 모터는 거의 회전하지 않으면서, 소리만 납니다.

15 : 500밀리 초간 기다립니다.

17 : ledcWrite 함수를 호출하여 fan\_channel에 0의 HIGH 값을 줍니다. 이렇게 하면 소리가 꺼집니다.

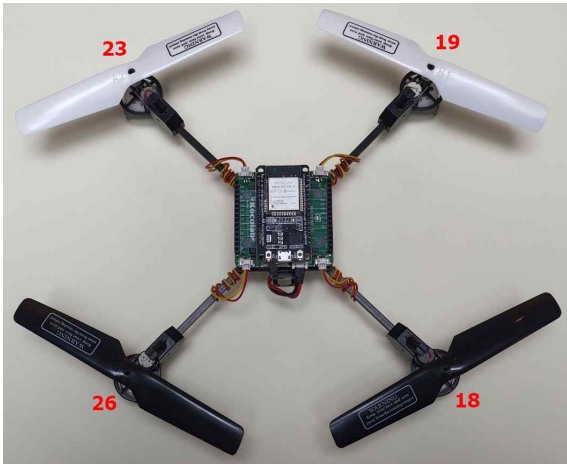
18 : 50밀리 초간 기다립니다.

21 : ledcWrite 함수를 호출하여 fan\_channel에 0의 HIGH 값을 줍니다. 이렇게 하면 소리가 꺼집니다.

2. 업로드를 수행합니다.



드론의 전원을 켜 후, 17 번 모터에서 나는 멜로디를 확인합니다.



## 04 드론 모터의 이해와 테스트

여기서는 드론 모터를 살펴보고 모터 테스트 프로그램을 작성해봅니다. 또 사용자로부터 입력을 받아 모터의 속도를 조절해 봅니다.

### 01 드론 모터의 구조 이해



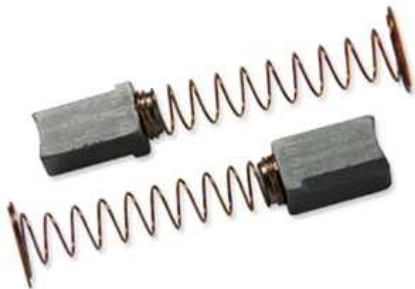
일반적으로 드론 용 모터로는 BLDC 모터가 사용됩니다. BLDC(BrushLess DC) 모터는 DC 모터의 일종으로 브러시 없는(Brushless) 모터입니다.

### 일반 DC 모터의 구조

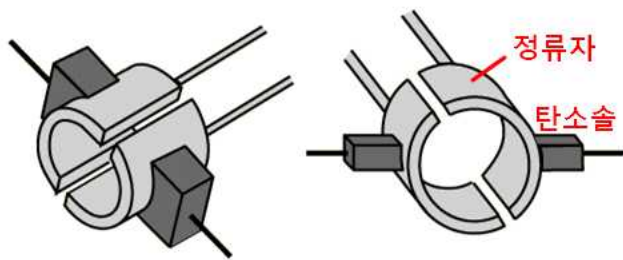
브러시가 있는 일반 DC 모터는 다음과 같은 모양입니다.



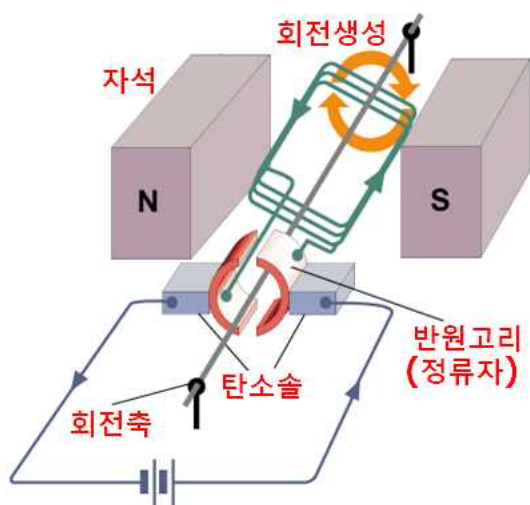
일반 DC 모터는 모터를 구동시키기 위해 다음과 같은 형태의 브러시가 사용됩니다.



이 브러시는 다음과 같은 형태로 정류자(commutator)를 통해 코일과 연결됩니다.



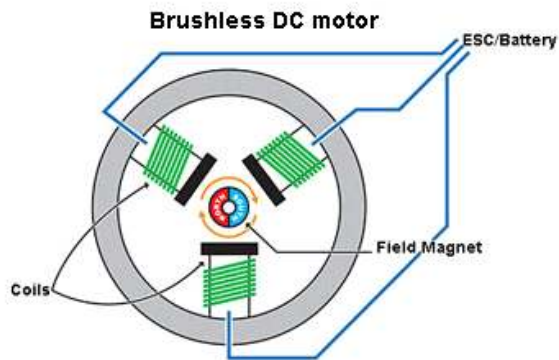
즉, 다음 그림과 같이 전지로부터의 전류가 카본 브러시와 정류자를 통해 코일로 전류가 흐르면서 회전을 하게 됩니다.



이 과정에서 브러시와 정류자 간에 마찰과 열이 발생하게 됩니다. 그래서 브러시가 있는 일반 DC모터는 이러한 마찰과 열에 의해 모터 효율이 60% 내외가 됩니다. 또 브러시의 마모에 의해 모터의 수명도 짧아지게 됩니다.

## BLDC 모터의 구조

이러한 단점을 극복하기 위해 BLDC 모터는 브러시를 사용하지 않습니다. BLDC 모터는 다음과 같은 구조입니다.

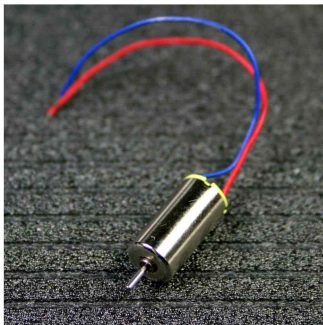


BLDC 모터는 축을 돌리기 위해 코일과 자석이 사용된다는 점에서는 일반 DC 모터와 같습니다. 그러나 BLDC 모터는 회전축에 연결되어 코일 내의 전원의 방향을 바꾸는 역할을 하는 브러시가 없습니다. 대신에 BLDC 모터는 모터의 내부 주변에 코일이 원통에 고정되어 있습니다. 중앙에는 회전축에 붙어있는 원통이 있고, 이 원통에 자석이 붙어있는 구조입니다. BLDC 모터는 BLDC 용 모터 컨트롤러로 구동하여야 하지만, 수명이 길고 마찰이 적어 우주 항공 분야, 의료 분야, 반도체, 측정기, 로봇 등 정밀제어분야에 주로 사용됩니다. BLDC 모터는 효율이 80% 이상입니다.

BLDC 모터에 대한 자세한 내용은 이 책에서는 다루지 않습니다. 본 책에서는 안전 문제상 BLDC 모터를 사용하지 않고 Coreless 모터를 사용합니다.

## CLDC 모터의 구조

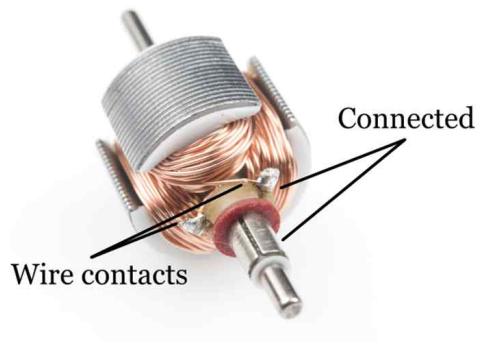
Coreless 모터는 다음과 같은 모양의 소형 모터입니다.



Coreless 모터의 내부 구조는 다음과 같습니다.



Coreless 모터는 브러시를 사용하는 DC 모터의 한 종류이지만 구리선이 감겨있는 철심이 없습니다. 즉, 일반 DC 모터의 내부에 코일은 다음과 같이 철심에 감겨져 있습니다.



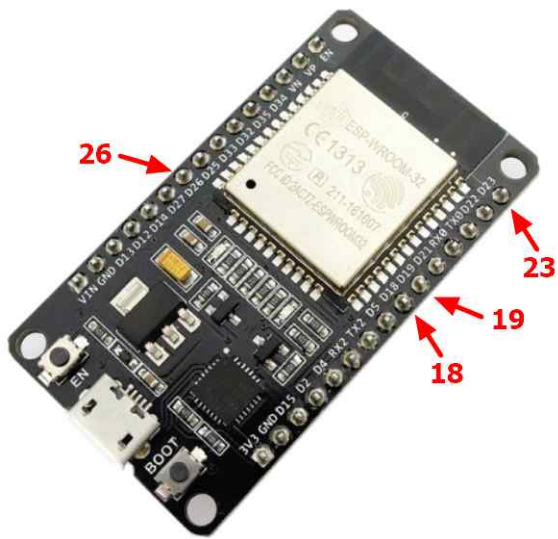
구리선을 벗겨낸 철심의 모양은 다음과 같습니다.



Coreless 모터는 모터 내부에 철심이 없기 때문에 더 가볍고 작게 만들 수 있습니다. Coreless 모터는 주로 의료 기기, 우주 항공, 자동차, 해저 탐사용 로봇들에 사용됩니다. Coreless 모터의 효율은 보통 70% ~ 80%입니다.

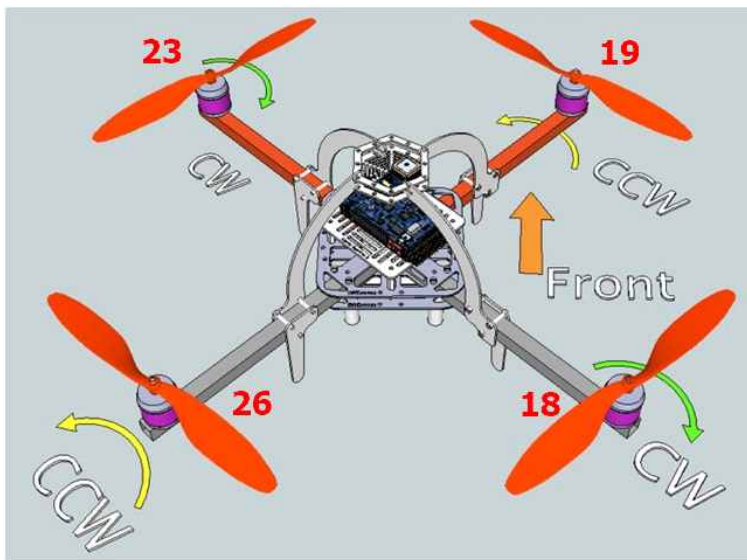
## 02 드론 모터 회로 살펴보기

본 책에서 다루고 있는 KocoNode 드론 모터는 ESP32의 23, 19, 18, 26 번 핀에 연결되어 있습니다.

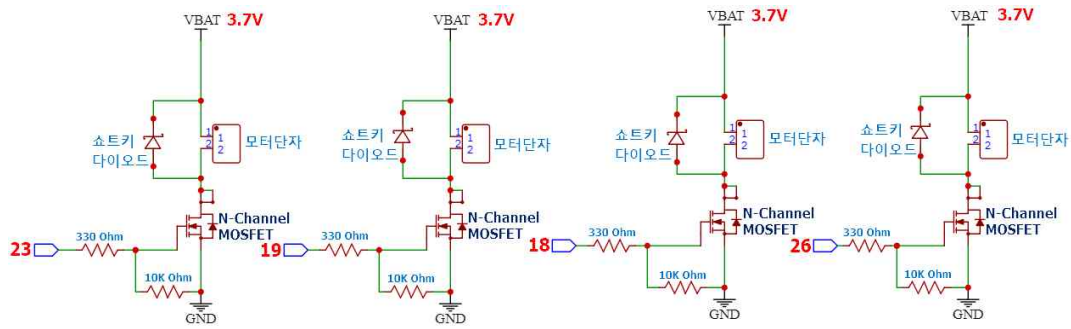


16, 17, 25, 26

다음과 같이 23, 18 번 핀에 연결된 모터는 시계 방향, 19, 26 번 핀에 연결된 모터는 반시계 방향으로 돌게 됩니다.



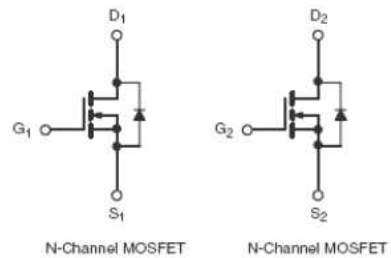
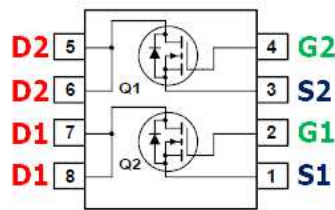
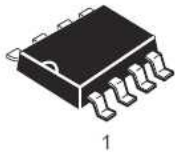
아두이노의 모터를 제어하는 핀은 모터와 직접 연결되지 않습니다. 일반적으로 모터를 제어하기 위해서는 모터 회로가 필요합니다. 본 책에서 다루는 드론의 경우는 다음과 같은 형태의 모터 회로를 가지고 있습니다.



모터 제어 핀은 23, 19, 18, 26을 통해 MOSFET을 통해 모터에 연결됩니다. MOSFET은 트랜지스터의 일종으로 전자 스위치입니다. 그림에서 23, 19, 18, 26 핀을 통해 ESP32에서 HIGH 값을 주면 모터 회로가 연결되어 모터가 회전하며, LOW 값을 주면 모터 회로가 끊기며 모터가 멈추게 됩니다.

이 책에서 사용하는 드론 모터 드라이버는 9926A로 모양은 다음과 같습니다.

# 9926A



## 03 모터 제어 프로그램 작성하기

여기서는 ESP32의 ledcWrite 함수를 이용하여 4개의 모터를 제어해보도록 합니다. ledcWrite 함수는 칩 내부에 있는 Timer 모듈에 명령을 주어 아래와 같은 형태의 사각 파형을 내보낼 수 있습니다.



## 모터 돌려 보기

먼저 아두이노 스케치를 이용하여 4 개의 모터를 차례대로 돌려 보도록 하겠습니다. 모터가 도는 방향에 대해 자세히 살펴보도록 합니다.

1. 다음과 같이 예제를 작성합니다.

273.ino



```
01 const int MOTOR_A = 16;
02 const int MOTOR_B = 17;
03 const int MOTOR_C = 25;
04 const int MOTOR_D = 26;
05 const int CHANNEL_A = 0;
06 const int CHANNEL_B = 1;
07 const int CHANNEL_C = 2;
08 const int CHANNEL_D = 3;
09 const int MOTOR_FREQ = 5000;
10 const int MOTOR_RESOLUTION = 10;
11
12 void setup() {
13     ledcAttachPin(MOTOR_A, CHANNEL_A);
14     ledcAttachPin(MOTOR_B, CHANNEL_B);
15     ledcAttachPin(MOTOR_C, CHANNEL_C);
16     ledcAttachPin(MOTOR_D, CHANNEL_D);
17
18     ledcSetup(CHANNEL_A, MOTOR_FREQ, MOTOR_RESOLUTION);
19     ledcSetup(CHANNEL_B, MOTOR_FREQ, MOTOR_RESOLUTION);
20     ledcSetup(CHANNEL_C, MOTOR_FREQ, MOTOR_RESOLUTION);
21     ledcSetup(CHANNEL_D, MOTOR_FREQ, MOTOR_RESOLUTION);
22
23     ledcWrite(CHANNEL_A, 0);
24     ledcWrite(CHANNEL_B, 0);
25     ledcWrite(CHANNEL_C, 0);
26     ledcWrite(CHANNEL_D, 0);
27
28     delay(3000);
29 }
30
31 unsigned int howMany = 3;
32 void loop() {
33
34     if(howMany > 0) {
35         howMany--;
36
37         ledcWrite(CHANNEL_A, 100); delay(1000);
38         ledcWrite(CHANNEL_B, 100); delay(1000);
39         ledcWrite(CHANNEL_C, 100); delay(1000);
40         ledcWrite(CHANNEL_D, 100); delay(1000);
```

```

41
42         ledcWrite(CHANNEL_A, 0);
43         ledcWrite(CHANNEL_B, 0);
44         ledcWrite(CHANNEL_C, 0);
45         ledcWrite(CHANNEL_D, 0);
46         delay(4000);
47     }
48
49 }

```

1~4 : 모터 A, B, C, D에 연결된 핀 상수를 선언합니다.

5~8 : 모터 A, B, C, D 핀에 연결한 채널 상수를 선언합니다.

9 : 모터 주파수 값을 저장할 주파수 상수를 선언합니다.

10 : 모터 듀티 사이클 해상도 값을 저장할 상수를 선언합니다.

13~16 : ledcAttachPin 함수를 호출하여 모터 A, B, C, D 핀에 채널을 연결합니다.

18~21 : ledcSetup 함수를 호출하여 모터 A, B, C, D 채널에 주파수와 듀티 사이클 해상도를 설정합니다.

23~26 : ledcWrite 함수를 호출하여 채널 A, B, C, D에 0값을 줍니다. 즉, 4개의 모터를 멈춥니다.

28 : 3초간 기다립니다. 이 부분은 실습 시 안전을 위해 넣은 부분입니다.

31 : howMany 변수를 선언한 후, 3으로 초기화합니다. howMany 변수는 34, 35줄에서 사용하여 34~47줄의 수행 횟수를 결정합니다.

