

Chapter 112. 서보 모터

1. 모터

모터는 바퀴, 팔, 다리, 손가락 등의 움직임을 구현하기 위해 필수적인 장치로 다양한 종류의 모터가 존재하지만 마이크로컨트롤러에서는 DC 모터, 스텝핑 모터, 서보 모터 등이 많이 사용된다. 각각의 모터들은 그 특징이 조금씩 다르므로 사용하고자 하는 목적에 맞는 모터를 선택하는 것이 중요하다. 각 모터의 특징은 다음과 같다.

- DC (Direct Current) 모터 : 축이 연속적으로 회전하는 형태의 모터로 전원이 끊어지는 경우에만 정지한다. 정지 시에는 관성으로 인해 정확한 정지 위치를 지정하기 어렵다.
- 스텝핑 (stepping) 모터 : 전원이 공급되면 축은 일정 각도를 회전하고 멈춘다. 축을 연속적으로 회전시키기 위해서는 모터로 펄스열이 전달되어야 하며 하나의 펄스에 반응하여 회전하는 양을 분할 각도라고 한다. 분할 각도가 1도인 경우 모터를 10도 회전시키기 위해서는 10개의 펄스를 전송해주어야 하는 식이다. 스텝핑 모터는 제어하기가 쉬운 장점은 있지만 분할 각도 단위로만 제어할 수 있는 한계가 있다. 즉, 분할 각도가 1도인 경우 9.5도 회전은 불가능하다.
- 서보 (servo) 모터 : DC 모터의 한 종류로 DC 모터에 귀환 제어 회로를 추가하여 정확한 위치 제어가 가능하도록 구성된 모터이다. 제어 회로로 인해 스텝핑 모터에 비해 비싼 단점은 있지만 정밀 제어가 가능하고, 오동작을 수정할 수 있으며, 속도 면에서도 스텝핑 모터에 비해 빠른 장점이 있다.

이 장에서는 서보 모터를 제어하는 방법에 대해 알아본다.

2. 서보 모터

서보 모터는 DC 모터의 한 종류로 PWM 입력으로 원하는 만큼 모터를 회전 시킨 후 위치를 유지할 수 있는 특징을 갖는다. 서보 모터는 크게 표준 서보 모터와 연속 회전 서보 모터의 두 종류로 나누어 볼 수 있다. 표준(standard) 서보 모터는 회전 범위가 제한된 서보 모터로 일반적으로 $0^{\circ} \sim 180^{\circ}$ 범위에서만 회전할 수 있어 로봇의 관절 또는 회전 범위가 제한된 기기의 움직임을 제어하기 위해 사용한다. 이에 비해 연속 회전 (continuous rotation) 서보 모터

는 일반 모터와 같이 360°를 무한 회전할 수 있다. 이 장에서 사용할 서보 모터는 표준 서보 모터이다.



그림 1. 표준 마이크로 서보 모터

서보 모터는 다른 종류의 모터와 다르게 제어가 간단하다. 서보 모터는 일반적으로 3개의 연결선을 (VCC, GND 그리고 제어선) 가지며 VCC는 붉은색, GND는 검정색이나 갈색, 제어선은 노란색, 주황색 또는 흰색으로 구별된다. DC 모터의 경우 VCC와 GND의 구별이 없으며 반대로 연결하는 경우 반대 방향으로 모터가 회전하지만, 서보 모터의 경우에는 전원을 반대로 연결하면 모터가 파손될 수 있으므로 주의하여야 한다.

서보 모터의 경우에는 전용 전원을 모터에 연결하고 제어선으로 위치를 조절하는 간단한 구성으로 제어가 가능하다. 아두이노 우노 보드의 레귤레이터는 1A 까지 전류를 공급할 수 있으므로 0.2A 정도의 전류를 필요로 하는 마이크로 서보 모터의 경우에는 아두이노 우노 보드의 5V 전원을 연결하여 사용할 수 있다. 하지만 모터는 전기 잡음을 유발할 수 있으므로 보드의 안정적인 동작을 위해 모터에는 전용 전원을 공급하는 것이 바람직하다. 특히 어댑터가 아닌 USB 연결을 통해 전원을 공급하는 경우에는 최대 0.5A의 전류만을 공급할 수 있으므로 전류 부족으로 보드가 정상적으로 동작하지 않을 수 있으므로 주의하여야 한다. 서보 모터를 위해 전용 전원을 사용하는 경우 서보 모터의 전원과 아두이노 우노 보드를 위한 전원의 GND는 반드시 연결해주어야 한다는 점도 잊지 말아야 한다.

