온라인 교재

아두이노 로봇(ABOT) 가지고 놀기

(원문: Robotics with the Board of Education Shield for Arduino)

제1부 / 제2부

저자: Andy Lindsay from Parallax.com
번역: 심재창 / 고주영 / 정욱진 / 이영학

원문:Parallax.com 온라인 배포자료(http://learn.parallax.com/ShieldRobot)
번역:심재창/고주영/이영학/정욱진 배포: 프라이봇(www.Fribot.com)

목 차

서문(Preface)

New to robotics?

No problem! The activities and projects in this text start with an introduction to the BOE Shield-Bot's brain, the Arduino® Uno. Then, you will build, test, and calibrate the BOE Shield-Bot. Next, you will learn to program the BOE Shield-Bot for basic maneuvers. After that, you'll be ready to add different kinds of sensors, and write sketches to make the BOE Shield-Bot sense its environment and respond on its own

New to microcontroller programming?

This is a good place to start! The code examples introduce Arduino programming concepts little by little, with each example sketch explained fully.

New to electronics?

See how each electronic component is used with a circuit symbol and part drawing. Traditional schematics next to wiring diagrams make it easy to build the circuits.

Robots are used in the auto, medical, and manufacturing industries, in all manner of exploration vehicles, and, of course, in many science fiction films. The word 'robot' first appeared in a Czechoslovakian satirical play, Rossum's Universal Robots, by Karel Capek in 1920. Robots in this play tended to be human-like. From this point onward, it seemed that many science fiction stories involved these robots trying to fit into society and make sense out of human emotions. This changed when General Motors installed the first robots in its manufacturing plant in 1961. These automated machines presented an entirely different image from the "human form" robots of science fiction.

Building and programming a robot is a combination of mechanics, electronics, and problem-solving. What you're about to learn while doing the activities and projects in this text will be relevant to real-world applications that use robotic control, the only differences being the size and sophistication. The mechanical principles, example program listings, and circuits you will use are similar to very common elements in industrial applications developed by engineers.

The goal of this text is to get you interested in and excited about the

fields of engineering, mechatronics, and software development as you construct, wire, and program an autonomous robot. This series of hands-on activities and projects will introduce you to basic robotic concepts using the BOE Shield-Bot. Its name comes from the Board of Education® Shield for Arduino prototyping board that is mounted on its wheeled chassis. An example of a BOE Shield-Bot with an infrared obstacle detection circuit built on the shield's prototyping area is shown below.

Submitted by Andy Lindsay on Mon, 02/20/2012

About the Author

Andy Lindsay joined Parallax Inc. in 1999, and has since authored a dozen books, including What's a Microcontroller? as well as numerous articles and product documents for the company. The original Robotics with the Boe-Bot that is the inspiration for this book was designed and updated based on observations and educator feedback that Andy collected while traveling the nation and abroad teaching Parallax Educator Courses and events. Andy studied Electrical and Electronic Engineering at California State University, Sacramento, and is a contributing author to several papers that address the topic of microcontrollers in pre-engineering curricula. When he's not writing educational material, Andy does product and application engineering for Parallax.

옮긴이들

"아두이노 로봇 가지고 놀기" 교재는 미국 Parallax.com 사의 온라인 배포자료(http://learn.parallax.com/ShieldRobot)를 한국어로 번역한 것이다. 그리고 Parallax 사의 부품을 사용하면 여러분들이 본 교재와 동일한 실습을 수행할 수 있도록 되어 있다.

아두이노 로봇(ABOT)은 아두이노를 사용한 로봇 자동차라는 뜻으로 프라이봇이 붙인 새로운 이름이며, Ardunio Uno, Duemilanove 와 Mega 중 하나의 마이크로컨트롤러 그리고 Parallax사의 Robotics Shield Kit를 준비하여 조립 제작할 수 있다.

ABOT은 중/고등 교보재 또는 대학 교보재 등으로 사용될 수 있는 로봇이며, 로봇 초보자이거나 마이크로컨트롤러 프로그래밍 초보자이거나 전자공학 초보자이더라도 본 교재를 사용하여 일련의 실습 과정들을 충분히 따라할 수 있다.

이 교재는 ABOT을 위한 아두이노 제어, ABOT을 조립하고 주행하기, 다양한 센서들을 이용하여 ABOT 제어하기 등을 여러분들로 하여금 직접 다루게 할 것이다. 그리고 이 교재의 목표는 여러분들이 자동차 조립, 전자회로 배선, 프로그래밍에 대한 엔지니어링, 메카트로닉스, 소프트웨어 기술개발에 흥미를 갖도록 하는 것이다.