37~40 : ledcWrite 함수를 호출하여 채널 A, B, C, D에 100 값을 줍니다. 1023 중 100에 해당하는 값만큼 HIGH가 (1023-100)에 해당하는 값만큼 LOW가 출력되게 됩니다. 그리고 1초간 지연을 줍니다.

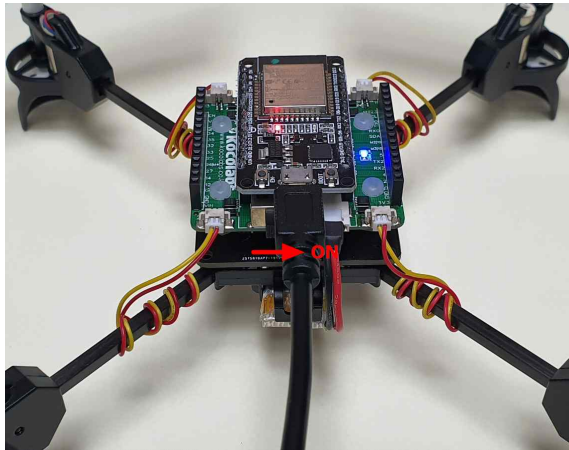
42~45 : ledcWrite 함수를 호출하여 채널 A, B, C, D에 0값을 줍니다. 즉, 4개의 모터를 멈춥니다.

46 : 4초간 기다립니다.

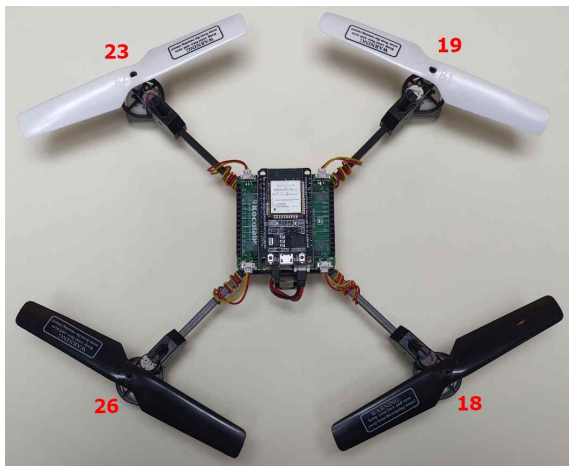
2. 업로드를 수행합니다.



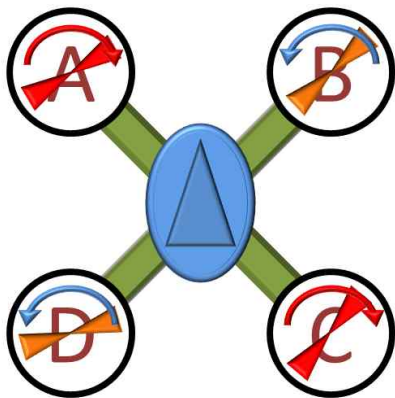
3. USB에 연결된 상태로 배터리 전원을 켭니다. 드론 모터는 배터리 전원으로 동작하도록 회로 구성이 되어 있습니다.



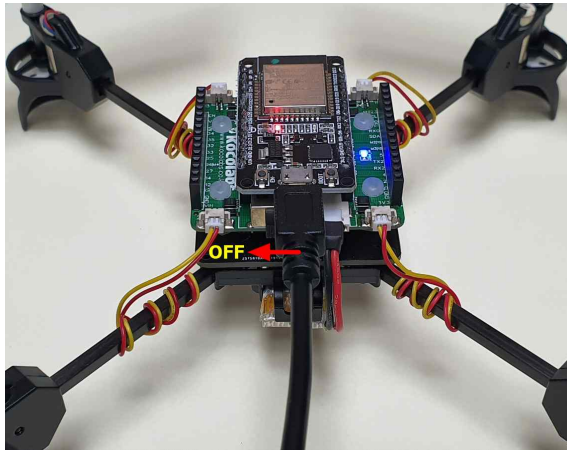
4. 23, 19, 18, 26의 순서대로 4 개의 모터가 1초 간격으로 차례대로 돌아가는 것을 확인합니다. 그리고 4 초간 멈추는 것을 확인합니다. 이 동작을 3회 반복합니다.



모터의 회전 방향은 다음과 같습니다.



5. 테스트가 끝났으면 모터 전원을 끕니다.



\*\*\* 외부 전원을 사용하여 4개의 모터를 동시에 구동할 경우에 ESP32로 공급되는 전력이 순간적으로 부족하여 brownout 리셋이 발생할 수 있습니다. ESP32의 경우 내부적으로 WiFi와 BLE 하드웨어 모듈을 내장하고 있으며, 이러한 하드웨어 모듈들은 안정적인 전력 공급을 필요로 합니다. 따라서 ESP32 칩 내부에는 Brownout Detector 기능이 있어 전원 전압이 내부적으로 설정한 전압값 이하로 떨어지면 자동으로 리셋을 발생시켜 ESP32를 재부팅하게 됩니다. 외부 전원만으로 ESP32를 구동할 경우에, 특히 모터와 같이 순간적으로 구동 전력을 많이 사용하는 하드웨어와 같이 구동할 경우에 ESP32에 brownout 리셋이 발생할 가능성이 높습니다. 이 경우, ESP32로 전원을 따로 공급하거나 또는 다음과 같이 Brownout Detector를 비활성화시켜 문제를 해결합니다.

```
01 #include "soc/soc.h"
02 #include "soc/rtc_cntl_reg.h"
03
04 void setup() {
05     WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
06     // 여러분의 코드를 작성합니다.
07 }
08
09 void loop() {
10     // 여러분의 코드를 작성합니다.
11 }
```

01 : soc/soc.h 파일을 포함합니다. soc.h 파일은 05줄에서 호출하는 함수 WRITE\_PERI\_REG에 대한 정보를 담고 있습니다.

02 : soc/rtc\_cntl\_reg.h 파일을 포함합니다. rtc\_cntl\_reg.h 파일은 05줄에서 사용하는 레지스터 RTC\_CNTL\_BROWN\_OUT\_REG에 대한 정보를 담고 있습니다.

05 : WRITE\_PERI\_REG 함수를 호출하여 RTC\_CNTL\_BROWN\_OUT\_REG 레지스터 변수에 0을 써 줍니다. 이렇게 하면 Brownout Detector를 비활성화시켜 brownout 리셋을 막게 됩니다.

## 05 사용자 입력 받기

우리는 앞에서 Serial.println 함수를 통해서 아두이노의 이야기를 듣는 방법을 살펴보았습니다. 그러면 아두이노가 우리의 이야기를 듣는 방법을 없을까요? 아두이노가 우리의 이야기를 들어야 우리가 원하는 것을 아두이노에게 시킬 수 있지 않을까요? 이 때 필요한 함수가 바로 Serial.read 함수입니다.

여기서는 시리얼 입력을 살펴봅니다. 시리얼 입력은 사용자의 입력을 받기 위해 필요하며 Serial.available 함수와 Serial.read 함수를 이용합니다. 사용자의 입력을 받기 때문에 아주 중요한 기능입니다.

ESP32는 USB 단자를 통해 PC로부터 메시지를 받습니다.

여러분은 아두이노 스케치를 통해 아두이노가 PC로부터 메시지를 받게 할 때 다음 세 함수를 주로 사용하게 됩니다.

```
Serial.begin(speed)
Serial.available()
Serial.read()
```

Serial.begin 함수는 앞에서 이미 살펴보았습니다. 여기서는 따로 설명하지 않습니다. Serial.available은 PC로부터 도착한 데이터의 바이트 수를 돌려줍니다. Serial.read는 PC로부터 받은 메시지의 첫 번째 바이트를 읽는 함수입니다.

## 01 사용자 입력 받기

여기서는 Serial.available 함수와 Serial.read 함수를 이용하여 PC를 통해 사용자로부터 문자를 입력받은 후, PC로 사용자 입력을 돌려보내 봅니다.

1. 다음과 같이 예제를 작성합니다.

281.ino

```
01 void setup() {
02     Serial.begin(115200);
03 }
04
05 void loop() {
06     if(Serial.available() > 0) {
07         char userInput = Serial.read();
08         Serial.print(userInput);
```

```
09     }  
10 }
```

6 : Serial.available 함수를 호출하여 시리얼을 통해 도착한 문자가 있는지 확인합니다. 도착한 문자가 있을 경우 6~9줄을 수행합니다. Serial.available 함수는 시리얼 입력 버퍼에 도착한 데이터의 개수를 주는 함수입니다.

7 : Serial.read 함수를 호출하여 키보드 입력 문자 하나를 userInput 변수로 받습니다. Serial.read 함수는 시리얼 입력 버퍼에 도착한 데이터를 한 바이트 읽어내는 함수입니다.

8 : Serial.print 함수를 호출하여 사용자로부터 전달된 문자를 출력합니다.

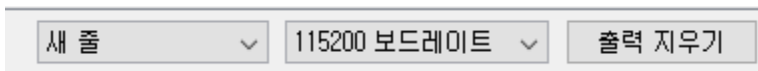
2. 업로드를 수행합니다.



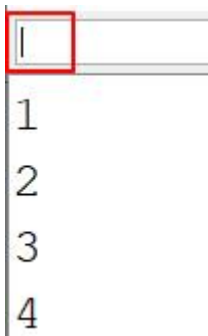
3. [시리얼 모니터] 버튼을 눌러줍니다.



4. 시리얼 모니터 창이 뜨면, 우측 하단에서 통신 속도를 115200으로 맞춰줍니다.



5. 시리얼 모니터 창의 빨간 박스 입력 창에 1, 2, 3, 4를 입력해 봅니다.



1,2,3,4 문자가 표시되는 것을 확인합니다.

## 02 모터 속도 조절해 보기

여기서는 시리얼을 이용하여 모터의 속도를 조절해 봅니다.

1. 다음과 같이 예제를 작성합니다. 263.ino, 271.ino 파일을 복사하여 편집합니다.  
282.ino

```
01 const int MOTOR_A = 16;
02 const int MOTOR_B = 17;
03 const int MOTOR_C = 25;
04 const int MOTOR_D = 26;
05 const int CHANNEL_A = 0;
06 const int CHANNEL_B = 1;
07 const int CHANNEL_C = 2;
08 const int CHANNEL_D = 3;
09 const int MOTOR_FREQ = 5000;
10 const int MOTOR_RESOLUTION = 10;
11
12 void setup() {
13     ledcAttachPin(MOTOR_A, CHANNEL_A);
14     ledcAttachPin(MOTOR_B, CHANNEL_B);
15     ledcAttachPin(MOTOR_C, CHANNEL_C);
16     ledcAttachPin(MOTOR_D, CHANNEL_D);
17
18     ledcSetup(CHANNEL_A, MOTOR_FREQ, MOTOR_RESOLUTION);
19     ledcSetup(CHANNEL_B, MOTOR_FREQ, MOTOR_RESOLUTION);
20     ledcSetup(CHANNEL_C, MOTOR_FREQ, MOTOR_RESOLUTION);
21     ledcSetup(CHANNEL_D, MOTOR_FREQ, MOTOR_RESOLUTION);
22
23     ledcWrite(CHANNEL_A, 0);
24     ledcWrite(CHANNEL_B, 0);
25     ledcWrite(CHANNEL_C, 0);
26     ledcWrite(CHANNEL_D, 0);
27
28     Serial.begin(115200);
29 }
30
31 void loop() {
32     if(Serial.available() > 0) {
33         char userInput = Serial.read();
34         Serial.println(userInput);
35
36         if(userInput >= '0' && userInput <= '9') {
37             int throttle = (userInput - '0') * 40;
38             ledcWrite(CHANNEL_A, throttle);
39             ledcWrite(CHANNEL_B, throttle);
40             ledcWrite(CHANNEL_C, throttle);
```

```

41         ledcWrite(CHANNEL_D, throttle);
42     }
43 }
44 }

```

36 : 사용자 입력 값이 '0'(0 문자)보다 크거나 같고 '9'(9 문자) 값보다 작으면

37 : 사용자 입력 값에서 '0' 문자 값을 빼서 숫자 값을 만든 후, 40을 곱해서 throttle 변수 값에 할당합니다. throttle 변수는 각 모터에 적용되는 속도 값을 저장하는 변수입니다.

\*\*\* '0'~'9' 문자에 대응되는 아스키 숫자 값은 48~57입니다. 그래서 사용자가 '3'문자를 입력할 경우 '3'-'0'=51-48=3이 됩니다. 이 3의 값에 40을 곱하면 120이 되며 이 값을 ledcWrite 함수의 두 번째 인자로 넣게 됩니다. 이 예제에서는 모터의 최대 속도로 360까지 줄 수 있습니다. 최대 속도는 1023을 주었을 때입니다.

38~41 : ledcWrite 함수를 호출하여 16, 17, 25, 26 번 핀에 throttle 값을 줍니다.

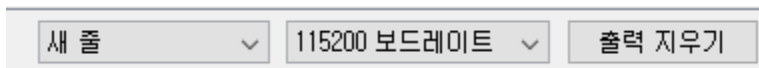
2. 업로드를 수행합니다.



3. [시리얼 모니터] 버튼을 눌러줍니다.

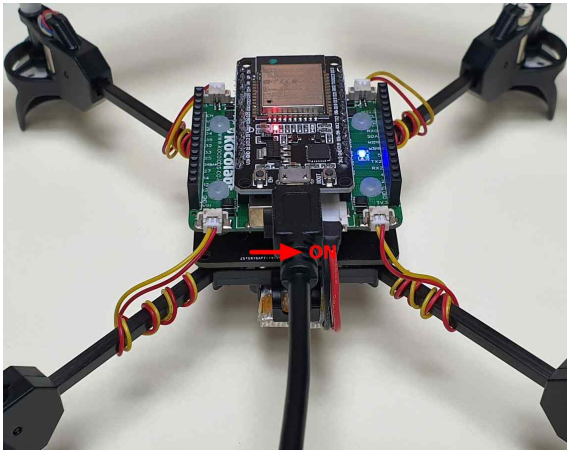


4. 시리얼 모니터 창이 뜨면, 우측 하단에서 통신 속도를 115200으로 맞춰줍니다.



5. USB에 연결된 상태로 배터리 전원을 켭니다. 드론 모터는 배터리 전원으로 동작하도록 회로 구성이 되어 있습니다.

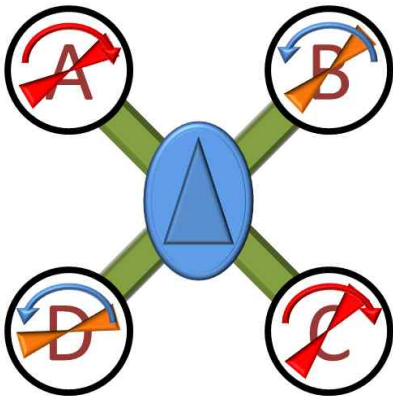




6. 시리얼 모니터 창의 빨간 박스 입력 창에 1, 2, 3, 4를 입력해 봅니다. 9까지 입력할 수 있습니다.

1  
2  
3  
4

7. 드론의 프로펠러가 회전하는 것을 확인합니다.



8. 테스트가 끝났으면 모터 전원을 끕니다.

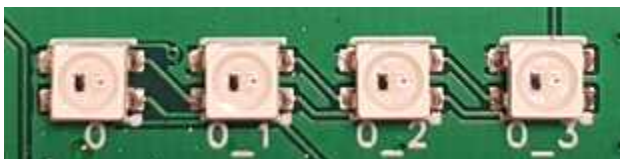


## 06 RGB 네오픽셀 LED 제어하기

다음은 RGB 네오픽셀 LED입니다. RGB 네오픽셀 LED는 WS2812 LED라고도 합니다. RGB 네오픽셀 LED는 R,G,B 각각에 대해 8비트 색깔을 낼 수 있으며, 전체 24비트로 색깔을 표현할 수 있습니다. 네오픽셀은 그 모양을 다양하게 구성하여 프로젝트에 활용하기 좋습니다. 그리고 하나의 제어 핀을 통해 직렬로 몇 개든지 연결이 가능합니다.



ESP32 Arduino AI 쉴드 상에는 다음과 같이 4개의 RGB 네오픽셀이 장착되어 있습니다. 4개의 RGB 네오픽셀은 아두이노 ESP32의 0 번 핀에 직렬로 연결되어 있습니다. 여기서는 0 번 핀에 직렬로 연결된 4개의 RGB 네오픽셀을 제어해 봅니다.



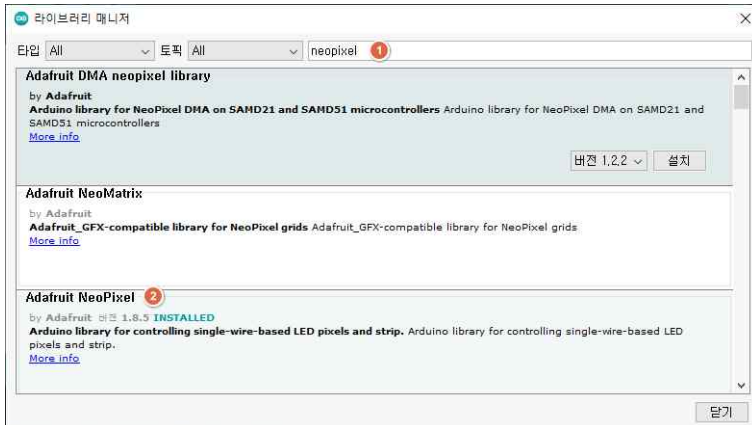
### 01 NeoPixel 라이브러리 설치하기

먼저 NeoPixel 라이브러리를 아두이노 소프트웨어에 설치합니다.