서보 모터의 위치 제어는 PWM 신호에 의해 이루어지므로 타이머/카운터의 PWM 신호 생성 기능을 이용하여 제어할 수 있다. 서보 모터는 PWM 신호를 받으면 내부적으로 현재 위치와 입력된 신호를 비교하여 입력된 PWM 신호에 맞는 위치로 모터를 회전시킨다. 즉, 듀티 사이클에 따라 서보 모터의 위치가 정해진다. 서보 모터는 50Hz의 PWM 주파수, 즉 주기

$\frac{1}{50Hz} = 20ms$ 의 PWM 신호를 필요로 한다. 20ms 중 서보 모터의 위치를 결정하는 HIGH 구간은 1~2ms로, 1ms에서는 반시계 방향으로 최대로 회전한 상태(0°)이며 2ms에서는 시계 방향으로 최대로 회전한 상태(180°)에 있게 된다. 서보 모터의 종류에 따라 회전할 수 있는 각도의 범위와 이에 따른 듀티 사이클에 다소 차이가 있을 수 있으므로 사용하고자 하는 서보 모터의 데이터시트를 확인하여야 한다.

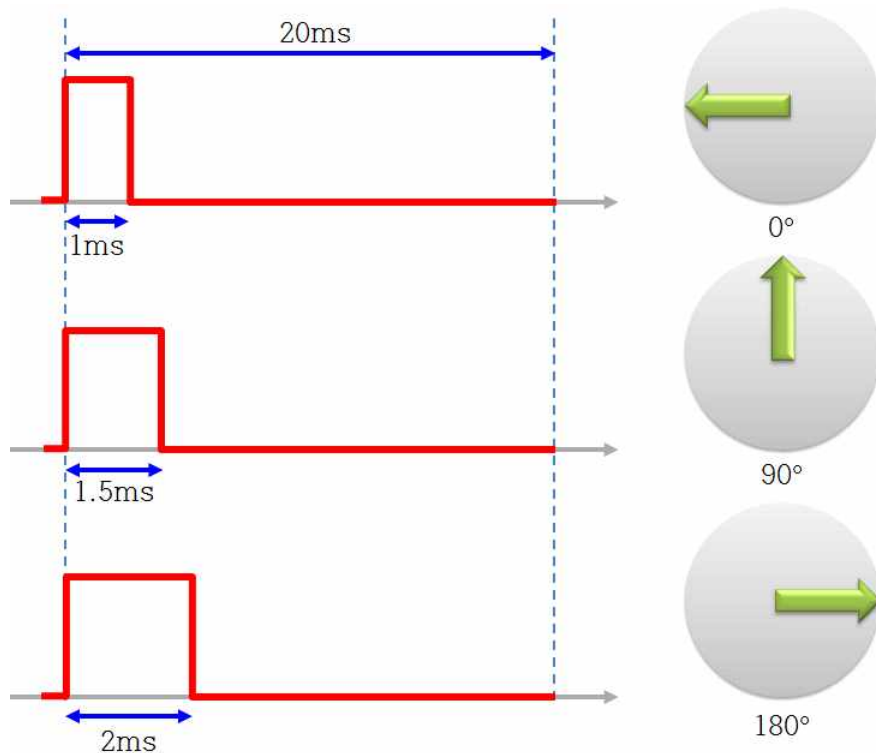


그림 2. 서보 모터 제어

서보 모터에 20ms 주기의 펄스를 입력하면 다음 20ms 동안만 이동한 위치를 유지하므로 매 20ms 마다 펄스를 계속해서 입력으로 주어야 한다. 즉, 1초에 50개의 펄스를 지속적으로 가해주어야 현재 위치를 유지할 수 있다.

한 가지 유의해야 할 점은 20ms 주기에서 1ms 펄스는 듀티 사이클로는 5%이며, 2ms 펄스는 듀티 사이클로 10%라는 점이다. 즉, 5%의 듀티 사이클 변화에 의해 180도 회전이 이루어진다는 점이다. 크게 문제가 될 것이 없어 보이는가? 8비트 타이머를 생각해 보자. 5% 듀티 사이클 변화는 256 단계 중 약 12 단계($256 \times 0.05 = 12.8$)에 해당한다. 타이머/카운터에서 듀티 사이클로 지정할 수 있는 값은 정수뿐이므로 1 단계 변화는 약 14도 회전($180/12.8 \approx 14$)에 해당한다. 정밀한 회전각 제어가 필요하지 않다면 8비트 타이머로 충분할 수 있지만 한 번