프라이봇(www.fribot.com)은 역자들과 함께 향후에도 지속적인 정보와 자료를 여러분들에게 제공할 계획이다. 짧은 시간동안 작업한 내용이라 조금은 거칠고 부족한 점이 있지만, 여러분들의 지도와 편달을 기대하며 이 교재를 공개한다. 마지막으로 번역을 위해 시간을 할애해 주신 아래 역자님들께도 여러분들과 함께 심심한 감사의 뜻을 전한다.

번역자 심재창 : jcshim@andong.ac.kr
경북대학교 전자공학과 공학박사
현)국립 안동대학교 컴퓨터공학과 교수
http://cafe.naver.com/arduinocafe

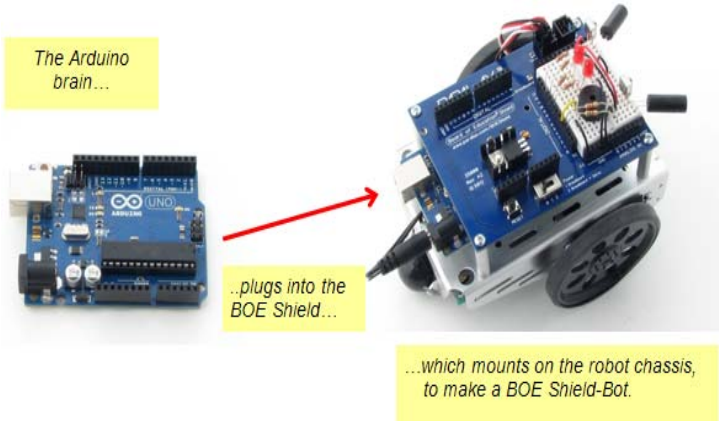
번역자 고주영 : sonice@andong.ac.kr
안동대학교 멀티미디어공학과 공학박사
현)안동대학교 교육개발원 강사

번역자 이영학 : annaturu@ikw.ac.kr
영남대학교 전자공학과 공학박사
현)경운대학교 항공대학 항공전자공학과 교수

번역자 정욱진 : mail@fribot.com
경북대학교 전자공학과 공학박사
현)위덕대학교 전자공학과 강사

제1장 ABOT의 두뇌

ABOT(Parallax사의 "BOE Shield-Bot robot" 명칭을 프라이봇이 "아두이노 로봇"이라는 뜻으로 변경한 새이름입니다.)은 로봇의 실습과 프로젝트를 위해 만들어졌다. ABOT 보드에 알루미늄 차체, 서버 모터 및 바퀴를 연결하면 로봇이 완성된다. ABOT 보드에는 프로그램 가능한 ATmega328이라는 두뇌가 포함되어 있다.



이 교재의 실습은 기계 부품의 조립 방법과 회로 연결 방법을 안내한다. 그리고 ABOT이 작동하도록 프로그램을 작성한다.

1. 주변의 상황을 모니터 센서가 감지한다.
2. 센싱된 것을 기반으로 판단을 내린다.
3. 모터를 제어한다(모터를 작동시켜 바퀴를 회전한다).
4. 로봇에 관심 있는 분들과 정보를 교환한다(여러분들의 몫).

하드웨어와 소프트웨어

실습 1: 아두이노 소프트웨어 다운로드와 압축풀기
실습 2: 간단한 "Hi !" 스케치 작성
실습 3: 값을 저장하고 가져오기
실습 4: 수학 문제 풀기
실습 5: 판단하기
실습 6: 수를 세면서 제어 반복하기
실습 7: 상수와 주석
제1장 요약

하드웨어와 소프트웨어

이 책의 실습을 위해서는 다음의 하드웨어가 필요하다.:

1. 아두이노 보드와 프로그래밍 USB 케이블
2. Robotics Shield Kit (Parallax #130-35000, Board of Education Shield included)
3. **최신의 아두이노 소프트웨어 다운로드**

만약 아두이노를 처음 사용해 본다면, 실습 1에서 소프트웨어를 설치한 후, 하드웨어를 연결하고 프로그래밍을 작성하고 업로드 한다. 이 장에는 다양한 스케치라 부르는 예제를 포함하는데, 일반적인 프로그래밍의 개념을 소개한다. 이 스케치는 로봇을 위한 가장 기초적이며 중요한 코드이고 다음과 같다.

- "Hi !" 출력
- 값을 저장하고 읽기
- 수학 문제 풀기
- 판단하기
- 숫자를 세고 제어를 반복하기

이장의 예제들은 외부회로와 소통을 크게 필요로 하지 않는다. 뒷 장에서 회로를 구성하고, 로봇을 이동한다. 그리고 데이터를 관리하는 방법과 재사용할 수 있는 코드의 프로그래밍에 대해서도 학습한다.