1. [스케치]--[라이브러리 포함하기]--[라이브러리 관리...] 메뉴를 선택합니다.



2. neopixel을 검색한 후, [Adafruit NeoPixel] 라이브러리를 설치합니다.



## 02 RGB 네오픽셀 LED 제어해 보기

여기서는 NeoPixel 라이브러리를 이용하여 RGB 네오픽셀 LED를 제어해 봅니다.

### LED 하나 켜고 꺼 보기

먼저 첫 번째 LED를 켜고 꺼봅니다.

1. 다음과 같이 예제를 작성합니다.

392.ino

```
01 #include <Adafruit_NeoPixel.h>
02
03 #define PIN      15
04 #define NUMPIXELS 4
05
06 Adafruit_NeoPixel pixels(NUMPIXELS, PIN);
07
08 void setup() {
09     pixels.begin();
10     pixels.clear();
```

```

11     pixels.setBrightness(255/100.0*20);
12
13     for(int cnt=0;cnt<5;cnt++) {
14         pixels.setPixelColor(0, pixels.Color(255, 0, 0));
15         pixels.show();
16         delay(500);
17
18         pixels.setPixelColor(0, pixels.Color(0, 255, 0));
19         pixels.show();
20         delay(500);
21     }
22
23     pixels.setPixelColor(0, pixels.Color(0, 0, 0));
24     pixels.show();
25 }
26
27 void loop() {
28
29 }

```

01 : Adafruit\_NeoPixel.h 파일을 포함합니다.

03 : 첫 번째 네오피셀이 연결된 핀 번호를 정의합니다.

04 : 네오피셀 LED의 개수를 정의합니다.

06 : Adafruit\_NeoPixel 객체인 pixels를 생성합니다. 첫 번째 인자는 네오피셀 LED의 개수, 두 번째 인자는 네오피셀이 연결된 핀 번호입니다.

09 : begin 함수를 호출하여 네오피셀을 초기화합니다.

10 : clear 함수를 호출하여 모든 네오피셀을 끕니다.

11 : setBrightness 함수를 호출하여 네오피셀을 밝기를 20%로 설정합니다. 네오피셀의 밝기는 최소 0에서 최대 255단계로 조절할 수 있습니다.

13 : 14~20줄을 5회 수행합니다.

14 : setPixelColor 함수를 호출하여 첫 번째 네오피셀((0번 네오피셀)의 색깔을 빨간색으로 설정합니다. pixels.Color 함수의 인자는 차례대로 R, G, B를 나타냅니다. R, G, B 각각의 값은 0~255 범위의 값을 줄 수 있습니다.

15 : show 함수를 호출하여 설정된 네오피셀의 색깔을 실제로 LED에 표시합니다.

16 : delay 함수를 호출하여 500 밀리 초 동안 기다립니다.

18 : setPixelColor 함수를 호출하여 첫 번째 네오피셀((0번 네오피셀)의 색깔을 초록색으로 설정합니다.

19 : show 함수를 호출하여 설정된 네오피셀의 색깔을 실제로 LED에 표시합니다.

20 : delay 함수를 호출하여 500 밀리 초 동안 기다립니다.

23 : setPixelColor 함수를 호출하여 첫 번째 네오피셀((0번 네오피셀)의 색깔을 검정색으로 설정합니다. 이렇게 하면 네오피셀이 꺼진 상태로 설정합니다.

24 : show 함수를 호출하여 설정된 네오피셀의 색깔을 실제로 LED에 표시합니다.

2. 업로드를 수행합니다.



3. 네오피셀을 동작을 확인합니다. 0 번째 RGB 네오피셀 LED가 0.5초 간격으로 빨간색과 초록색을 번갈아 바뀌는 것을 확인합니다. 이 동작을 5회 수행합니다.

## 전체 LED 켜고 꺼 보기

이번엔 전체 LED를 켜고 꺼봅니다.

1. 다음과 같이 예제를 수정합니다.

392\_2.ino

```
01 #include <Adafruit_NeoPixel.h>
02
03 #define PIN      15
04 #define NUMPIXELS 4
05
06 Adafruit_NeoPixel pixels(NUMPIXELS, PIN);
07
08 void setup() {
09     pixels.begin();
10     pixels.clear();
11     pixels.setBrightness(255/100.0*20);
12
13     for(int cnt=0;cnt<5;cnt++) {
14         pixels.fill(pixels.Color(255,0,0));
15         pixels.show();
16         delay(500);
17
18         pixels.fill(pixels.Color(0, 0, 255));
19         pixels.show();
20         delay(500);
21     }
22     pixels.fill(pixels.Color(0, 0, 0));
23     pixels.show();
24 }
25
26 void loop() {
```

```
27  
28 }
```

14, 18, 22 : setPixelColor 함수를 fill 함수로 변경합니다. fill 함수는 전체 LED의 색깔을 설정하는 함수입니다. 18줄에서는 전체 LED를 파란색으로 설정합니다.

2. 업로드를 수행합니다.



3. 네오픽셀을 동작을 확인합니다. 전체 RGB 네오픽셀 LED가 0.5초 간격으로 빨간색과 파란색을 번갈아 바뀌는 것을 확인합니다. 이 동작을 5회 수행합니다.

## 전체 LED 여러 색깔 켜고 꺼 보기

이번엔 전체 LED에 대해 여러 색깔로 켜고 꺼봅니다.

1. 다음과 같이 예제를 수정합니다.

392\_3.ino

```
01 #include <Adafruit_NeoPixel.h>  
02  
03 #define PIN      15  
04 #define NUMPIXELS 4  
05  
06 Adafruit_NeoPixel pixels(NUMPIXELS, PIN);  
07  
08 uint32_t BLACK = pixels.Color(0, 0, 0);  
09 uint32_t RED = pixels.Color(255, 0, 0);  
10 uint32_t YELLOW = pixels.Color(255, 150, 0);  
11 uint32_t GREEN = pixels.Color(0, 255, 0);  
12 uint32_t CYAN = pixels.Color(0, 255, 255);  
13 uint32_t BLUE = pixels.Color(0, 0, 255);  
14 uint32_t PURPLE = pixels.Color(180, 0, 255);  
15 uint32_t WHITE = pixels.Color(255, 255, 255);  
16 uint32_t COLORS[] = {BLACK, RED, YELLOW, GREEN, CYAN, BLUE, PURPLE, WHITE};  
17  
18 void setup() {  
19     pixels.begin();  
20     pixels.clear();  
21     pixels.setBrightness(255/100.0*20);  
22  
23     for(int cnt=0;cnt<5;cnt++) {  
24         for(int n=0;n<sizeof(COLORS)/sizeof(COLORS[0]);n++) {  
25             pixels.fill(COLORS[n]);  
26             pixels.show();  
27             delay(300);
```

```

28         }
29     }
30
31     pixels.fill(BLACK);
32     pixels.show();
33 }
34
35 void loop() {
36
37 }

```

08~15 : 네오픽셀 LED에 사용할 색깔을 정의합니다.

16 : 08~15줄에서 정의한 색깔들의 배열을 선언합니다.

23 : 24~28줄을 5회 수행합니다.

24 : COLORS 배열에 있는 색깔의 개수만큼 25~27줄을 수행합니다. sizeof(COLORS)는 COLORS 배열 전체의 크기를 의미하며, sizeof(COLORS[0])는 COLORS[0] 번 항목의 크기를 의미합니다. 배열 전체의 크기를 항목 한 개의 크기로 나누면 배열 항목의 개수가 됩니다.

25 : COLORS 배열의 n 번째 항목의 색깔로 네오픽셀 LED 전체를 설정합니다.

26 : show 함수를 호출하여 설정된 네오픽셀의 색깔을 실제로 LED에 표시합니다.

27 : delay 함수를 호출하여 300 밀리 초 동안 기다립니다.

2. 업로드를 수행합니다.



3. 네오픽셀을 동작을 확인합니다. 전체 RGB 네오픽셀 LED가 COLORS 배열에 있는 색깔 순서대로 켜졌다 꺼졌다 하는 것을 확인합니다. 이 동작을 5회 수행합니다.

## 무지개 색깔 내보기

이번엔 전체 LED에 대해 무지개 색깔로 켜고 꺼봅니다.

1. 다음과 같이 예제를 수정합니다.

392\_4.ino

```

01 #include <Adafruit_NeoPixel.h>
02
03 #define PIN 15
04 #define NUMPIXELS 4
05
06 Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
07
08 uint32_t BLACK = pixels.Color(0, 0, 0);
09 uint32_t RED = pixels.Color(255, 0, 0);
10 uint32_t YELLOW = pixels.Color(255, 150, 0);

```



```

11 uint32_t GREEN = pixels.Color(0, 255, 0);
12 uint32_t CYAN = pixels.Color(0, 255, 255);
13 uint32_t BLUE = pixels.Color(0, 0, 255);
14 uint32_t PURPLE = pixels.Color(180, 0, 255);
15 uint32_t WHITE = pixels.Color(255, 255, 255);
16 uint32_t COLORS[] = {BLACK, RED, YELLOW, GREEN, CYAN, BLUE, PURPLE, WHITE};
17
18 uint32_t wheel(int pos) {
19     //Input a value 0 to 255 to get a color value.
20     //The colours are a transition r - g - b - back to r.
21     if(pos < 0 || pos > 255) {
22         return pixels.Color(0, 0, 0);
23     } else if(pos < 85) {
24         return pixels.Color(255 - pos * 3, pos * 3, 0);
25     } else if(pos < 170) {
26         pos -= 85;
27         return pixels.Color(0, 255 - pos * 3, pos * 3);
28     } else {
29         pos -= 170;
30         return pixels.Color(pos * 3, 0, 255 - pos * 3);
31     }
32 }
33
34 void rainbow_cycle(int wait) {
35     int NUM_LEDS = pixels.numPixels();
36
37     for(int j=0;j<255;j++) {
38         for(int i=0;i<NUM_LEDS;i++) {
39             int rc_index = (i*256.0/NUM_LEDS)+j;
40             pixels.setPixelColor(i, wheel(rc_index&255));
41         }
42         pixels.show();
43         delay(wait);
44     }
45
46     pixels.clear();
47     pixels.show();
48 }
49
50 void setup() {
51     pixels.begin();
52     pixels.clear();
53     pixels.setBrightness(255/100.0*20);
54
55     rainbow_cycle(10);
56 }
57
58 void loop() {
59
60 }

```

18~32 : wheel 함수를 정의합니다. wheel 함수는 인자로 0~255 사이의 값을 받아 색깔

값을 내어줍니다. 색깔은 빨간색, 초록색, 파란색 다시 빨간색과 같은 단계로 변합니다. 함수에 대한 자세한 설명은 하지 않습니다.

34~44 : rainbow\_cycle 함수를 정의합니다.

35 : numPixels 함수를 호출하여 LED의 개수를 받습니다.

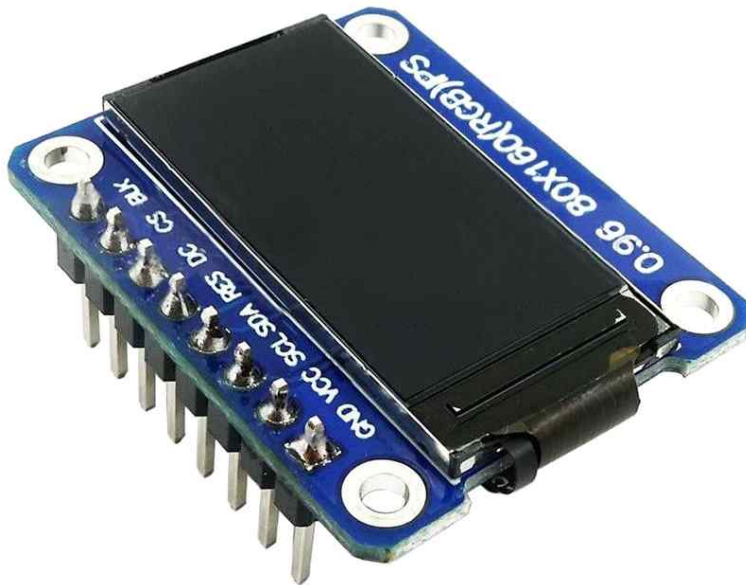
2. 업로드를 수행합니다.



3. 네오픽셀을 동작을 확인합니다. 각각의 LED가 여러 색깔로 예쁘게 표시되는 것을 확인합니다.

## 07 RGB LCD 제어하기

다음은 0.96 인치 RGB IPS LCD 디스플레이입니다. 이 LCD를 제어하기 위한 TFT 드라이버는 ST7735입니다. 작은 크기의 LCD이지만 160x80의 해상도와 16비트 RGB 컬러 픽셀을 가지고 있습니다. SPI 인터페이스를 이용하여 아두이노 ESP32와 연결되며, 전압은 3.3V를 사용합니다.



ESP32 Arduino AI 쉴드 상에는 다음과 같이 장착됩니다. RGB LCD는 IO18,23,4,19,5번 핀을 통해 아두이노 ESP32에 연결됩니다.



### RGB LCD 핀 설명

다음은 RGB LCD 핀 설명입니다.

핀	설명	라즈베리파이 피코 핀
GND	전원 기준 핀	GND
VCC	전원 공급 핀, 3.3V	VCC
SCL	SPI 클럭 핀	IO18
SDA	SPI 데이터 핀	IO23
RES	디스플레이 리셋 핀	IO4
DC	SPI 데이터 핀/ command 선택 핀	IO19

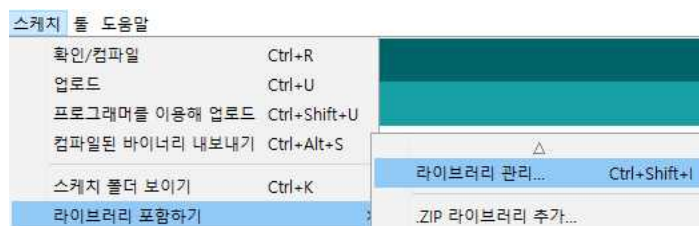
CS	SPI 칩 선택 핀, active low	IO5
BLK	LCD 백라이트 제어 핀	VCC

ESP32 Arduino AI 쉴드에서 BLK 핀의 경우 VCC와 연결되어 LCD 백라이트가 항상 켜진 상태를 유지하게 됩니다.

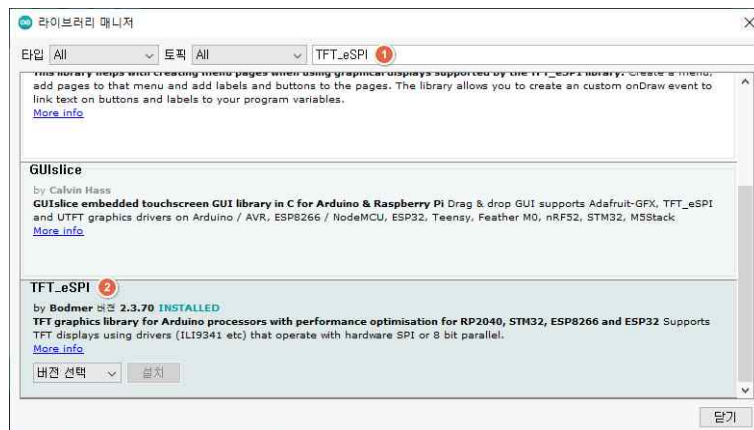
## 01 ST7735 라이브러리 설치하기

먼저 ST7735 라이브러리를 아두이노 소프트웨어에 설치합니다.

1. [스케치]--[라이브러리 포함하기]--[라이브러리 관리...] 메뉴를 선택합니다.



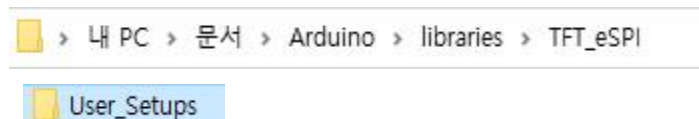
2. TFT\_eSPI를 검색한 후, [TFT\_eSPI] 라이브러리를 설치합니다.



3. 다음 디렉터리로 이동합니다. TFT\_eSPI 디렉터리를 확인합니다.



4. TFT\_eSPI 디렉터리로 이동합니다. User\_Setups 디렉터리를 확인합니다.



5. User\_Setups 디렉터리 아래에서 다음 파일을 찾습니다.

Setup43\_ST7735.h

6. 다음 디렉터리에 TFT\_eSPI\_Setups 디렉터를 새로 생성합니다.