에 변경할 수 있는 최소 각도가 14도라면 그리 많은 용도로 사용하기는 어려울 수 있다. 16비트 타이머의 경우에는 한 번에 변경할 수 있는 최소 각도는 약 0.055도로 ($\frac{180}{2^{16} \times 0.05} \approx 0.0549164$) 이는 서보 모터가 표현할 수 있는 최소 각도 차이 이상의 값에 해당한다. 서보 모터는 현재 위치와 입력 받은 PWM 신호에 의한 위치 차이가 아주 작은 경우 동작하지 않는다. 즉, 모터 자체에서 표현할 수 있는 최소 각도 차이가 정해져 있다. 이처럼 현재 서보 모터의 위치에서 근소한 차이의 PWM 신호가 입력되어도 서보 모터가 동작하지 않는 영역을 데드 밴드(dead band)라고 하며 수 마이크로초에서 수십 마이크로초에 이르기까지 서보 모터의 종류에 따라 다른 값을 가진다. 데드 밴드가 5 마이크로초라고 가정해 보자. PWM 신호의 펄스 폭은 최대 1ms의 차이를 가지며 이를 통해 180도 회전을 표현하므로 $5\mu s$ 에 해당하는 각도는 $\frac{5}{1000} \times 180 = 0.9$ 도가 된다. 이 각도는 16비트 타이머/카운터를 이용해야만 충분히 표현할 수 있으며 이 장에서도 1번 타이머/카운터를 사용하여 서보 모터를 제어할 것이다.

3. 서보 모터 제어

먼저 서보 모터를 아두이노 우노 보드에 연결한다. 1번 타이머/카운터의 OC1A 출력 핀(PB1 핀, 아두이노 9번 핀)을 통해 서보 모터 제어를 위한 PWM 신호를 출력하도록 할 것이다.

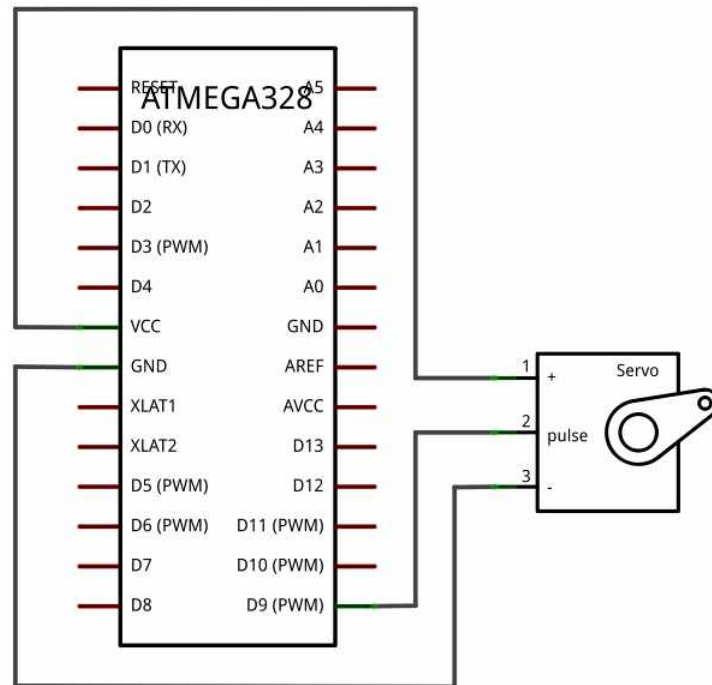


그림 3. 서보 모터 연결 회로도

코드 1은 50Hz PWM 주파수를 이용하여 서보 모터를 0도에서 180도까지 회전시키는 예이다.

코드 1. 서보 모터 위치 제어

```
#define F_CPU 16000000L
#include <avr/io.h>
#include <util/delay.h>

#define ROTATION_DELAY 1000 // 1초 대기
#define PULSE_MIN 1363 // 최소 펄스 길이
#define PULSE_MID 3146
#define PULSE_MAX 4928 // 최대 펄스 길이

void INIT_TIMER1(void)
{
    // Fast PWM 모드, TOP = ICR1
    TCCR1A |= (1 << WGM11);
    TCCR1B |= (1 << WGM12) | (1 << WGM13);
```

```
TCCR1B |= (1 << CS11);           // 분주율 8, 2MHz

ICR1 = 41942;                     // 20ms 주기

TCCR1A |= (1 << COM1A1);          // 비반전 모드

// OC1A (PB1, 아두이노 9번) 핀을 출력으로 설정
DDRB |= (1 << PB1);
}

int main(void)
{
    INIT_TIMER1();

    while(1)
    {
        OCR1A = PULSE_MIN;         // 0도
        _delay_ms(ROTATION_DELAY);

        OCR1A = PULSE_MID;         // 90도
        _delay_ms(ROTATION_DELAY);

        OCR1A = PULSE_MAX;         // 180도
        _delay_ms(ROTATION_DELAY);
    }
}
```