실습1: 소프트웨어 다운과 설치

이번 작업이 아두이노 시스템 관련해서 처음 접근하는 것이면 소프트웨어를 다운받고 설치해야 한다.

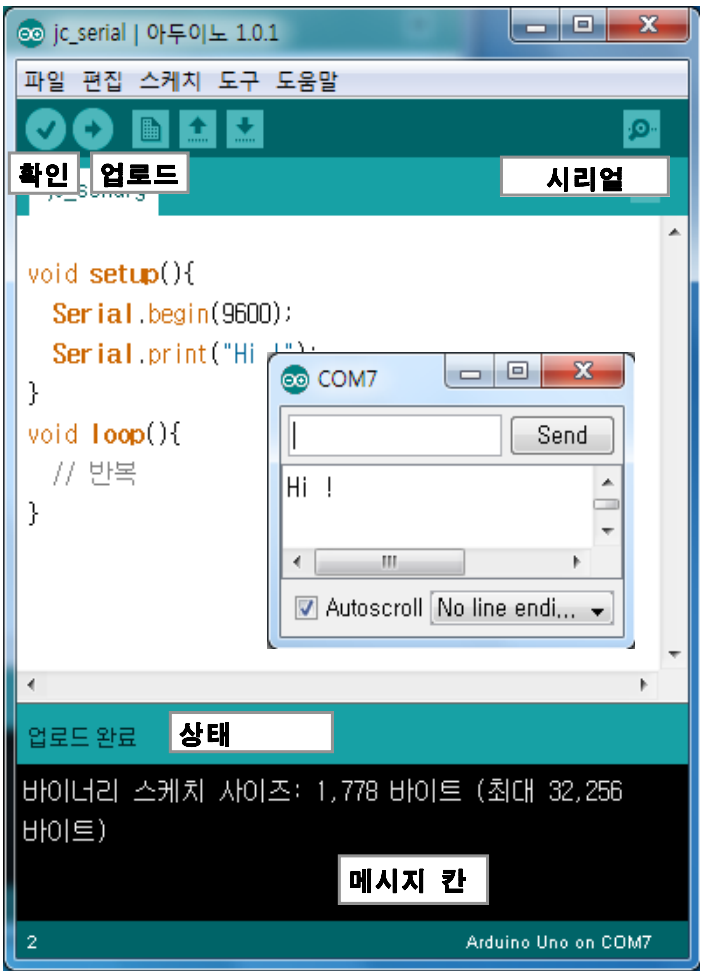
- http://www.arduino.cc 에 접속하고 Getting Started 링크를 클릭하자.
- 설명에 따라 최신의 아두이노 소프트웨어를 다운로드하고 압축을 푼다. 아두이노와 컴퓨터 간의 통신을 위해서 USB 드라이버를 시스템에 설치한다.
- 컴퓨터에서 작성한 스케치를 아두이노 보드에 성공적으로 업로드시켜야 한다.

아두이노 개발 환경에서는 로봇의 기본적인 작업을 위해서 작성하는 프로그램을 스케치라고 부른다. 아두이노 스케치는 사용자가 쉽게 접근할 수 있는 템플릿을 제공하며 C나 C++프로그래밍 언어로 작성된다.
만약 예제 코드를 직접 입력하면 프로그래밍 기술이 훨씬 빨리 향상된다. 그러나 테스트용 스케치나 회로 문제 또는 버그 해결과 관련한 도움말이 있으면 편리하다. 그래서 모든 스케치를 제공하며 다음 링크에서 다운로드 받을 수 있다.

- Download Chapter 1 Arduino code
- 컴퓨터에 저장하고 압축을 풀고 사용한다.

실습2: 간단한 "Hi !" 스케치

"Hi !" 메시지를 시리얼 모니터 윈도우에 출력하는 아두이노 개발 환경 창을 캡처한 그림은 다음과 같다.



- 아두이노 소프트웨어를 열고 다음 코드를 입력하자:

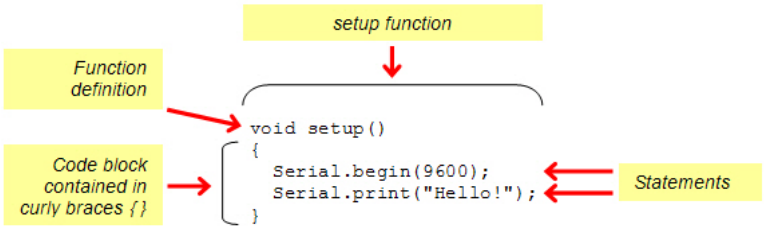
```
void setup(){
  Serial.begin(9600);
  Serial.print("Hi !");
}
void loop()
```

```
{  // 여기에 코드를 적으면 자동으로 반복된다.
}
```