> 내 PC > 문서 > Arduino > libraries >

TFT\_eSPI\_Setups

7. TFT\_eSPI\_Setups 디렉터리 아래로 Setup43\_ST7735.h 파일을 복사한 후, 다음과 같이 이름을 Setup43\_ST7735\_ESP32\_80x160.h으로 변경합니다.

> 내 PC > 문서 > Arduino > libraries > TFT\_eSPI\_Setups

Setup43\_ST7735\_ESP32\_80x160.h

Setup136\_LilyGo\_TTV\_ESP32\_135x240.h

8. 다음과 같이 파일의 내용을 수정한 후, 저장해 줍니다.

```
1 // Setup for ESP32 and ST7735 80 x 160 TFT
2
3 // See SetupX_Template.h for all options available
4
5 #define ST7735_DRIVER
6
7
8 #define TFT_WIDTH 80
9 #define TFT_HEIGHT 160
10
11
12 #define ST7735_GREENTAB160x80
13
14 // For ST7735, ST7789 and ILI9341 ONLY, define the colour order IF the blue and red are swapped on your display
15 // Try ONE option at a time to find the correct colour order for your display
16
17 // #define TFT_RGB_ORDER TFT_RGB // Colour order Red-Green-Blue
18 // #define TFT_RGB_ORDER TFT_BGR // Colour order Blue-Green-Red
19
20 #ifndef ESP32
21 #define TFT_MISO 19
22 #define TFT_MOSI 23
23 #define TFT_SCLK 18
24 #define TFT_CS 5//15 // Chip select control pin
25 #define TFT_DC 19//2 // Data Command control pin
26 #define TFT_RST 4 // Reset pin (could connect to RST pin)
27 // #define TFT_RST -1 // Set TFT_RST to -1 if display RESET is connected to ESP32 board RST
28 #else
29 // Display GND to NodeMCU pin GND (0V)
30 // Display VCC to NodeMCU 5V or 3.3V
31 // Display SCK to NodeMCU pin D5
32 // Display SDI/MOSI to NodeMCU pin D7
33 // Display BLK to NodeMCU pin VIN
34 #define TFT_CS PIN_D8 // Chip select control pin D8
35 #define TFT_DC PIN_D3 // Data Command control pin
36 #define TFT_RST PIN_D4 // Reset pin (could connect to NodeMCU RST, see next line)
37 #endif
38
39 #define LOAD_GLCD // Font 1. Original Adafruit 8 pixel font needs ~1820 bytes in FLASH
40 #define LOAD_FONT2 // Font 2. Small 16 pixel high font, needs ~3534 bytes in FLASH, 96 characters
41 #define LOAD_FONT4 // Font 4. Medium 26 pixel high font, needs ~5848 bytes in FLASH, 96 characters
42 #define LOAD_FONT6 // Font 6. Large 48 pixel font, needs ~2666 bytes in FLASH, only characters 1234567890:.-.apm
43 #define LOAD_FONT7 // Font 7. 7 segment 48 pixel font, needs ~2438 bytes in FLASH, only characters 1234567890:.-.
44 #define LOAD_FONT8 // Font 8. Large 75 pixel font needs ~3256 bytes in FLASH, only characters 1234567890:.-.
45 // #define LOAD_FONT8N // Font 8. Alternative to Font 8 above, slightly narrower, so 3 digits fit a 160 pixel TFT
46 #define LOAD_GFXFF // FreeFonts. Include access to the 48 Adafruit_GFX free fonts FF1 to FF48 and custom fonts
47
48 #define SMOOTH_FONT
49
50
51 // #define SPI_FREQUENCY 20000000
52 #define SPI_FREQUENCY 27000000 // Actually sets it to 26.67MHz = 80/3
53
54 #define TFT_RGB_ORDER TFT_BGR
```

24 : 15를 5로 변경합니다.

25 : 2를 19로 변경합니다.

54 : TFT\_RGB\_ORDER를 TFT\_BGR로 정의합니다.

9. TFT\_eSPI 디렉터리로 이동합니다.

📁 > 내 PC > 문서 > Arduino > libraries > TFT\_eSPI

10. 다음 파일을 찾습니다.

📄 User\_Setup\_Select.h

11. 다음과 같이 파일을 수정한 후, 저장해 줍니다.

```
1 // This header file contains a list of user setup files and defines which one the
2 // compiler uses when the IDE performs a Verify/Compile or Upload.
3 //
4 // Users can create configurations for different Espressif boards and TFT displays.
5 // This makes selecting between hardware setups easy by "uncommenting" one line.
6
7 // The advantage of this hardware configuration method is that the examples provided
8 // with the library should work with different setups immediately without any other
9 // changes being needed. It also improves the portability of users sketches to other
10 // hardware configurations and compatible libraries.
11 //
12 // Create a shortcut to this file on your desktop to permit quick access for editing.
13 // Re-compile and upload after making and saving any changes to this file.
14
15 // Customised User_Setup files are stored in the "User_Setups" folder.
16
17 #ifndef USER_SETUP_LOADED // Lets PlatformIO users define settings in
18 // platformio.ini, see notes in "Tools" folder.
19
20 // Only ONE line below should be uncommented. Add extra lines and files as needed.
21
22 // #include <User_Setup.h> 주석처리!!! // Default setup is root library folder
23 #include <../TFT_eSPI_Setups/Setup43_ST7735_ESP32_80x160.h>
24
25 // #include <User_Setups/Setup1_ILI9341.h> // Setup file configured for my ILI9341
26 // #include <User_Setups/Setup2_ST7735.h> // Setup file configured for my ST7735
```

22 : 주석 처리합니다. 이 부분을 주석처리하지 않으면 LCD가 비정상적으로 동작합니다.

23 : Setup43\_ST7735\_ESP32\_80x160.h 파일을 포함합니다.

## 02 RGB LCD 제어해 보기

여기서는 TFT\_eSPI 라이브러리를 이용하여 RGB LCD를 제어해 봅니다. TFT\_eSPI 라이브러리에는 다음과 같이 몇 가지 색깔이 정의되어 있습니다.



```

251  /*****
252  **                                     Section 6: Colour enumeration
253  *****/
254  // Default color definitions
255  #define TFT_BLACK      0x0000      /*  0,   0,   0 */
256  #define TFT_NAVY      0x000F      /*  0,   0, 128 */
257  #define TFT_DARKGREEN  0x03E0      /*  0, 128,   0 */
258  #define TFT_DARKCYAN  0x03EF      /*  0, 128, 128 */
259  #define TFT_MAROON    0x7800      /* 128,   0,   0 */
260  #define TFT_PURPLE    0x780F      /* 128,   0, 128 */
261  #define TFT_OLIVE     0x7BE0      /* 128, 128,   0 */
262  #define TFT_LIGHTGREY  0xD69A      /* 211, 211, 211 */
263  #define TFT_DARKGREY   0x7BEF      /* 128, 128, 128 */
264  #define TFT_BLUE      0x001F      /*   0,   0, 255 */
265  #define TFT_GREEN     0x07E0      /*   0, 255,   0 */
266  #define TFT_CYAN      0x07FF      /*   0, 255, 255 */
267  #define TFT_RED       0xF800      /* 255,   0,   0 */
268  #define TFT_MAGENTA   0xF81F      /* 255,   0, 255 */
269  #define TFT_YELLOW    0xFFE0      /* 255, 255,   0 */
270  #define TFT_WHITE     0xFFFF      /* 255, 255, 255 */
271  #define TFT_ORANGE    0xFDA0      /* 255, 180,   0 */
272  #define TFT_GREENYELLOW 0xB7E0      /* 180, 255,   0 */
273  #define TFT_PINK      0xFE19      /* 255, 192, 203 */ //Lighter pink, was 0xFC9F
274  #define TFT_BROWN     0x9A60      /* 150,  75,   0 */
275  #define TFT_GOLD      0FEA0      /* 255, 215,   0 */
276  #define TFT_SILVER    0xC618      /* 192, 192, 192 */
277  #define TFT_SKYBLUE   0x867D      /* 135, 206, 235 */
278  #define TFT_VIOLET    0x915C      /* 180,  46, 226 */

```

## 문자열 표시해보기

먼저 화면에 문자열을 표시해봅니다.

1. 다음과 같이 예제를 작성합니다.  
3102.ino

```

01 #include <SPI.h>
02 #include <TFT_eSPI.h>
03
04 TFT_eSPI tft = TFT_eSPI();
05
06 void setup() {
07
08     tft.begin();
09     tft.setRotation(3);

```

```

10
11     tft.fillScreen(TFT_DARKGREY);
12
13     tft.setCursor(0, 10);
14     tft.setTextFont(2);
15     tft.setTextColor(TFT_RED, TFT_BLACK);
16     tft.setTextSize(1);
17     tft.println("Hello World!");
18
19     tft.setTextFont(4);
20     tft.setTextColor(TFT_YELLOW);
21     tft.println(1234.56);
22
23 }
24
25 void loop() {
26
27 }

```

01 : SPI.h 파일을 포함합니다. ST7735 driver chip은 아두이노 ESP32 와 SPI 통신을 수행합니다.

02 : TFT\_eSPI.h 파일을 포함합니다.

04 : TFT\_eSPI 객체를 생성한 후, tft 변수에 할당합니다.

08 : begin 함수를 호출하여 TFT LCD를 초기화합니다.

09 : setRotation 함수를 호출하여 LCD의 출력 방향을 3으로 설정합니다. 설정 값은 0, 1, 2, 3을 가질 수 있습니다. 설정 값에 따라 LCD 출력 기준점이 달라집니다. 설정 값의 역할을 보기 위해 값을 변경해 봅니다.

11 : fillScreen 함수를 호출하여 화면의 색깔을 어두운 회색으로 설정합니다. 여기서는 15 줄에서 수행되는 글자 배경색 설정의 효과를 보기 위해 어두운 회색으로 설정합니다.

13 : setCursor 함수를 호출하여 글자 출력의 픽셀 위치를 설정합니다. 첫 번째 인자는 가로 픽셀 위치, 두 번째 인자는 세로 픽셀 위치를 나타냅니다. 여기서는 가로 0픽셀, 세로 10픽셀 위치에 커서를 설정합니다.

14 : setTextFont 함수를 호출하여 폰트를 설정합니다. 폰트는 Setup43\_ST7735\_ESP32\_80x160.h 파일에 정의되며 1, 2, 4, 6, 7, 8 값으로 설정할 수 있습니다. 6, 7, 8의 경우 숫자만 표시됩니다.

15 : setTextColor 함수를 호출하여 글자의 색깔을 설정합니다. 표시될 글자의 색깔은 빨간색, 배경색은 검정색으로 설정합니다.

16 : setTextSize 함수를 호출하여 글자의 크기를 설정합니다. 이 함수는 기본 폰트가 표시될 픽셀의 배수가 됩니다.

17 : println 함수를 호출하여 "Hello World!" 문자열을 출력합니다.

19 : setTextFont 함수를 호출하여 폰트를 설정합니다.

20 : setTextColor 함수를 호출하여 글자의 색깔을 설정합니다. 여기서는 글자의 색깔만

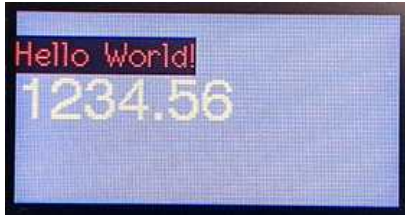


노란색으로 설정합니다. 이 경우 글자의 배경색은 기본 배경색이 사용됩니다.  
21 : println 함수를 호출하여 1234.56 숫자를 출력합니다.

2. 업로드를 수행합니다.



3. 다음과 같이 화면에 문자열이 출력되는 것을 확인합니다.



## 그림 그려보기

다음은 화면에 그림을 그려봅니다.

1. 다음과 같이 예제를 작성합니다.

3102\_2.ino

```
01 #include <SPI.h>
02 #include <TFT_eSPI.h>
03
04 TFT_eSPI tft = TFT_eSPI();
05
06 void setup() {
07
08     tft.begin();
09     tft.setRotation(3);
10
11     tft.fillScreen(TFT_BLACK);
12
13     tft.fillRect(4,4,56,56,TFT_BLUE);
14     tft.drawFastVLine(18,16,44,TFT_BLACK);
15     tft.drawFastVLine(32,4,44,TFT_BLACK);
16     tft.drawFastVLine(46,16,44,TFT_BLACK);
17     tft.fillRect(52,48,4,8,TFT_BLACK);
18
19     tft.setCursor(73, 10);
20     tft.setTextFont(2);
```

```


21 tft.setTextSize(1);
22 tft.setTextColor(TFT_BLUE);
23 tft.println("MicroPython");
24 tft.setCursor(73, 25);
25 tft.println("ST7735");
26 tft.setCursor(73, 40);
27 tft.println("LCD 80x160");
28
29 }
30
31 void loop() {
32
33 }

```

13 : fillRect 함수를 호출하여 사각영역을 파란색으로 채웁니다. 첫 번째, 두 번째 인자는 가로 픽셀, 세로 픽셀의 위치, 세 번째, 네 번째 인자는 픽셀 단위의 가로 크기, 세로 크기, 다섯 번째 인자는 색깔을 나타냅니다.

14~16 : drawFastVLine 함수를 호출하여 수직선을 그립니다. 첫 번째, 두 번째 인자는 가로 픽셀, 세로 픽셀의 위치, 세 번째 인자는 픽셀 단위의 세로 길이, 네 번째 인자는 색깔을 나타냅니다.

17 : fillRect 함수를 호출하여 사각영역을 검정색으로 채웁니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음과 같이 화면에 그림이 표시되는 것을 확인합니다.



## 픽셀 찍어보기

여기서는 픽셀단위로 색깔을 칠해 봅니다.

1. 다음과 같이 예제를 작성합니다.

3102\_3.ino

```

01 #include <SPI.h>
02 #include <TFT_eSPI.h>
03
04 TFT_eSPI tft = TFT_eSPI();

```

```

05
06 void setup() {
07
08     tft.begin();
09     tft.setRotation(3);
10
11     tft.fillScreen(TFT_BLACK);
12
13     for(int y=0;y<80;y++) {
14         for(int x=0;x<160;x++) {
15             tft.drawPixel(x, y, TFT_GREEN);
16             delay(1);
17         }
18     }
19
20 }
21
22 void loop() {
23
24 }

```

11 : fillScreen 함수를 호출하여 화면을 검정색으로 설정합니다.

13 : 0에서 80 미만의 정수 y에 대하여 14~17줄을 수행합니다.

14 : 0에서 160 미만의 정수 x에 대하여 15~16줄을 수행합니다.

15 : drawPixel 함수를 호출하여 x, y 위치의 픽셀을 초록색으로 설정합니다.

16 : delay 함수를 호출하여 1밀리 초 기다립니다.

2. 업로드를 수행합니다.



3. 화면에 픽셀 단위로 가로로 차례대로 색깔이 표시되는 것을 확인합니다.



## LCD 귀퉁이 점찍어보기

여기서는 LCD의 네 귀퉁이를 확인해 봅니다.

1. 다음과 같이 예제를 작성합니다.

3102\_4.ino

```
01 #include <SPI.h>
02 #include <TFT_eSPI.h>
03
04 TFT_eSPI tft = TFT_eSPI();
05
06 void setup() {
07
08     tft.begin();
09     tft.setRotation(3);
10
11     tft.fillScreen(TFT_BLACK);
12
13     for(int y=0;y<10;y++) {
14         for(int x=0;x<10;x++) {
15             tft.drawPixel(x, y, TFT_RED);
16         }
17     }
18
19     for(int y=70;y<80;y++) {
20         for(int x=0;x<10;x++) {
21             tft.drawPixel(x, y, TFT_GREEN);
22         }
23     }
24
25     for(int y=70;y<80;y++) {
26         for(int x=150;x<160;x++) {
27             tft.drawPixel(x, y, TFT_BLUE);
28         }
29     }
30
31     for(int y=0;y<10;y++) {
32         for(int x=150;x<160;x++) {
33             tft.drawPixel(x, y, TFT_YELLOW);
34         }
35     }
36
37 }
```

```

38
39 void loop() {
40
41 }

```

2. 업로드를 수행합니다.



3. 다음과 같이 화면에 네 귀퉁이에 색깔이 표시되는 것을 확인합니다.



빨간색 위치가 (0, 0), 초록색 위치가 (0, 79), 파란색 위치가 (159, 79), 노란색 위치가 (159, 0)인 것을 확인합니다.

## 기타 함수 사용해 보기

앞에서 사용하지 않은 몇 가지 주요 함수를 사용해 봅니다.

1. 다음과 같이 예제를 작성합니다.

3102\_5.ino

```

01 #include <SPI.h>
02 #include <TFT_eSPI.h>
03
04 TFT_eSPI tft = TFT_eSPI();
05
06 void setup() {
07
08     tft.begin();
09     tft.setRotation(3);
10
11     tft.fillScreen(TFT_BLACK);
12
13     tft.drawLine(0,0,159,79,TFT_RED);
14     tft.drawFastHLine(0,40,159,TFT_BLUE);
15     tft.drawRect(10,50,20,10,TFT_GREEN);

```

```
16 tft.drawCircle(70,60,10,TFT_CYAN);
17 tft.fillCircle(100,20,10,TFT_PURPLE);
18
19 }
20
21 void loop() {
22
23 }
```

13 : drawLine 함수를 호출하여 선을 그립니다.

14 : drawFastHLine 함수를 호출하여 수평선을 그립니다.

15 : drawRect 함수를 호출하여 직사각형을 그립니다.

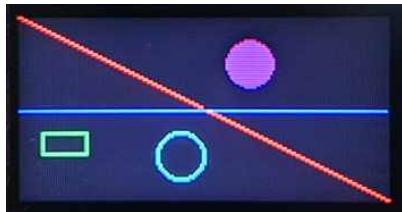
16 : drawCircle 함수를 호출하여 원을 그립니다.

17 : fillCircle 함수를 호출하여 채워진 원을 그립니다.