코드 1은 고속 PWM 모드를 사용한다. 8비트인 0번과 2번 타이머/카운터의 경우 WGM (Waveform Generation Mode) 비트를 통해 8개의 모드를 선택할 수 있는 반면, 16비트인 1번 타이머/카운터의 경우 WGM 비트가 4비트로 16개의 모드 중 선택할 수 있다. WGM 비트는 TCCR1B 레지스터와 TCCR1A 레지스터에 각각 2비트씩 포함되어 있다.

모드	TCCR1B		TCCR1A		설명	TOP
	WGM13	WGM12	WGM11	WGM10		
0	0	0	0	0	정상 모드 (디폴트 모드)	0xFFFF
1	0	0	0	1	8비트 위상 교정 PWM 모드	0x00FF
2	0	0	1	0	9비트 위상 교정 PWM 모드	0x01FF
3	0	0	1	1	10비트 위상 교정 PWM 모드	0x03FF
4	0	1	0	0	CTC 모드	OCR1A
5	0	1	0	1	8비트 고속 PWM 모드	0x00FF
6	0	1	1	0	9비트 고속 PWM 모드	0x01FF
7	0	1	1	1	10비트 고속 PWM 모드	0x03FF
8	1	0	0	0	위상 및 주파수 교정 PWM 모드	ICR1
9	1	0	0	1	위상 및 주파수 교정 PWM 모드	OCR1A
10	1	0	1	0	위상 교정 PWM 모드	ICR1
11	1	0	1	1	위상 교정 PWM 모드	OCR1A
12	1	1	0	0	CTC 모드	ICR1
13	1	1	0	1	-	-
14	1	1	1	0	고속 PWM 모드	ICR1
15	1	1	1	1	고속 PWM 모드	OCR1A

표 1. WGM 비트 설정에 따른 1번 타이머/카운터의 파형 생성 모드

1번 타이머/카운터의 경우 다른 타이머/카운터와 다르게 16비트이므로 여러 가지 다양한 모드를 지원하고 있다. 1번 타이머/카운터는 16비트 전체를 사용할 수도 있지만 필요한 만큼의 비트 수를 선택해서 사용할 수도 있다. 또한 8비트 타이머/카운터에서는 볼 수 없었던 ICR1 (Input Capture Register) 레지스터가 존재한다. ICR 레지스터는 16비트 레지스터로 ICP1 핀 (PB0 핀, 아두이노 8번 핀)에 특정 신호가 입력될 때 현재 카운터의 값을 저장할 수 있도록 해주므로 입력 펄스의 길이를 측정하는 용도로 사용할 수 있다. ICR1 레지스터는 OCR1A 레지스터와 마찬가지로 카운터의 TOP 값을 지정할 수 있도록 해주며 코드 1에서도 이러한 용도로 사용하였다. 입력 캡처는 뒤에서 다시 다룬다.

코드 1에서는 14번 파형 생성 모드를 사용하므로 TOP 값은 ICR1 레지스터에 저장되고 비교 일치 값은 OCR1A 레지스터에 저장된다. 아두이노 우노는 16MHz 클록(f_{clk})을 사용하고 분주율(N)을 8로 설정하였으므로 타이머/카운터에 공급되는 클록은 2MHz가 된다. 고속 PWM 모드에서 PWM 주파수는 식 (1)과 같이 계산된다.

$$f_{PWM} = \frac{f_{clk}}{N \cdot (1 + TOP)} \quad (1)$$

따라서 50Hz의 PWM 주파수를 얻기 위한 TOP 값은 식 (2)와 같이 계산할 수 있다.

$$TOP = \frac{f_{clk}}{f_{PWM} \cdot N} - 1 = \frac{16M}{50 \cdot 8} - 1 \approx 41942 \quad (2)$$

TOP 값의 5%에 해당하는 값이 최소 펄스폭에 해당하는 비교일치 값이며, TOP 값의 10%에 해당하는 값이 최대 펄스폭에 해당하는 비교일치 값이다. 하지만 서보 모터의 종류에 따라 듀티 사이클과 최대 회전 각도는 달라질 수 있으며, 실험에 사용한 서보 모터의 경우 3.25%에서 11.75% 듀티 사이클에서 0도에서 170도 범위를 회전하였다.

$$PULSE\ MIN = 41942 \times 0.0323 \approx 1363 \quad (3)$$

$$PULSE\ MAX = 41952 \times 0.1175 \approx 4928 \quad (4)$$

코드 2는 가변저항을 이용하여 서보 모터의 위치를 제어하는 코드의 예이다. 먼저 그림 4와 같이 서보 모터와 가변저항을 연결한다.