- 시리얼(Serial)의 첫 문자인 s가 대문자이므로 주의를 하자. 소문자로 적으면 작동하지 않는다.
- 스케치에서 괄호()와 중괄호{}를 자주 사용하는데, 용도에 맞게 바르게 사용해야 한다.
- 툴바에서 체크 표시된 [확인] 버튼을 눌러서 코드 오류를 확인한다.
- 메시지 칸에 "바이너리 스케치 사이즈"가 보이는가?
- 그렇다면 코드가 완성되었고 아두이노에 업로드할 준비가 된 것이다.
- 오류가 보이면, 코드 컴파일이 되지 않은 것이므로 오류를 수정하자.
- 툴바의 오른쪽 화살표가 포함된 [업로드] 버튼을 누르면 코드 아래에 있는 상태 줄에 "Compiling sketch…," "Uploading…," 그러고 나서 "Done uploading."이 출력된다.
- 업로딩이 완료된 다음 툴바의 오른쪽 끝에 있는 [시리얼 모니터] 버튼을 클릭하자.
- 시리얼 모니터가 열리자마자 "Hi !" 메시지가 나오지 않으면, 모니터의 오른쪽 아래에 있는 "9600 baud" 부분을 확인하자.
- [파일] → [저장]을 선택하여 스케치를 HiMessage로 저장하자.

Hi 스케치 코드의 작동 원리

함수는 코드의 각 줄에 적힌 문장들을 담는데 ABOT에게 어떤 일을 하도록 명령을 한다. 아두이노 언어는 setup()과 loop()라는 두 개의 미리 만들어진 함수가 있다. setup() 함수는 다음과 같다.



아두이노는 setup() 함수의 중괄호 { } 사이에 있는 문장들을 프로그램이 시작될 때 단 한번만 실행한다. 이 예제에서, 두 개의 문장인 Serial.begin(speed)와 Serialprint(val)는 모두 함수를 호출하는 것인데 아두이노에서 미리 코드로 만들어진 Seiral()과 관련된 것이다. 여기서 speed와 val를 함수의 매개변수(*parameters*)라 한다. 함수에서 명령을 처리하기 위해서 함수를 호출할 때 전달할 필요가 있는 값들이다.

Serial.begin(9600); 은 ***speed*** 변수에 9600을 전달한다. 이것은 데이터를 초당 9600 비트로 시리얼 모니터와 주고받을 준비가 되었다는 것을 말해 준다.
이것은 비트 1 또는 비트 0을 1초에 9600개를 처리한다는 의미이며, 이것을 일반적으로 보드레이트(*baud rate*)라 한다.

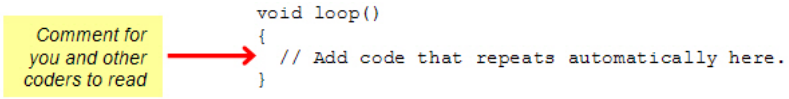
Serial.print("Hi !"); 메시지 "Hi !"를 매개 변수로 전달한다. 모니터는 시리얼 비트 스트림 "Hi !" 메시지를 해석해서 출력한다.

setup() 함수를 마치면, 아두이노는 자동으로 빠져나와 loop() 함수로 이동하여 중괄호 속에서 지시하는 문장을 실행한다. loop() 함수 속에 있는 문장들은 독립적으로 계속해서 반복된다.

이 스케치에서는 단지 "Hi !" 메시지를 한 번 프린트 하고, loop() 함수 속에는 실제 명령이 하나도 없는 예이다.

이 코드에는 단지 다른 프로그래머들이 읽을 수 있도록 주석이라고 부르는 것만 들어 있다. 한 줄에서 연속된 슬래시 //의 오른쪽에 있는 내용은 단지 프로그래

머들이 읽기 위한 것으로 아두이노 소프트웨어가 컴파일하지는 않는다. 컴파일러(*compiler*)는 스케치 코드를 가지고 마이크로 컨트롤러가 이해하는 언어인 숫자로 바꾼다.



void가 무엇일까? 왜 함수 끝에는 ()가 오는가? 함수의 첫 줄은 함수에 대한 정의이다. 3개의 부분으로 나누어진다. 리턴형, 함수이름, 파라미터 목록이다. 예로 void setup() 함수는 리턴형이 void이고 함수 이름은 setup이고 () 속에 아무 것도 없으므로 파라미터 목록은 비어 있다.

void의 의미는 "아무것도 없다"는 뜻이다. 다른 함수가 setup() 또는 loop() 함수를 호출해도 리턴 되는 값이 없다. 비어있는 파라미터의 의미는 함수에서 작업을 할 때 받아야 할 값이 필요 없다는 뜻이다.