2. 업로드를 수행합니다.

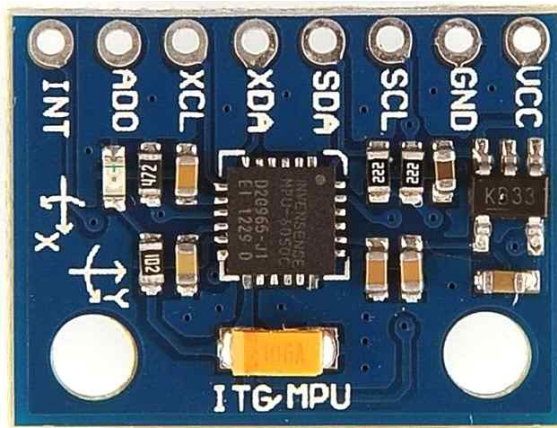


3. 다음과 같이 화면이 표시되는 것을 확인합니다.

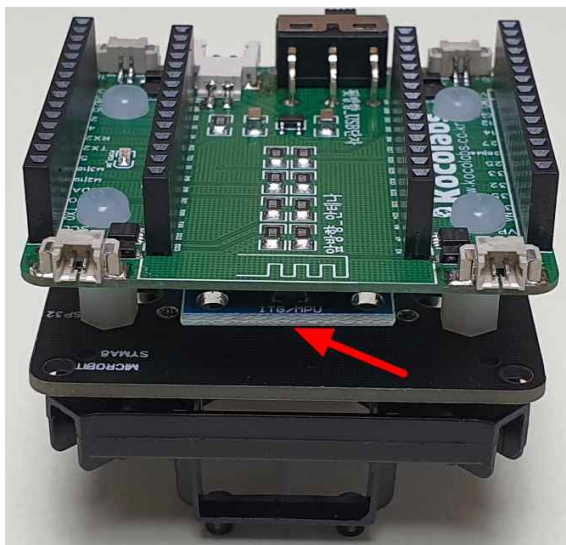


## 08 MPU6050 균형계 살펴보기

우리가 사용하고 있는 아두이노 드론에는 다음과 같은 GY-521 MPU6050 모듈이 장착되어 있습니다.

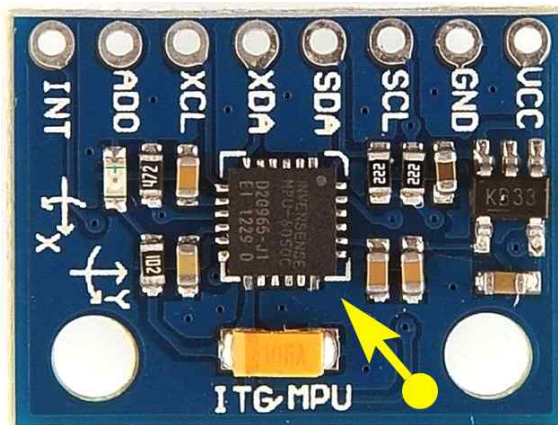


다음과 같이 드론의 전방 우측에 장착되어 있습니다.

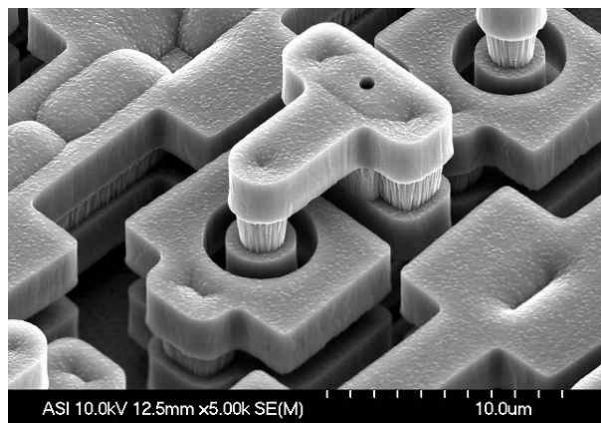
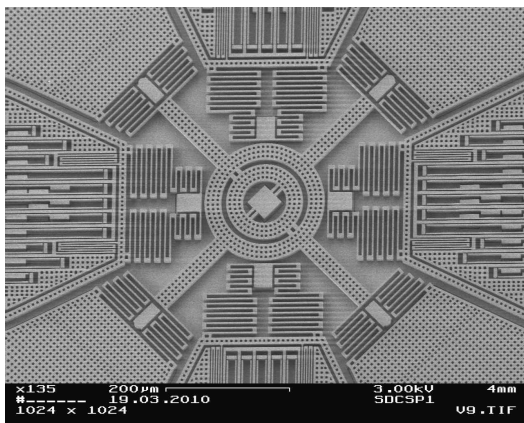


MPU6050 모듈은 가속도 자이로 센서를 이용하여 드론의 기울어진 정도와 회전속도를 알려줍니다. 우리는 드론의 기울어진 정도나 회전속도에 따라 모터의 속도를 조절해 드론의 중심을 잡게 됩니다. MPU6050 모듈은 가속도 3축, 자이로 3축, 온도에 대한 총 7 가지 센서 값을 제공합니다. 이 중 우리는 가속도, 자이로 센서에 대한 값을 활용하게 됩니다. 가속도 자이로 센서에 대한 분석 방법은 아주 복잡하지만, 드론의 동작을 이해하기 위해 꼭 필요한 부분입니다.

이 책에서 다루는 가속도 자이로 센서는 InvenSense 사의 제품인 MPU6050 센서로 다음 사진의 가운데에 있는 칩입니다.



MPU6050 모듈은 하나의 칩 안에 MEMS 가속도 센서와 MEMS 자이로 센서를 가지고 있습니다. MEMS란 Micro Electro Mechanical Systems의 약자로 미세 전자기계 시스템으로 불리며, 반도체 제조 공정 기술을 기반으로 한 마이크로미터( $\mu\text{m}$ )이나 밀리미터(mm)크기의 초소형 정밀기계 제작 기술을 말합니다. 아래 그림은 MEMS 기술로 만들어진 초소형 기계 시스템을 보여주고 있습니다.

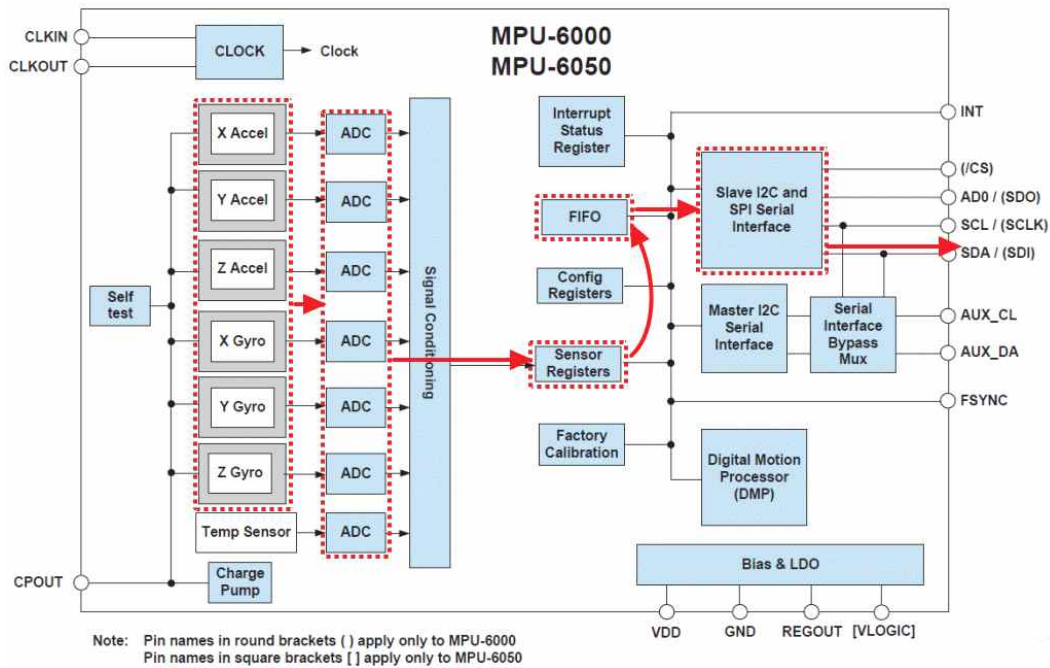


<출처 : <http://www.machinedesign.com>> <출처 : <http://www.kinews.net>>

MPU6050 센서는 각 채널에 대해 16 비트 크기의 값을 출력해 주는 ADC 모듈을 가지고 있습니다.

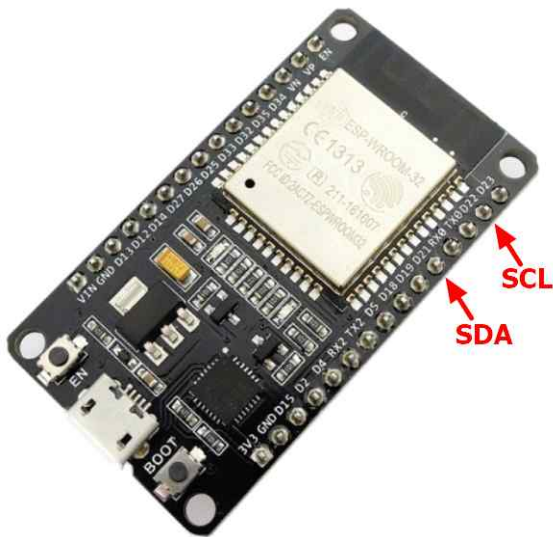
다음은 MPU6050 센서의 내부 블록도입니다.





X, Y, Z 축에 대한 가속도와 자이로 값이 각각의 ADC 블록을 거쳐 센서 레지스터(Sensor Register)에 저장됩니다. 센서 레지스터는 센서 내부에 있는 이름을 가진 변수와 같습니다. 센서 레지스터에 저장된 값은 I2C 통신을 통해 ESP32 아두이노로 전달됩니다.

다음은 ESP32의 I2C 핀입니다.

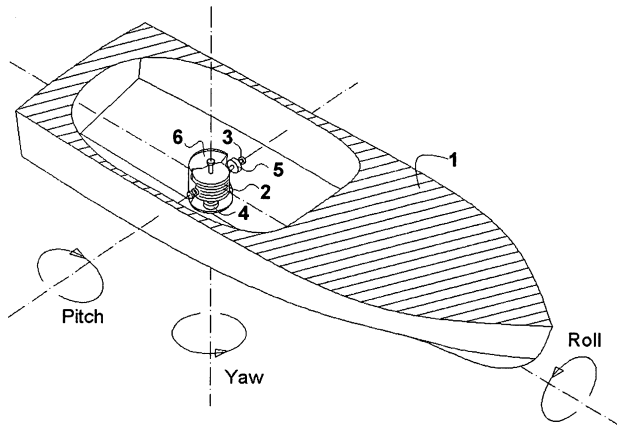


MPU6050 센서는 ESP32와 I2C 통신을 합니다. ESP32의 I2C 핀은 21, 22 번 핀으로 각각 ESP32 내부에 있는 I2C 모듈의 SDA, SCL 핀과 연결됩니다. ESP32의 SDA, SCL 핀은 MPU6050 센서의 SDA, SCL 핀과 연결됩니다.

## 01 Roll, Pitch, Yaw 이해하기

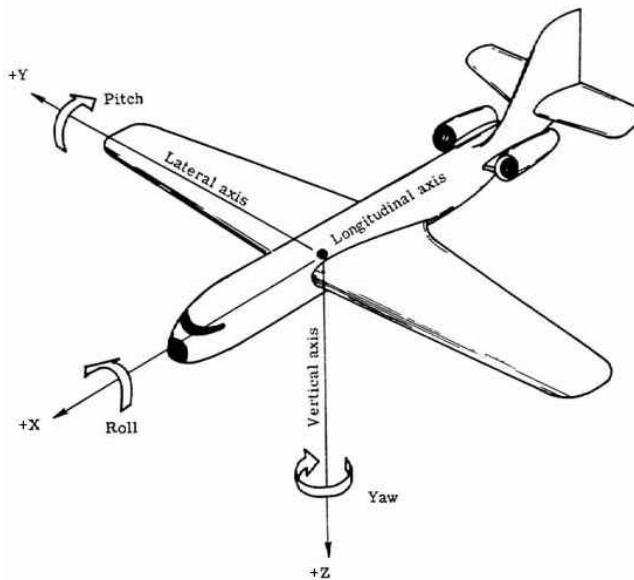
드론에서 Roll, Pitch, Yaw는 아주 중요한 요소입니다. 이 세 가지 조건에 대한 정확한 정보가 이 없다면, 드론을 제대로 띄울 수가 없습니다. 그러면 Roll, Pitch, Yaw란 무엇일까요?

다음과 같은 형태의 배가 물에 떠 있는 경우를 생각해 봅시다.



전방을 기준으로 배는 좌우로 흔들릴 수 있습니다. 배는 앞뒤로도 흔들릴 수 있습니다. 배는 방향을 전환할 수도 있습니다. 이 때, 각각을 Roll, Pitch, Yaw라고 합니다. 배의 경우는 Yaw가 아주 중요한 요소가 됩니다.

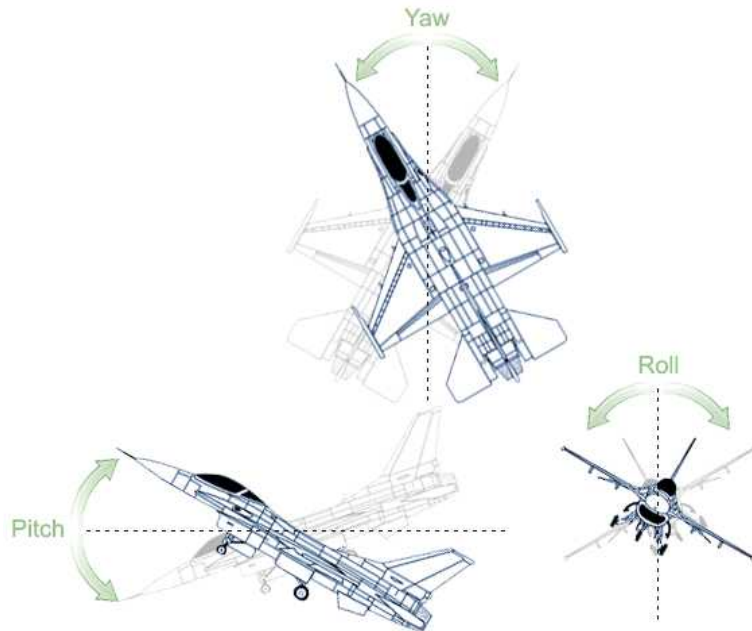
이러한 현상은 비행기에도 나타날 수 있습니다. 다음 그림을 살펴봅시다.



Roll은 비행체의 좌우 기울어짐의 정도, Pitch는 전후 기울어짐의 정도, Yaw는 수평 회전

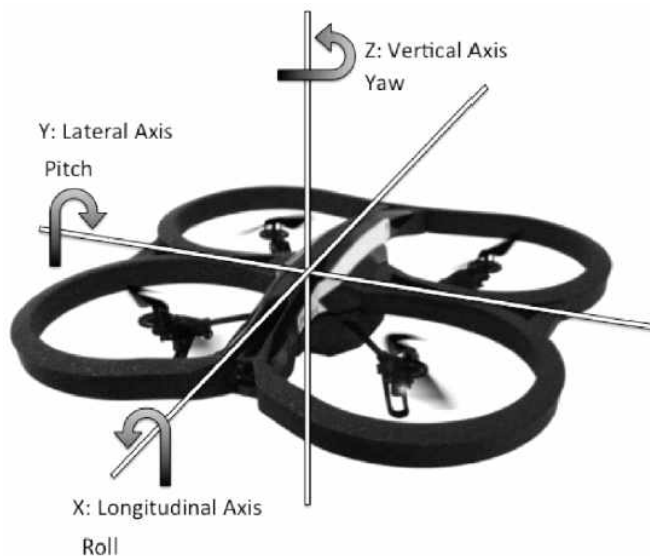
정도를 나타냅니다.

다음 그림은 비행기의 Roll, Pitch, Yaw를 좀 더 구체적으로 보여주고 있습니다.



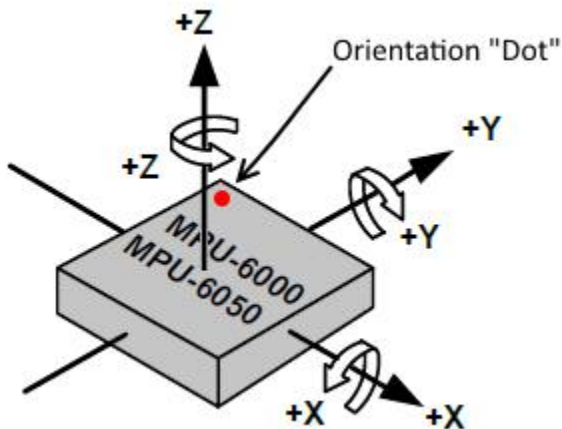
비행기의 경우엔 Roll, Pitch, Yaw가 모두 중요합니다.

드론의 경우는 비행기와 같습니다. 아래 그림은 드론의 Roll, Pitch, Yaw를 나타냅니다.