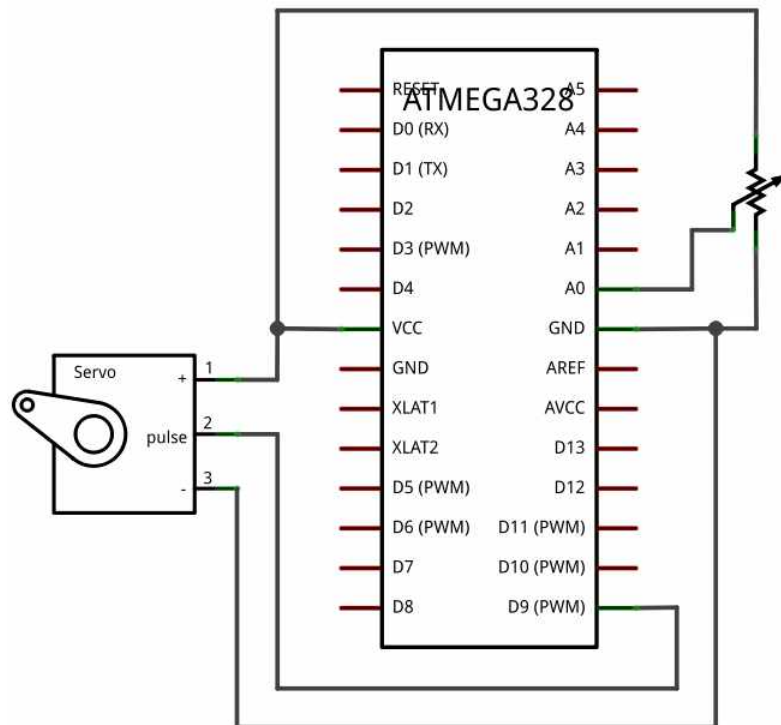


그림 4. 서보 모터와 가변저항 연결 회로도

가변저항은 채널 0번에 연결하였고 서보 모터는 코드 1에서와 동일하게 타이머 1번을 이용하여 제어한다.

코드 2. 가변저항을 이용한 서보 모터 위치 제어

```
#define F_CPU 16000000L
#include <avr/io.h>
#include <util/delay.h>

#define ROTATION_DELAY 10 // 10밀리초 대기
#define PULSE_MIN 1363 // 최소 펄스 길이
#define PULSE_MAX 4928 // 최대 펄스 길이

void INIT_TIMER1(void)
{
    // Fast PWM 모드, TOP = ICR1
    TCCR1A |= (1 << WGM11);
```

```
TCCR1B |= (1 << WGM12) | (1 << WGM13);

TCCR1B |= (1 << CS11);           // 분주율 8, 2MHz

ICR1 = 41942;                    // 20ms 주기

TCCR1A |= (1 << COM1A1);         // 비반전 모드

// OC1A (PB1, 아두이노 9번) 핀을 출력으로 설정
DDRB |= (1 << PB1);
}

void ADC_INIT(unsigned char channel)
{
    ADMUX |= 0x40;                // AVCC를 기준 전압으로 선택

    ADCSRA |= 0x07;               // 분주비 설정
    ADCSRA |= (1 << ADEN);        // ADC 활성화
    ADCSRA |= (1 << ADSC);        // 자동 트리거 모드

    ADMUX |= ((ADMUX & 0xE0) | channel); // 채널 선택
    ADCSRA |= (1 << ADSC);        // 변환 시작
}

int read_ADC(void)
{
    while(!(ADCSRA & (1 << ADIF))); // 변환 종료 대기

    return ADC;                   // 10비트 값을 반환
}

int main(void)
{
    int read;

    ADC_INIT(0);                  // AD 변환기 초기화
```

```

INIT_TIMER1();

while(1)
{
    read = read_ADC();          // 가변저항 값 읽기
    OCR1A = PULSE_MIN + (int)(3.48 * read);

    _delay_ms(ROTATION_DELAY);
}
}