반복하도록 스케치 바꾸기

마이크로컨트롤러 프로그램은 일반적으로 계속하여 반복 실행된다. 하나 또는 그 이상의 문장들이 반복해서 계속 실행된다는 의미이다. loop() 함수는 중괄호로 된 블록에 있는 코드를 자동적으로 반복한다는 것을 기억하자.
Serial.print("Hi !");를 loop() 함수로 옮기자. 메시지가 반복되는 속도를 늦추기 위해서 delay(*ms*) 함수를 이용하자.

- HiMessage를 HiRepeated으로 저장.
- setup()함수에 있는 Serial.print("Hi !");를 loop()함수로 이동하자.
- 다음 줄에 delay(1000);을 추가하자.
- 여러분들이 변경한 것과 아래 그림에 있는 코드가 같은지 비교하고 맞는지 살펴보자.
- 스케치를 ABOT에 업로드하고 시리얼 모니터를 다시 열자.

한 줄 추가한The delay(1000)은 1000을 ms 파라미터로 delay()함수에 전달한다. 1000 밀리 초 지연을 하기 위함이다. 1ms는 1/1000초 이다. 그래서 delay(1000)은 다음 코드 라인으로 가기 전에 1000/1000=1 초 지연을 한다.

새 줄에 Hi ! 메시지

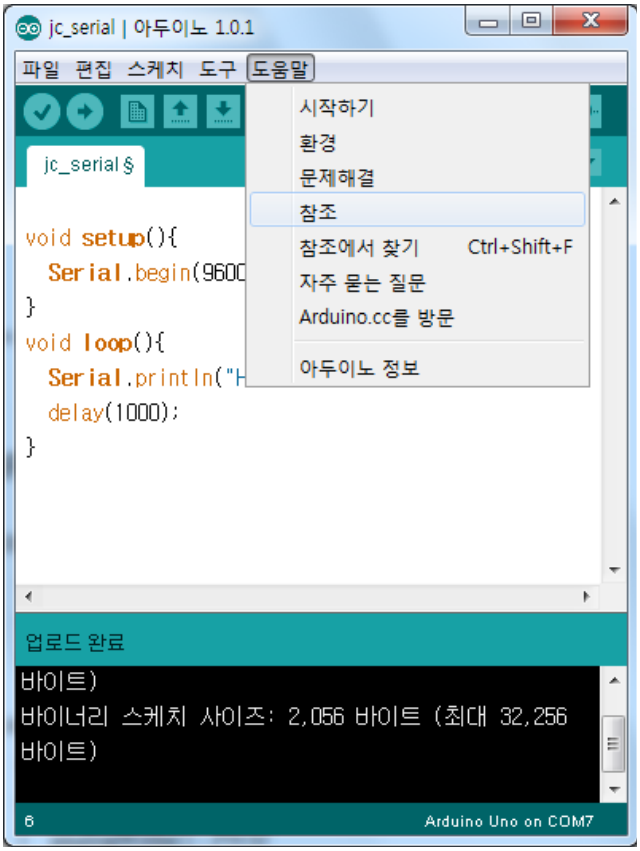
"Hi !"메시지를 새 줄에 어떻게 출력할까? 시리얼 모니터에서 스크롤하여 아래로 내려가도록 해야 한다. print()함수를 println()함수로 바꾸기만 하면 된다.

- Serial.print("Hi!")를 Serial.println("Hi!")로 바꾸자.
- 스케치를 수정하고 업로드하여 Hi ! 메지지 마다 새줄 넣기가 되는지 관찰하자.