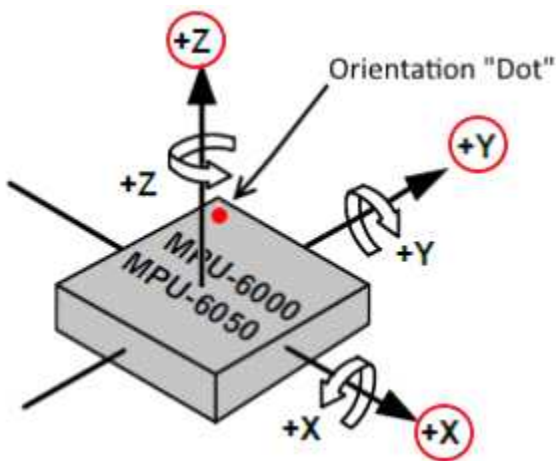
드론의 전방을 기준으로 좌우 기울어짐을 Roll, 전후 기울어짐을 Pitch, 수평 회전 정도를 Yaw라고 합니다. 드론의 Roll, Pitch, Yaw 대한 정보는 MPU6050 센서를 이용하여 얻어낼 수 있습니다.

MPU6050 센서를 좀 더 자세히 살펴보도록 하겠습니다.

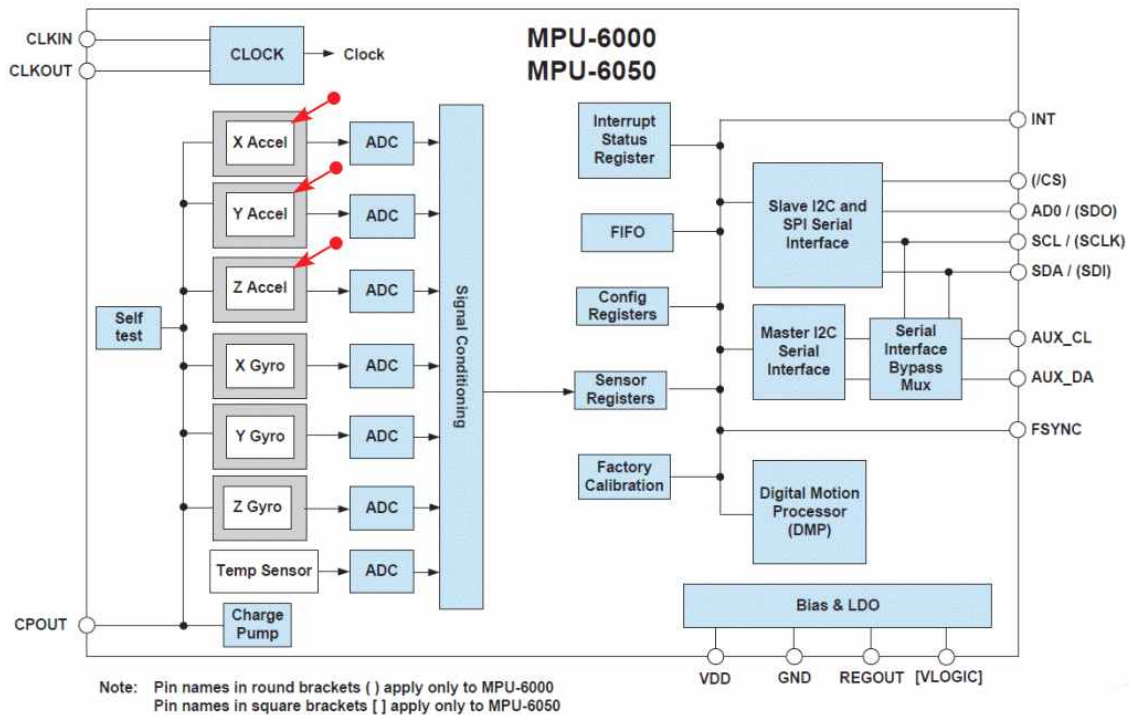


그림에서 MPU6050 센서는 총 6개의 축을 표시하고 있습니다. 직선 3축과 곡선 3축을 합쳐서 총 6개의 축이 됩니다. 직선 3축은 기울기 센서가 사용합니다. 곡선 3축은 자이로 센서가 사용합니다. 직선 3축은 각 축에 대해 중력 방향을 기준으로 센서의 기울어진 정도를 측정할 때 사용합니다. 곡선 3축은 직선 3축에 대한 회전 정도를 측정할 때 사용합니다.

## 02 가속도 센서 축의 이해



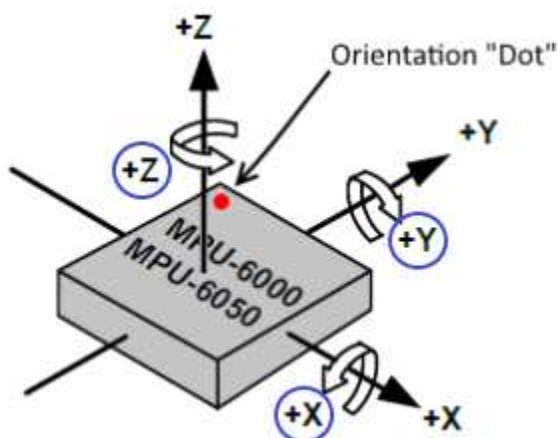
그림에서 직선 +X, +Y, +Z 방향은 가속도 센서의 + 방향입니다. 예를 들어, 센서가 정적인 상태에서 직선의 +X가 중력 방향과 같은 방향을 보게 되면 가속도 센서 X\_Accel은 음수 값을 갖습니다. 또, 직선 +X가 중력 방향과 반대 방향을 보게 되면 가속도 센서 X\_Accel은 양수 값을 갖습니다. 직선 +Y, +Z에 대해서도 마찬가지로 방식으로 생각하면 됩니다. X\_Accel, Y\_Accel, Z\_Accel은 아래 그림의 센서 값을 나타냅니다.



가속도 센서 값은 뒤에서 실습을 통해 구체적으로 살펴 볼 것입니다.

MPU6050 센서를 수평면에 두었을 때, 직선 +Z는 중력과 정 반대 방향을 보게 되며, 가속도 센서 Z\_Accel은 양수 값을 갖게 됩니다.

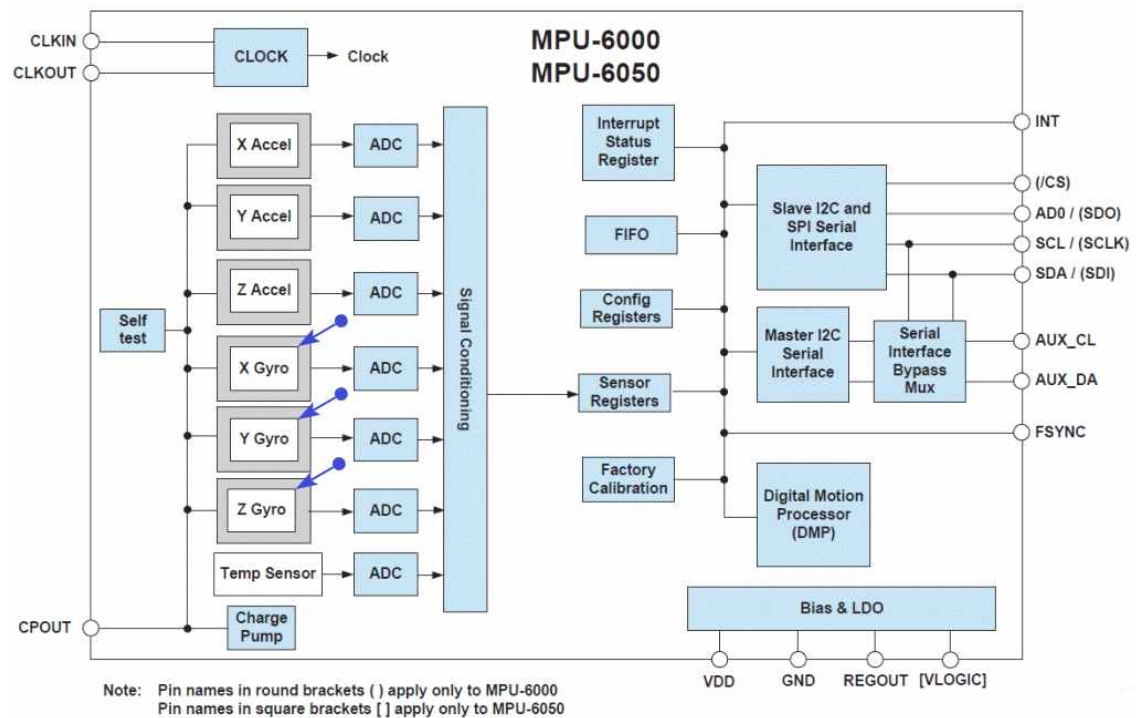
### 03 자이로 센서 축의 이해



그림에서 곡선 축의 +X, +Y, +Z는 센서가 회전할 경우에 자이로 센서의 + 값의 기준이 됩니다. 예를 들어, 센서가 +X 방향으로 돌면 자이로(각속도 또는 회전속도) 센서 X\_Gyro

는 양수 값을 갖습니다. 반대 방향으로 돌면 자이로 센서 X\_Gyro는 음수 값을 갖습니다. 곡선 축 +Y, +Z에 대해서도 마찬가지로 방식으로 생각하면 됩니다.

X\_Gyro, Y\_Gyro, Z\_Gyro는 아래 그림의 센서 값을 나타냅니다.

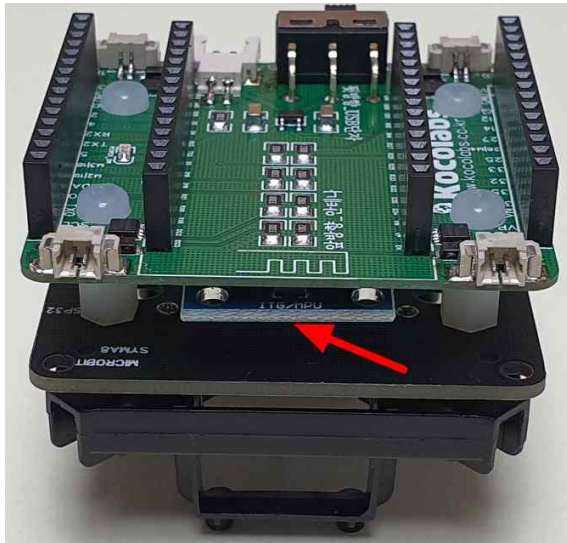


자이로 센서 값을 뒤에서 실습을 통해 구체적으로 살펴 볼 것입니다.

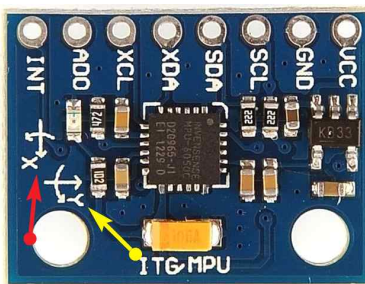
## 04 ESP32 아두이노 드론과 가속도 자이로 센서

우리가 사용하는 ESP32 아두이노 드론은 직선 축 +X, +Y가 각각 전방, 좌측을 보고 있습니다. 다음 그림을 살펴보도록 합니다.



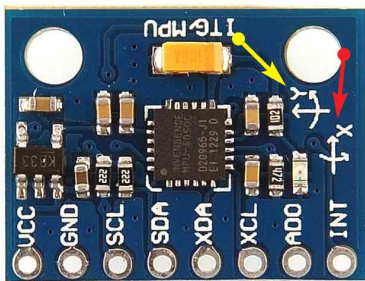


MPU6050 부분만 확대해 보면 다음과 같습니다.



빨간색 화살표는 직선과 곡선 축 +X를 가리키고, 노란색 화살표는 직선과 곡선 축 +Y를 가리키고 있습니다. 직선 축 +X는 전방을, 직선 축 +Y는 좌측을 가리킵니다.

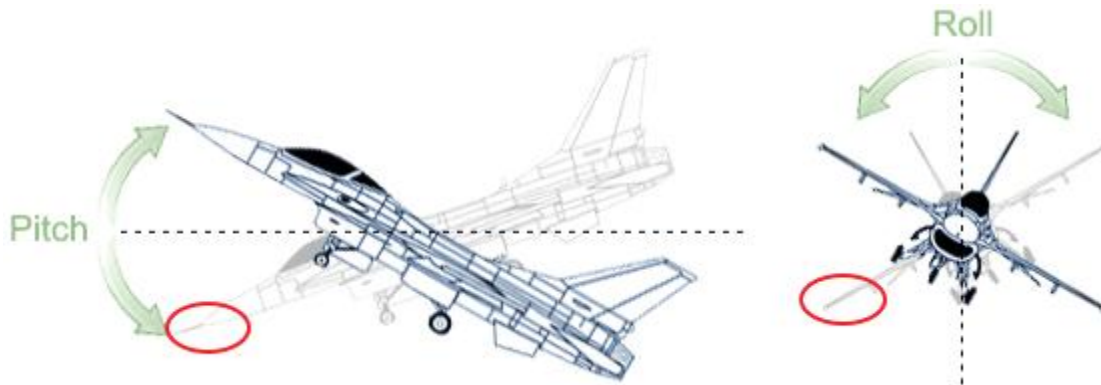
그러면 ESP32 아두이노 드론에 대해서 드론의 회전하는 상태에 따라 가속도와 자이로 센서 값에 대해 생각해 보겠습니다.



먼저 드론이 앞으로 기울어진 상태와 오른쪽으로 기울어진 상태에 대해서 생각해 보겠습니다. 드론이 앞으로 기울어지면 직선 +X축은 중력 방향을 향하게 되므로 기울기 센서 X\_Accel은 음수 값을 갖습니다. 드론이 오른쪽으로 기울어지면 직선 +Y축이 중력 반대 방

향을 향하게 되므로 기울기 센서 Y\_Accel은 양수 값을 갖습니다.

다음 그림은 비행기를 예로 앞으로 기울어진 상태와 오른쪽으로 기울어질 상태를 나타냅니다. 오른쪽 비행기의 경우 지면을 향해 나오는 그림입니다.



드론이 회전할 경우에 대해서도 살펴보겠습니다. 수평 상태에서 드론이 오른쪽으로 기울면서 회전이 발생할 경우 곡선 축 +X 방향으로 돌게 됩니다. 따라서 자이로 센서 X\_Gyro는 양수 값을 갖습니다. 드론이 왼쪽으로 기울면서 회전이 발생할 경우 곡선 축 +X 반대 방향으로 돌게 됩니다. 따라서 자이로 센서 X\_Gyro는 음수 값을 갖습니다. 수평 상태에서 드론의 앞부분이 땅 방향으로 회전이 발생할 경우 곡선 축 +Y 방향으로 돌게 됩니다. 따라서 자이로 센서 Y\_Gyro는 양수 값을 갖습니다. 드론의 앞부분이 하늘 방향으로 회전이 발생할 경우 곡선 축 +Y 반대 방향으로 돌게 됩니다. 따라서 자이로 센서 Y\_Gyro는 음수 값을 갖습니다.

## 05 MPU6050 레지스터 살펴보기

여기서는 MPU6050을 초기화하기 위한 설정 레지스터와 자이로 센서 값을 저장하는 레지스터를 살펴보도록 합니다. 레지스터는 CPU와 디바이스가 통신하기 위한 디바이스가 가진 변수와 같습니다.

다음은 PWR\_MGMT\_1 레지스터를 나타냅니다. 이 레지스터는 MPU6050의 내부 0x68 번지에 있습니다.

6B	107	PWR_MGMT_1	R/W	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]
----	-----	------------	-----	--------------	-------	-------	---	----------	-------------

SLEEP 부분이 1로 설정되면 MPU6050은 sleep mode가 되며 반대로 0으로 설정되면 깨어나게 됩니다.

다음은 MPU6050의 내부 0x3b~0x48 번지에 있는 14 바이트의 레지스터를 나타냅니다. 0x3b 번지부터 시작해 총 14 바이트 크기의 레지스터에 차례대로 가속도 센서, 온도 센서, 자이로 센서 값이 저장됩니다.

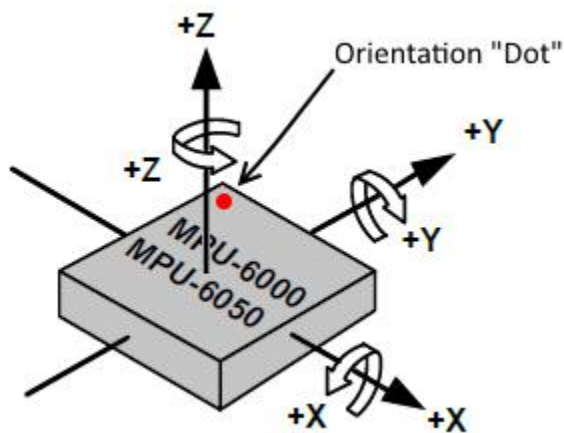


Addr (Hex)	Addr (Dec.)	Register Name
3B	59	ACCEL_XOUT_H
3C	60	ACCEL_XOUT_L
3D	61	ACCEL_YOUT_H
3E	62	ACCEL_YOUT_L
3F	63	ACCEL_ZOUT_H
40	64	ACCEL_ZOUT_L
41	65	TEMP_OUT_H
42	66	TEMP_OUT_L
43	67	GYRO_XOUT_H
44	68	GYRO_XOUT_L
45	69	GYRO_YOUT_H
46	70	GYRO_YOUT_L
47	71	GYRO_ZOUT_H
48	72	GYRO_ZOUT_L

각 센서의 값은 2바이트 크기를 갖습니다.