```

코드 2에서는 서보 모터의 위치 결정을 위해 PULSE_MIN에서 PULSE_MAX 사이의 값을 1024 단계로 나누어 0에서 1023까지의 AD 변환 값에 대응되도록 하였다.

$$\begin{aligned}
 &PULSE_MIN + \frac{PULSE_MAX - PULSE_MIN}{1024} \times read \\
 &\approx PULSE_MIN + 3.48 \times read
 \end{aligned}
 \tag{3}$$

4. 아두이노의 서보 모터 제어

아두이노에는 서보 모터 제어를 위한 전용 라이브러리로 서보(Servo) 라이브러리를 제공하고 있다. 아두이노 우노의 경우 서보 라이브러리는 1번 타이머/카운터를 사용하여 12개까지의 서보 모터를 사용할 수 있도록 해준다. 서보 모터를 연결하는 핀은 타이머/카운터와 연관된 파형 출력 핀(OC1x: x = A, B)이 아니어도 무방하며 서보 라이브러리에서 인터럽트를 통해 제어한다. 따라서 서보 라이브러리를 사용하는 경우에는 파형 출력 핀인 OC1A 핀(PB1, 아두이노 9번)과 OC1B 핀(PB2, 아두이노 10번)의 PWM 출력 기능은 사용할 수 없다.

서보 모터를 사용하기 위해서는 먼저 '프로젝트 → Add/Import Sketch Library → Core → Servo' 메뉴를 선택하여 서보 라이브러리 사용을 위한 헤더 파일을 추가한다. 다음으로는 연결하고자 하는 서보 모터별로 Servo 클래스의 객체를 생성한다.

```
#include <Servo.h>
```

```
Servo myServo;          // 서보 모터 제어를 위한 변수
```

서보 모터 설정을 위해서는 attach 함수를 사용하면 된다.

```
uint8_t attach(int pin)
```

```
uint8_t attach(int pin, int min, int max)
```

매개변수

pin : 서보의 데이터 선이 연결된 핀 번호

min : 0°에 해당하는 마이크로초 단위의 펄스 폭 (디폴트값은 544)

max : 180°에 해당하는 마이크로초 단위의 펄스 폭 (디폴트값은 2400)

반환값 : 현재 서보 모터가 연결된 채널

서보 모터를 지정한 핀에 연결하고 현재 서보 모터가 연결된 채널을 반환한다. 아두이노 우노의 경우 12개까지 서보 모터를 연결할 수 있으므로 연결에 성공하면 1~12 사이의 값을 반환하며 연결할 수 있는 개수를 초과하는 경우 255를 반환한다. min과 max는 PWM 신호의 최소 및 최대 폭을 지정하기 위해 사용한다. 모터에 따라서는 최대 회전 각도가 0°~180° 범위와 차이가 날 수 있으며 해당 PWM 신호의 폭 역시 다를 수 있으므로 모터에 따라 설정할 필요가 있을 때 사용한다. 실험에 사용한 서보 모터의 경우 디폴트값에서 0°에서 170°로 동작하는데 문제가 없었다.

서보 모터가 연결되면 모터의 위치는 write 함수에 의해 설정이 가능하며, 현재 서보 모터의 위치는 read 함수에 의해 알아낼 수 있다.

```
void write(int value)
```

매개변수

value : 서보 모터의 위치 값 (0 ~ 180 사이의 값)

반환값 : 없음

```
int read(void)
    매개변수 : 없음
    반환값 : 서보 모터의 현재 축 위치 (0 ~ 180 사이의 값)
```

코드 3은 그림 4의 회로를 이용하여 코드 2와 동일한 동작을 하도록 아두이노 스타일로 작성한 코드이다.

코드 3. 가변저항을 이용한 서보 모터 위치 제어 - 아두이노 스타일

```
#include <Servo.h>                // 서보 라이브러리
#define ROTATION_DELAY      10    // 10밀리초 대기

Servo microServo;                 // 서보 모터를 위한 객체
int servoPin = 9;                 // 서보 연결 핀

void setup(){
    microServo.attach(servoPin);  // 서보 모터 연결
}

void loop(){
    int analogValue = analogRead(A0); // 가변저항 읽기
    // 가변저항 값을 각도로 변환
    int angle = map(analogValue, 0, 1023, 0, 180);

    microServo.write(angle);       // 모터 위치 제어
    delay(ROTATION_DELAY);
}
```