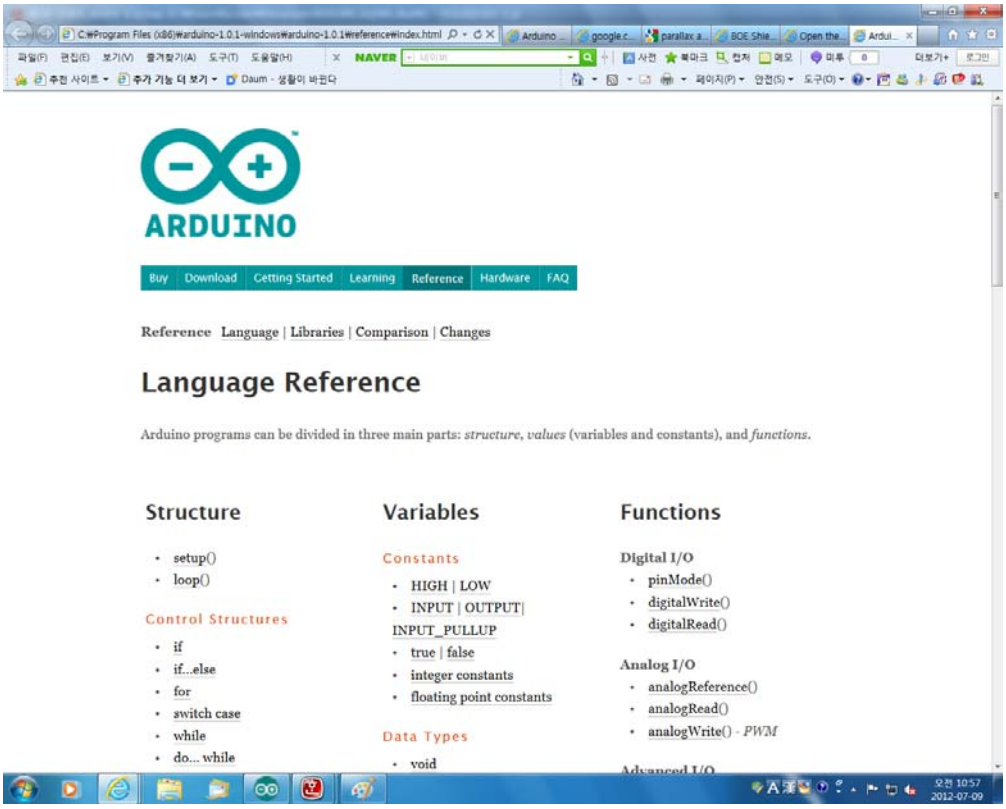
아두이노 참고자료 열기

아직 질문이 있는가? 아두이노 언어 참고(Reference)를 살펴보자. 이 페이지는 setup, loop, print, println, delay, 및 스케치에서 사용할 수 있는 다른 함수들에 대해서 자세하게 설명하고 있다.

- [도움말]에서 [참조] 클릭



- 참조를 선택하면 웹에 다음처럼 나타난다.



- 이 사이트에서 궁금하여 알고 싶은 것이 있는지 잘 살펴보자. setup, loop 및 delay 등을 메인 참조 페이지에서 볼 수 있다.
- 만약 'print'나 'println' 등을 참조하려면 먼저 Serilal 링크를 열면 나온다.

실습3: 값을 저장하고 불러오기

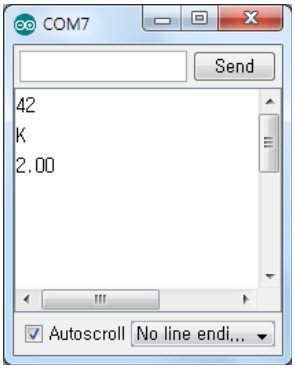
*변수(Variables)*는 값을 아두이노 마이컴의 메모리에 저장하고 가져오려고 생성하는 이름이다. 여기서는 변수 선언(*variable declarations*)된 세 개의 예제를 다음 스케치부터 살펴볼 수 있다.

```
int a = 42;   char c = 'K';   float root4 = sqrt(4.0);
```

int a = 42; 는 이름이 a인 변수를 생성하도록 선언한 것이다. int 가 적힌 부분은 아두이노 소프트웨어가 어떤 형태의 변수를 사용할지 말해준다. 예로 int 형은 -32,768에서 32,767까지 범위의 정수 값을 저장할 수 있다.

이 선언은 역시 42라는 값을 a에 초기 값으로 대입한다.(선언할 때는 int a; 로 해 두고 나중에 a = 42로 값을 넣어도 되지만 int a = 42;로 선언하면 최적화가 된 것이다) 다음은 char c = 'K'; 는 char형(문자를 저장한다) 의 이름이 c인 변수 선언하고 'K'를 대입한 것이다. 다음의 float root4 = sqrt(4.0); 은 float형으로 변수이름을 root4로 선언을 하였다. float형은 소수가 있는 값을 저장할 수 있다. 여기서 root4는 초기 값으로 소수점이 있는 값을 4의 루트 제곱 sqrt(4.0)을 저장한다. 이제 코드는 메모리에 값으로 저장되었으나, 이것을 어떻게 다시 꺼내서 사용할 수 있을까?

간단한 한 가지 방법은 함수의 매개변수로 전달하여 사용하는 것이다. 여기 3가지 사용 예제가 있다. Serial.println(*val*) 함수는 괄호속의 변수를 출력해 주는 함수이다. 괜찮은 것은 Serial.println()함수는 자동으로 데이터 형을 알아 처리를 해주므로 신경을 쓰지 않아도 시리얼 모니터에 값을 출력한다.



예제 스케치 – StoreRetrieveLocal

- [파일]→[새파일] 새로운 스케치를 생성하고 StoreRetrieveLocal 로 저장.
- 아두이노 에디터에서 아래의 코드를 입력.
- 파일로 저장, ABOT에 업로드.
- 시리얼 모니터를 열고 값이 바르게 출력되었는지 확인.