## 06 가속도 자이로 센서 값 읽어보기

여기서는 드론에 부착된 MPU6050 가속도 자이로 센서 값을 있는 그대로 읽어보도록 합니다.



1. 다음과 같이 예제를 작성합니다.

340.ino

```
01 #include <Wire.h>
02
03 void setup() {
04     Serial.begin(115200);
05
06     Wire.begin();
07     Wire.setClock(400000);
08
09     Wire.beginTransmission(0x68);
10     Wire.write(0x6b);
11     Wire.write(0x0);
12     Wire.endTransmission(true);
13 }
14
15 void loop() {
16     Wire.beginTransmission(0x68);
17     Wire.write(0x3b);
18     Wire.endTransmission(false);
19     Wire.requestFrom((uint16_t)0x68,(uint8_t)14,true);
20
21     int16_t AcXH = Wire.read();
22     int16_t AcXL = Wire.read();
23     int16_t AcYH = Wire.read();
24     int16_t AcYL = Wire.read();
25     int16_t AcZH = Wire.read();
26     int16_t AcZL = Wire.read();
27     int16_t TmpH = Wire.read();
28     int16_t TmpL = Wire.read();
29     int16_t GyXH = Wire.read();
30     int16_t GyXL = Wire.read();
31     int16_t GyYH = Wire.read();
32     int16_t GyYL = Wire.read();
33     int16_t GyZH = Wire.read();
34     int16_t GyZL = Wire.read();
35
36     int16_t AcX = AcXH <<8 | AcXL;
37     int16_t AcY = AcYH <<8 | AcYL;
38     int16_t AcZ = AcZH <<8 | AcZL;
```

```

39     int16_t GyX = GyXH <<8 | GyXL;
40     int16_t GyY = GyYH <<8 | GyYL;
41     int16_t GyZ = GyZH <<8 | GyZL;
42
43     static int cnt_loop;
44     cnt_loop ++;
45     if(cnt_loop%200 !=0) return;
46
47     Serial.printf(" AcX = %6d", AcX);
48     Serial.printf(" | AcY = %6d", AcY);
49     Serial.printf(" | AcZ = %6d", AcZ);
50     Serial.printf(" | GyX = %6d", GyX);
51     Serial.printf(" | GyY = %6d", GyY);
52     Serial.printf(" | GyZ = %6d", GyZ);
53     Serial.println();
54 }

```

1 : Wire.h 헤더 파일을 포함합니다. 이 파일은 I2C 통신을 할 때 필요한 파일입니다. 아두이노 마이크로 프로는 MPU6050 센서와 I2C 통신을 하기 때문에 이 파일이 필요합니다.

4: Serial의 통신 속도를 115200으로 설정하고 있습니다.

6 : Wire.begin 함수를 호출합니다. Wire.begin 함수는 I2C 통신 기능을 활성화하는 함수입니다.

7 : Wire.setClock 함수를 호출합니다. Wire.setClock 함수는 I2C 통신 속도를 설정하는 함수로 이 예제에서는 400KHz로 설정합니다. 400KHz는 400Kbps의 속도와 같습니다. Wire.SetClock 함수는 100000(standard mode) 또는 400000(fast mode)을 설정 값으로 받을 수 있습니다.

9,16 : Wire.beginTransmission 함수를 호출합니다. 이 함수는 인자로 주어진 I2C 슬레이브 모듈과 통신을 시작할 때 호출합니다. 여기서는 0x68 번지 값을 갖는 MPU6050과 I2C 통신을 시작하고 있습니다.

10,11,17 : Wire.write 함수를 호출합니다. 이 함수는 전송하고자 하는 1 바이트 데이터를 내부 메모리 큐에 저장하는 역할을 합니다.

12,18 : Wire.endTransmission 함수를 호출하고 있습니다. 이 함수는 Wire.write 함수에 의해 큐에 저장된 하나 이상의 바이트 데이터를 슬레이브 모듈로 보내면서 전송을 끝냅니다.

12 : 인자로 true 값을 넘기고 있습니다. true 값이 인자로 넘어오면 endTransmission 함수는 데이터 전송 후, 정지 메시지를 보내고 I2C 버스의 제어 권을 놓습니다.

18 : 인자로 false 값을 넘기고 있습니다. false 값이 인자로 넘어올 경우에 endTransmission 함수는 데이터 전송 후 통신 재시작 메시지를 보냅니다. 이 경우에는 I2C 버스에 대한 제어 권을 놓지 않습니다. 그래서 19 번 째 줄에서는 Wire.requestFrom 함수를 호출하여 추가적인 데이터를 요구하고 있습니다. Wire.requestFrom 함수에서는

MPU6050(0x68)에게 2 바이트의 데이터를 요구하고 있습니다. 3 번 째 인자로는 false 값을 사용하고 있는데, 이는 데이터 요청 후 정지 메시지를 보내며 I2C의 제어 권을 놓는 것을 의미합니다.

19 : Wire.requestFrom 함수를 호출하고 있습니다. 0x68 번지 값을 갖는 MPU6050에게 14 바이트의 데이터를 요청하고 동시에 true 값을 넘겨 I2C 버스의 제어 권을 놓습니다. ESP32 용 I2C 라이브러리는 주소로 uint16\_t, 데이터 개수로 uint8\_t의 매개변수를 사용합니다.

9~12 : MPU6050(0x68)으로 0x6b, 0을 보내고 있습니다. 이는 MPU6050의 내부 0x6b 번지를 0으로 설정하여 MPU6050을 깨우게 됩니다. 0x6b 번지에는 PWR\_MGMT\_1 레지스터가 있습니다. 다음은 PWR\_MGMT\_1를 나타냅니다.

6B	107	PWR_MGMT_1	R/W	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]
----	-----	------------	-----	--------------	-------	-------	---	----------	-------------

SLEEP 부분이 1로 설정되면 MPU6050은 sleep mode가 되며 반대로 0으로 설정되면 깨어나게 됩니다.

16~19 : MPU6050(0x68)의 내부 0x3b~0x48 번지에 있는 14 바이트 데이터를 요청하고 있습니다. 0x3B 번지는 가속도 자이로 센서 레지스터의 시작 주소를 나타내고 있습니다. 0x3B 번지에는 ACCEL\_XOUT\_H 레지스터가 있으며, 가속도 센서 X\_Accel 값의 상위 바이트가 저장되는 레지스터입니다. 0x3B 번지부터 시작해 총 14 바이트 크기의 레지스터에 가속도 자이로 센서 값과 온도 센서 값이 저장됩니다. 다음 그림을 참조합니다.

# MPU6050

주소 (16진수)	주소 (10진수)	레지스터
3B	59	ACCEL_XOUT_H
3C	60	ACCEL_XOUT_L
3D	61	ACCEL_YOUT_H
3E	62	ACCEL_YOUT_L
3F	63	ACCEL_ZOUT_H
40	64	ACCEL_ZOUT_L
41	65	TEMP_OUT_H
42	66	TEMP_OUT_L
43	67	GYRO_XOUT_H
44	68	GYRO_XOUT_L
45	69	GYRO_YOUT_H
46	70	GYRO_YOUT_L
47	71	GYRO_ZOUT_H
48	72	GYRO_ZOUT_L

I2C 통신



## 메모리 변수

AcXH
AcXL
AcYH
AcYL
AcZH
AcZL
TmpH
TmpL
GyXH
GyXL
GyYH
GyYL
GyZH
GyZL

각 센서의 값은 16 비트의 크기를 갖습니다.

21~34 : Wire.read() 함수를 이용해 가속도 센서 값, 온도 센서 값, 자이로 센서 값을 읽어내고 있습니다. I2C 통신을 통해서 1 바이트 씩 데이터를 받게 되며, 총 14 바이트의 데이터를 받게 됩니다. 예를 들어, 21,22 줄에서 Wire.read() 함수를 이용해 가속도 센서 X 축 값을 읽어내고 있습니다. I2C 통신을 통해서 1 바이트 씩 데이터를 받게 되며, 총 2 바이트의 데이터를 받게 됩니다. 먼저 온 1 바이트를 AcXH, 나중에 온 1 바이트를 AcXL 변수에 저장합니다.

36 : AcXH의 값을 8비트 왼쪽으로 밀어 상위 8비트의 위치로 놓고, 하위 8비트를 AcXL 값으로 채워 AcX 변수에 저장합니다.

37~41 : 같은 방식으로 AcY, AcZ, GyX, GyY, GyZ 값을 채워넣습니다.

47~52 : 읽어온 센서 값을 출력합니다. %6d는 6자리 십진수로 표현하도록 합니다.

53 : 새 줄을 출력합니다.

43 : 정적 변수 cnt\_loop 변수를 선언합니다. 함수 내에 선언된 정적 변수는 함수를 빠져

나가도 유지되는 변수입니다.

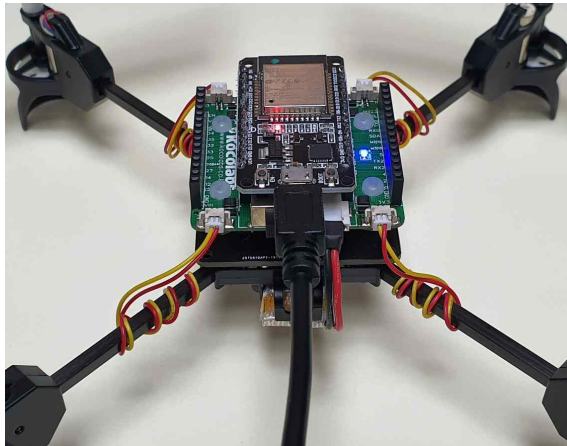
44 : loop 함수를 한 번 수행할 때마다 1씩 증가시킵니다.

45 : cnt\_loop 값이 200의 배수가 아니면 loop 함수를 빠져나가고, 200의 배수이면

47~53 줄을 수행합니다. 예제에서는 loop 함수를 200번 수행할 때마다 한 번 출력합니다.

출력이 너무 빠르면 200보다 큰 값을, 출력이 너무 느리면 200보다 작은 값을 사용합니다.

2. 드론을 다음과 같이 평평한 지면에 놓습니다.



스케치 업로드 시 가속도 자이로 센서가 초기화되는데 센서를 수평 상태로 초기화하기 위해 평평한 면에 놓는 것입니다.

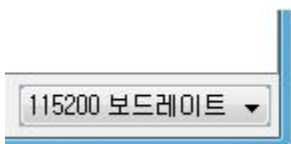
3. 업로드를 수행합니다.



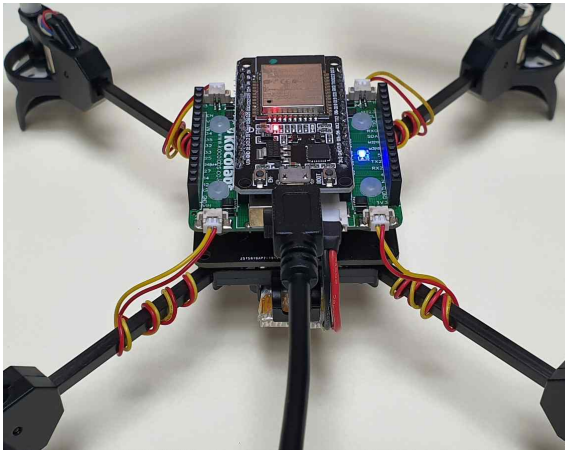
4. 업로드가 완료되면, [시리얼 모니터] 버튼을 클릭합니다.



5. 시리얼 통신 속도를 115200으로 맞추어 줍니다.

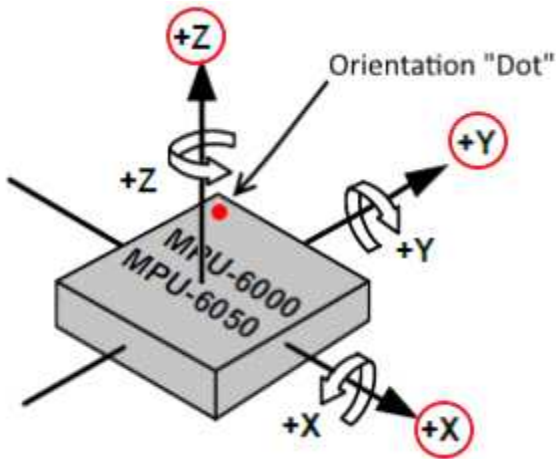


6. 드론을 평평한 지면에 놓은 상태로 시리얼 모니터를 확인해 봅니다.



7. 다음과 같이 센서를 테스트합니다.

먼저 가속도 센서를 테스트합니다.



❶ 직선 화살표 +Z를 지면 위를 향하게 할 때 - AcZ : 16384 근처 값

AcX = -728	AcY = -148	AcZ = 20712	GyX = -348	GyY = 350	GyZ = 173
AcX = -732	AcY = -208	AcZ = 20696	GyX = -305	GyY = 273	GyZ = 169
AcX = -716	AcY = -216	AcZ = 20776	GyX = -381	GyY = 279	GyZ = 187
AcX = -664	AcY = -224	AcZ = 20752	GyX = -308	GyY = 262	GyZ = 169
AcX = -728	AcY = -152	AcZ = 20820	GyX = -365	GyY = 313	GyZ = 169

이 경우 +Z가 중력과 반대 방향입니다. 결과 화면에서 AcZ는 20752~20820 값을 갖습니다. 이 값은 센서를 만드는 과정에서 생성된 오차이며 보정을 통해 사용할 수 있습니다. 독자 여러분의 센서값은 다를 수 있습니다.

❷ 직선 화살표 +Z를 지면 아래를 향하게 할 때 - AcZ : -16384 근처 값

AcX = 408	AcY = -176	AcZ = -12456	GyX = -349	GyY = 268	GyZ = 155
AcX = 488	AcY = -196	AcZ = -12564	GyX = -326	GyY = 251	GyZ = 174
AcX = 444	AcY = -240	AcZ = -12592	GyX = -347	GyY = 280	GyZ = 170
AcX = 352	AcY = -228	AcZ = -12504	GyX = -362	GyY = 234	GyZ = 183
AcX = 504	AcY = -248	AcZ = -12476	GyX = -329	GyY = 280	GyZ = 168

이 경우 +Z가 중력과 같은 방향입니다. 결과 화면에서 AcZ는 -12592~-12456 값을



갖습니다. 이 값은 센서를 만드는 과정에서 생성된 오차이며 보정을 통해 사용할 수 있습니다. 독자 여러분의 센서값은 다를 수 있습니다.

**③ 직선 화살표 +X를 지면 위를 향하게 할 때 - AcX : 16384 근처 값**

AcX = 17080	AcY = -120	AcZ = 3156	GyX = -320	GyY = 268	GyZ = 195
AcX = 17168	AcY = -84	AcZ = 3420	GyX = -361	GyY = 234	GyZ = 186
AcX = 17152	AcY = -76	AcZ = 3448	GyX = -343	GyY = 341	GyZ = 187
AcX = 17060	AcY = -136	AcZ = 3268	GyX = -308	GyY = 239	GyZ = 128
AcX = 16944	AcY = -56	AcZ = 3424	GyX = -330	GyY = 281	GyZ = 166

이 경우 +X가 중력과 반대 방향입니다. 결과 화면에서 AcX는 16944~17168 값을 갖습니다. 이 값은 센서를 만드는 과정에서 생성된 오차이며 보정을 통해 사용할 수 있습니다. 독자 여러분의 센서값은 다를 수 있습니다.

**④ 직선 화살표 +X를 지면 아래를 향하게 할 때 - AcX : -16384 근처 값**

AcX = -15808	AcY = 204	AcZ = 3436	GyX = -343	GyY = 327	GyZ = 164
AcX = -15844	AcY = 172	AcZ = 3168	GyX = -329	GyY = 253	GyZ = 182
AcX = -15816	AcY = 268	AcZ = 3424	GyX = -350	GyY = 304	GyZ = 180
AcX = -15832	AcY = 160	AcZ = 3328	GyX = -314	GyY = 263	GyZ = 179
AcX = -15788	AcY = 160	AcZ = 3392	GyX = -341	GyY = 257	GyZ = 165

이 경우 +X가 중력과 같은 방향입니다. 결과 화면에서 AcX는 -15844~-15788 값을 갖습니다. 이 값은 센서를 만드는 과정에서 생성된 오차이며 보정을 통해 사용할 수 있습니다. 독자 여러분의 센서값은 다를 수 있습니다.

**⑤ 직선 화살표 +Y를 지면 위를 향하게 할 때 - AcY : 16384 근처 값**

AcX = 704	AcY = 16316	AcZ = 2992	GyX = -277	GyY = 268	GyZ = 180
AcX = 860	AcY = 16336	AcZ = 3132	GyX = -322	GyY = 234	GyZ = 195
AcX = 664	AcY = 16308	AcZ = 3184	GyX = -376	GyY = 298	GyZ = 173
AcX = 800	AcY = 16380	AcZ = 3204	GyX = -339	GyY = 275	GyZ = 201
AcX = 760	AcY = 16364	AcZ = 3196	GyX = -389	GyY = 268	GyZ = 153

이 경우 +Y가 중력과 반대 방향입니다. 결과 화면에서 AcY는 16308~16380 값을 갖습니다. 이 값은 센서를 만드는 과정에서 생성된 오차이며 보정을 통해 사용할 수 있습니다. 독자 여러분의 센서값은 다를 수 있습니다.