// ABOT – StoreRetrieveLocal

```
setup(){
  Serial.begin(9600);
  int a = 42;
  char c = 'K';
  float root4 = sqrt(4.0);
  Serial.println(a);
  Serial.println(c);
  Serial.println(root4);
}
void loop(){
}
```

ASCII는 *American Standard Code for Information Exchange*의 약어이다.

이것은 컴퓨터의 키나 화면에 문자를 나타내기 위한 일반적인 코드 시스템이다. 예를 들면 아두이노나 시리얼 모니터 모두에서 ASCII 코드 75 또는 'K' 문자는 동일하다. char c = 'K'; 선언은 아두이노가 숫자 75를 c 변수에 저장한다. Serial.println(c)는 아두이노가 숫자 75를 시리얼 모니터로 보낸다. 시리얼 모니터가 75를 받으면 자동 적으로 K를 화면에 출력한다.

- ASCII 코드 0-127 보기.

전역(Global)과 지역(Local) 변수

지금까지는 변수 선언을 중괄호 속인 함수 블록 안에서 지역 변수로 사용하였다. 지역 변수는 오직 해당 함수 속에서만 사용이 가능하고 수정할 수 있다. 그리고 지역 변수는 함수가 선언되어 사용되고 있을 때만 존재한다. 함수 블록을 떠나면, 사용하던 메모리를 해제하여 다른 함수가 지역 변수를 사용할 수 있다.
만약 스케치에서 둘 이상의 함수에서 변수를 사용해야 하면 전역 변수를 사용해야 한다. 전역변수를 만들기 위해서는 setup()함수가 시작하기 전인 함수 외부에서 선언하면 된다. 그러면, 스케치의 모든 함수가 값을 수정하거나 활용할 수 있다. 다음 스케치 예제는 전역 변수를 선언하고 함수에서 값을 할당하는 것이다.

예제 스케치 - StoreRetrieveGlobal

이 예제 스케치는 a, c, 및 root4를 전역 변수로 선언한 것이다. 지금부터 이 변수들은 전역 변수이므로 프로그램 전체에서 활용이 가능하다. 물론 setup()함수와 loop()함수에서 접근할 수 있다.

- 스케치를 아래처럼 수정하자.
- 파일을 StoreRetrieveGlobal로 저장한 후 ABOT에 업로드.
- 시리얼 모니터를 열고 loop() 함수에서 반복적으로 출력하는 값이 옳은지 확인

// ABOT - StoreRetrieveGlobalint

```
int a; char c; float root4;
void setup(){
  Serial.begin(9600);
  a = 42;
  c = 'K';
  root4 = sqrt(4.0);
}
void loop(){
  Serial.println(a);
  Serial.println(c);
  Serial.println(root4);
  delay(1000);
}
```

여러분의 차례 – 다양한 변수형

단지 int, char, float, 및 byte 이외에도 많은 데이터 형이 있다.

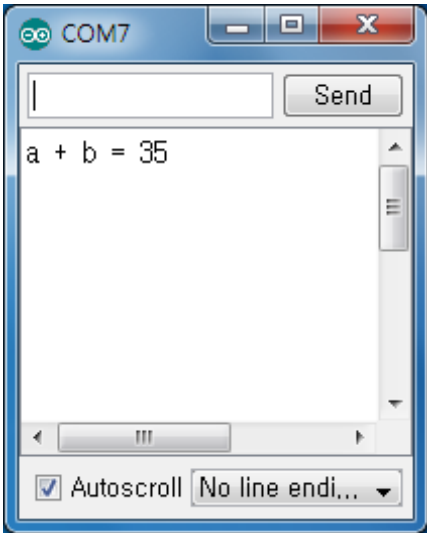
- 아두이노 언어 참고를 열고 데이터 형의 리스트를 확인해 보자.
- float 링크를 따라가서 이와 같은 데이터 형에 대해서 학습해 보자.
- 이장의 마지막 부분에서 사용하는 long 형과 int 형에 대해서도 살펴보자. 어떤 점이 닮았고, 어떤 점이 다른가?

실습4: 수학 문제 풀기

산술연산자는 스케치에서 계산하기에 유용하다. 이 실습에서는 기초적인 연산자에 초점을 맞춘다. 대입 assignment (=), 덧셈 addition (+), 뺄셈 subtraction (-), 곱셈 multiplication (*), 나눗셈 division(/), 나머지 modulus (%, 나누고 남는 값).