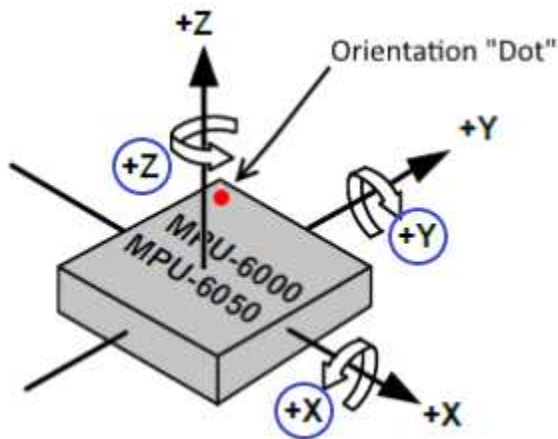
**⑥ 직선 화살표 +Y를 지면 아래를 향하게 할 때 - AcY : -16384 근처 값**

AcX = 520	AcY = -16292	AcZ = 2728	GyX = -419	GyY = 283	GyZ = 170
AcX = 528	AcY = -16392	AcZ = 2728	GyX = -341	GyY = 262	GyZ = 184
AcX = 644	AcY = -16260	AcZ = 2332	GyX = -332	GyY = 319	GyZ = 172
AcX = 672	AcY = -16328	AcZ = 2612	GyX = -340	GyY = 281	GyZ = 160
AcX = 640	AcY = -16352	AcZ = 2612	GyX = -311	GyY = 268	GyZ = 164

이 경우 +Y가 중력과 같은 방향입니다. 결과 화면에서 AcY는 -16392~-16260 값을 갖습니다. 이 값은 센서를 만드는 과정에서 생성된 오차이며 보정을 통해 사용할 수 있습니다. 독자 여러분의 센서값은 다를 수 있습니다.

다음은 자이로 센서를 테스트합니다.





⑦ 곡선 화살표 +Z축 +방향 또는 -방향 회전 시킬 때 - GyZ : -32768 ~ 32767

AcX = -1600	AcY = -1108	AcZ = 19288	GyX = 404	GyY = 1901	GyZ = -17380
AcX = -1212	AcY = -3476	AcZ = 22924	GyX = 4694	GyY = -45	GyZ = -32768
AcX = 1244	AcY = 860	AcZ = 22648	GyX = 5193	GyY = -3349	GyZ = -32768
AcX = -2856	AcY = 3196	AcZ = 22116	GyX = 113	GyY = 1315	GyZ = 745
AcX = -2636	AcY = 148	AcZ = 23468	GyX = -5681	GyY = -2850	GyZ = 32767

드론을 수평 상태에서 방향을 바꾸는 회전입니다. +방향 - 양수, -방향 - 음수

⑧ 곡선 화살표 +X축 +방향 또는 -방향 회전 시킬 때 - GyX : -32768 ~ 32767

AcX = 112	AcY = 5524	AcZ = 18212	GyX = 32767	GyY = 1850	GyZ = -6915
AcX = 1020	AcY = 8416	AcZ = 13780	GyX = -2316	GyY = 2322	GyZ = 504
AcX = -2428	AcY = 9092	AcZ = 16960	GyX = -27436	GyY = 3090	GyZ = 2390
AcX = -3464	AcY = -5752	AcZ = 23512	GyX = -24831	GyY = -656	GyZ = 4816
AcX = -3492	AcY = -8344	AcZ = 17324	GyX = 6880	GyY = 3745	GyZ = -404

드론을 수평 상태에서 좌우로 기울이는 회전입니다. +방향 - 양수, -방향 - 음수

⑨ 곡선 화살표 +Y축 +방향 또는 -방향 회전 시킬 때 - GyY : -32768 ~ 32767

AcX = -2248	AcY = -180	AcZ = 21232	GyX = -6734	GyY = 32767	GyZ = -4192
AcX = -3260	AcY = -620	AcZ = 19304	GyX = 543	GyY = 10169	GyZ = -2221
AcX = -4432	AcY = 112	AcZ = 21824	GyX = 5425	GyY = -32626	GyZ = 4914
AcX = -2228	AcY = 1148	AcZ = 22056	GyX = 3370	GyY = -24592	GyZ = 2743
AcX = -4048	AcY = -160	AcZ = 20208	GyX = -8749	GyY = 32767	GyZ = -4267

드론을 수평 상태에서 전후로 기울이는 회전입니다. +방향 - 양수, -방향 - 음수

\*\*\* 센서 테스트 시 이상적으로 가속도 자이로 센서의 값이 0이 나와야 하지만 실제로는 그렇지 않습니다. 이 값들이 0에 가까워지도록 뒤에서 보정의 과정을 수행합니다.

## 07 가속도 자이로 센서 값 해석하기

여기서는 MPU6050 센서로부터 전달되는 가속도 자이로 값의 의미를 살펴보고자 합니다.

MPU6050 센서를 통해 얻게 되는 가속도 자이로 센서 값은 16 비트 크기를 갖습니다. 그

래서 이전 예제에서 가속도 자이로 센서의 값을 저장했던  $AcX$ ,  $AcY$ ,  $AcZ$ ,  $GyX$ ,  $GyY$ ,  $GyZ$  변수는 `int16_t` 타입의 16 비트 크기 변수입니다. 16 비트 변수를 통해 표현할 수 있는 숫자는  $-32768 \sim 32767$  사이의 정수 값입니다. 즉, 최소  $-32768$ 에서 최대  $32767$  사이의 정수 값을 표현할 수 있습니다.

## 가속도 센서 값 해석하기

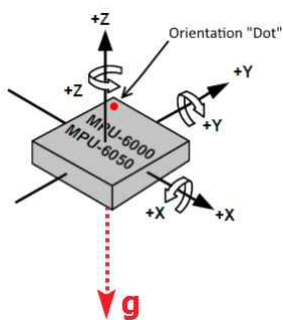
가속도 센서를 통해 얻게 되는 값에 대해서 먼저 생각해 보도록 하겠습니다.

$AcX$ ,  $AcY$ ,  $AcZ$ 는 최저  $-32768 \sim 32767$  사이의 값을 가질 수 있습니다. 그러면 이 값들은 무엇을 의미할까요? 다음 표를 통해 그 의미를 알아보도록 하겠습니다.

AFS_SEL 레지스터 값	최대 표현 범위	g 당 가속도 센서 값
0	$\pm 2g$	16384/g
1	$\pm 4g$	8192/g
2	$\pm 8g$	4096/g
3	$\pm 16g$	2048/g

AFS\_SEL는 MPU6050 센서 내부의 레지스터입니다. 이 레지스터 값에 따라 센서 값의 의미는 달라집니다. 예를 들어, AFS\_SEL의 값이 0으로 설정되어 있을 때에는  $-32768 \sim 32767$  사이의 값은  $-2g \sim +2g$  사이의 값을 의미합니다. 여기서  $g$ 는 중력 가속도를 나타냅니다. 즉,  $-32768$ 은  $-2g$ ,  $32767$ 은  $+2g$ 를 의미합니다.

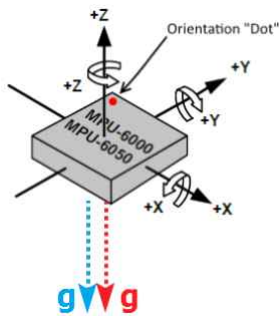
그러면  $1g$ 는 얼마일까요? 바로 16384가 됩니다. AFS\_SEL 레지스터의 기본 설정 값은 0이며, 우리는 현재 이 값을 사용하고 있습니다. 그래서 드론을 수평으로 놓은 상태에서는  $AcZ$ 의 값이 16384 근처의 값이 나오게 됩니다. 센서의 위치에 따라 오차는 있을 수 있습니다.



그림에서 직선 화살표  $+Z$ 가 중력 가속도  $g$ 와 반대 방향일 때  $AcZ$ 의 값은  $g$ 에 해당하는 크기의 양수 값을 갖게 됩니다. 즉, 16384 값을 갖습니다. 반대로  $+Z$ 가 중력 가속도  $g$ 와 같은 방향일 때  $AcZ$ 의 값은  $g$ 에 해당하는 크기의 음수 값을 갖게 됩니다. 즉,  $-16384$  값을 갖습니다.  $+X$ ,  $+Y$ 도 마찬가지입니다.

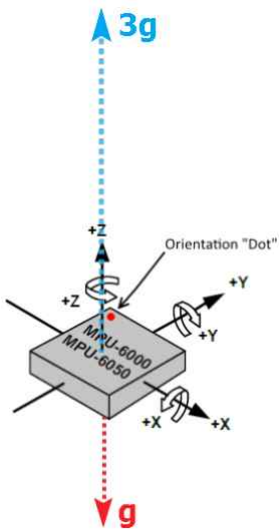
그러면  $AcZ$ 가 32767 값을 갖는 경우는 어떤 경우일까요? 다음 그림을 보면서 이해해 보도

록 하겠습니다.



그림에서 직선 화살표 +Z 반대 방향으로 2g 이상의 가속도가 생길 때 AcZ의 값은 2g에 해당하는 크기의 양수 값을 갖게 됩니다. 즉, 32767 값을 갖습니다. 센서를 지면에 수평 상태로 중력 가속도 방향으로 g 만큼의 가속도를 주게 되면 AcZ의 값은 32767 값을 갖게 됩니다.

AcZ가 -32768의 값을 갖는 경우에 대해서도 생각해 보겠습니다. 다음 그림을 보면서 이해해 보도록 하겠습니다.



센서를 지면에 수평 상태로 둔 채, 직선 화살표 +Z 방향으로 3g에 해당하는 가속도를 주게 되면 중력 가속도 g와 상쇄된 후, 2g에 해당하는 가속도가 +Z 방향으로 작용하게 됩니다. 가속도의 방향과 +Z 축의 방향이 같으므로, AcZ 값은 -2g에 해당하는 값을 갖게 됩니다. 즉, AcZ의 값은 -32768의 값을 갖습니다. +X, +Y도 마찬가지입니다.

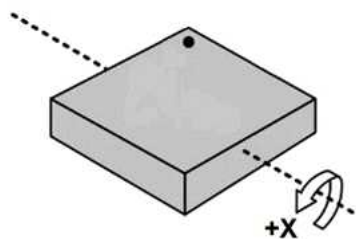
## 자이로 센서 값 해석하기

다음은 자이로 센서를 통해 얻게 되는 값에 대해서 생각해 보도록 하겠습니다.

GyX는 최저 -32768 ~ 32767 사이의 값을 가질 수 있습니다. 그러면 이 값들은 무엇을 의미할까요? 다음 표를 통해 그 의미를 알아보도록 하겠습니다.

FS_SEL 레지스터 값	최대 표현 범위	°/s 당 자이로 센서 값
0	$\pm 250$ °/s	131/°/s
1	$\pm 500$ °/s	65.5/°/s
2	$\pm 1000$ °/s	32.8/°/s
3	$\pm 2000$ °/s	16.4/°/s

FS\_SEL는 MPU6050 센서 내부의 레지스터입니다. 이 레지스터 값에 따라 센서 값의 의미는 달라집니다. 예를 들어, FS\_SEL의 값이 0으로 설정되어 있을 때에는 -32768 ~ 32767 사이의 값은 -250°/s ~ +250°/s 사이의 값을 의미합니다. 여기서 °/s는 각속도를 나타냅니다. 즉, -32768은 -250°/s, 32767은 +250°/s를 의미합니다. 다음 그림을 보면서 좀 더 이해해 보도록 하겠습니다.



그림에서 곡선 축 +X 방향으로 1초 동안 일정한 회전 속도(각속도)로 250도 회전했을 때, GyX의 값은 1초 동안 계속해서 250°/s에 해당하는 크기의 양수 값을 갖게 됩니다. 즉, 1초 동안 계속해서 32767 값을 유지하게 됩니다. 반대로 곡선 축 +X 반대 방향으로 1초 동안 일정한 회전 속도로 250도 회전했을 때, GyX의 값은 1초 동안 계속해서 250°/s에 해당하는 크기의 음수 값을 갖게 됩니다. 즉, 1초 동안 계속해서 -32768 값을 유지하게 됩니다. +Y, +Z도 마찬가지입니다.

그러면 곡선 축 +X 방향으로 1초 동안 일정한 회전 속도로 1도 회전했을 때, GyX는 어떤 값을 유지하고 있을까요? 다음 식을 보면서 이해해 보도록 합니다.

$250^\circ/\text{s} == 32767 \text{ 이므로 } 1^\circ/\text{s} == (32767/250) == 131$
---

1초 동안 250도 회전할 경우에 GyX의 값이 32767이라면, 1초 동안 1도 회전할 경우의 GyX는 (32768/250) 값을 유지하게 됩니다. 이 값은 바로 131입니다.

FS\_SEL 레지스터의 기본 설정 값은 0이며, 우리는 현재 이 값을 사용하고 있습니다.

250°/s는 생각보다 빠르지는 않은 속도입니다. 1 초 동안 한 바퀴를 돌지 못하는 회전 속도이기 때문입니다.

자이로 센서를 지면에 둔 상태로 시계의 초침과 같은 속도와 방향으로 자이로 센서가 수평 회전하는 경우를 생각해 보도록 하겠습니다.



시계의 초침의 경우엔  $360^\circ/60s$ 이므로  $6^\circ/s$ 의 각속도를 갖게 됩니다. 또 시계 방향(곡선 축 +Z 반대 방향)으로 회전을 하게 됩니다. 따라서 GyZ의 값은  $6^\circ/s$ 에 해당하는 크기의 음수 값을 갖게 됩니다. 다음 식을 통해 GyZ의 값을 정해 보도록 하겠습니다.

$$1^\circ/s == 131 \text{ 이므로 } 6^\circ/s == 131 \times 6 == 786$$

즉, 자이로 센서가 시계의 초침과 같은 속도로 반시계 방향(곡선 축 +Z 방향)으로 회전할 경우에 GyZ의 레지스터 값은 -786 값이 됩니다.

그러면 자이로 센서가 다음과 같은 조건으로 회전했을 경우 1 초 후에 몇 도 회전해 있을 까요?

(곡선 축 +Z 방향으로 회전하는 경우)

0.0~0.1 초 동안 $1^\circ/s$
0.1~0.2 초 동안 $2^\circ/s$
0.2~0.3 초 동안 $3^\circ/s$
0.3~0.4 초 동안 $4^\circ/s$
0.4~0.5 초 동안 $5^\circ/s$
0.5~0.6 초 동안 $6^\circ/s$
0.6~0.7 초 동안 $7^\circ/s$
0.7~0.8 초 동안 $8^\circ/s$
0.8~0.9 초 동안 $9^\circ/s$
0.9~1.0 초 동안 $10^\circ/s$

다음과 같이 계산합니다.

0.0~0.1 초 동안 $1^\circ/s = 1^\circ/s \times 0.1s = 0.1^\circ$
0.1~0.2 초 동안 $2^\circ/s = 2^\circ/s \times 0.1s = 0.2^\circ$
0.2~0.3 초 동안 $3^\circ/s = 3^\circ/s \times 0.1s = 0.3^\circ$
0.3~0.4 초 동안 $4^\circ/s = 4^\circ/s \times 0.1s = 0.4^\circ$
0.4~0.5 초 동안 $5^\circ/s = 5^\circ/s \times 0.1s = 0.5^\circ$

0.5~0.6 초 동안 $6^{\circ}/s = 6^{\circ}/s \times 0.1s = 0.6^{\circ}$ 0.6~0.7 초 동안 $7^{\circ}/s = 7^{\circ}/s \times 0.1s = 0.7^{\circ}$ 0.7~0.8 초 동안 $8^{\circ}/s = 8^{\circ}/s \times 0.1s = 0.8^{\circ}$ 0.8~0.9 초 동안 $9^{\circ}/s = 9^{\circ}/s \times 0.1s = 0.9^{\circ}$ 0.9~1.0 초 동안 $10^{\circ}/s = 10^{\circ}/s \times 0.1s = 1.0^{\circ}$
--

1.0 초 후에는 최초 위치로부터 좌측으로  $5.5^{\circ}$  회전해 있게 됩니다.

이 방법을 사용하면 각속도와 자이로 센서 측정 주기 시간을 이용해 자이로 센서가 회전한 각도를 구할 수 있습니다.

자이로 센서는 각속도를 측정합니다. 그래서 곡선 축 +X, +Y, +Z 방향을 기준으로 각속도 ( $\omega$ )를 측정해 측정 주기 시간( $\Delta t$ )과 곱해서 변화된 각을 계산할 수 있습니다. 변화 각은 다음과 같습니다.

$$\Delta\theta = \omega \times \Delta t \quad (\Delta\theta : \text{미세 회전 각도}, \omega : \text{회전 각속도}, \Delta t : \text{주기})$$

새로운 방향각은 이전 각에 이 변화된 각을 더해 얻어집니다. 현재 각도를 구하는 식은 다음과 같습니다.

$$\theta_{now} = \theta_{prev} + \omega \times \Delta t \quad (\theta_{now} : \text{현재 각도}, \theta_{prev} : \text{이전 각도})$$

즉, 많은 미세 변화 각( $\Delta\theta$ )을 누적하여 현재의 각도를 구할 수 있습니다.