- 아두이노 언어 참고(Arduino Language Reference)을 열고 산술 연산자에 대해서 살펴보자

다음 예제 스케치는 SimpleMath로 두 변수 a와 b를 함께 더하고 결과를 c에 저장한다. 결과를 시리얼 모니터에 출력한다. 이번에 c는 문자로 선언되지 않고 정수로 선언되어 있다. int c = a + b;에서 대입연산자(=)는 a와 b를 더한 결과값을 c에 복사한다. 아래 그림처럼 시리얼 모니터에는 12 + 23 = 35가 표시된다.



- 아두이노에서 SimpleMath를 입력하고 저장한 다음 ABOT에 업로드하자.
- 결과가 맞는지 시리얼 모니터를 통해서 확인해보자.

// ABOT - SimpleMath

```
void setup(){
```

```
  Serial.begin(9600);
  int a = 12;
  int b = 23;
  int c = a + b;
  Serial.print("a + b = ");
  Serial.println(c);
}
void loop(){  // 비어있음, 반복되는 코드가 없음
}
```

변수는 저장하는 값의 크기에 맞추는 것이 좋다. 적합한 메모리를 활용하면 더 큰 스케치를 실행할 때 효과적으로 실행할 수 있다. 소수점이 있는 경우는 float를 사용할 수 있다. 수를 세기 위해서 만약 정수를 사용한다면 byte, int 또는 long을 선택하자. byte를 사용하면 0에서 255까지 부호 없는 숫자가 된다. 만약 결과가 -32,768에서 32,767의 사이라면 int를 활용할 수 있다.

만약 큰 범위의 숫자가 필요하다면, long 변수를 사용할 수 있다. 이것은 -2,147,483,648에서 2,147,483,647까지 가능하다.

여러분의 차례 - 다른 산술 연산자의 실습

아직 사용해 보지 않은 -, *, /, 및 % 시도해 보자.

- 덧셈 (+) 연산자를 뺄셈 (-) 연산자로 바꾸어 보자 그리고 ABOT에 다시 업로드 하자.
- 시리얼 모니터를 활용하고 검사를 하자.
- 곱셈 (*), 나눗셈 (/) 및 모듈러 (%) 연산자.

수학에서 소수점이 있는 경우

원을 그려 두고 ABOT이 원을 따라 간다고 상상을 해보고, 원의 주변길이를 계산할 수 있다. 원주는 2 × π × r이며 r은 원의 지름이고 π ≈ 3.14159 이다. 이 계산은 소수점이 있는 계산이 더 쉽다. 여기 코드가 있는데 3.14159 숫자 대신 PI를 활용하고 있다. PI는 C 언어에서 스케치 전체에서 변경할 수 없는 상수(*constant*)로 정의 되어 있다. 소수점이 있는 경우는 부동소수점으로 값을 사용한다.

```
float r = 0.75;
float c = 2.0 * PI * r;
```

예제 스케치 – Circumference

- 아두이노 에디터에 Circumference 스케치를 입력하고 저장.
- 값으로 0.75와 2.0 활용한지 확인하자. 2.0 대신 2를 사용하면 안 됨.
- 스케치를 ABOT에 업로드하고 시리얼 모니터로 결과를 확인.

```
// ABOT – Circumference
setup(){
  Serial.begin(9600);
  float r = 0.75;
  float c = 2.0 * PI * r;
  Serial.print("원의 둘레 길이: 원주 = ");
  Serial.println(c);
}
void loop(){  // 비어있음, 반복되는 코드가 없음
}
```

여러분의 차례 - 원의 면적

원의 면적은 a = π × r^2 이다.

힌트: r^2 은 r × r 로 계산 된다.

- CircumferenceArea로 스케치를 저장.
- 면적이 계산된 결과를 저장하기 위한 float 변수 하나를 더 추가하고, 이 값을 출력하도록 코딩하자. 스케치를 실행하고, 계산기에 의해서 계산한 것과 비교를 해 보자.

실습5: 판단하기

ABOT은 입력 센서에 따라 이동할 때 여러 가지 결정이 필요하다. 판단하는 간단한 스케치 데모를 보자. a와 b 두 수를 비교해서 a가 b보다 큰지 아닌지를 if…else 문장을 사용한다. if 조건에서 (a>b)가 참이면, if 코드 블록의 문장을 실행한다. Serial.print("a가 b보다 크다"); 만약 a가 b보다 크지 않다면 else 코드블럭을 대신 실행한다. Serial.print("a 는 b보다 크지 않다");

- 아두이노 에디터에 코드를 입력하고, 저장한 다음 ABOT에 업로드한다.
- 시리얼 모니터를 열고 바르게 메시지가 나오는지 확인한다.
- a와 b의 값을 서로 바꾸어 보자.
- 스케치를 다시 업로드해서 살펴보자.

// ABOT - SimpleDecisionsvoid

```
setup(){
  Serial.begin(9600);
  int a = 5;
  int b = 4;
  if(a > b)  {
    Serial.print("a 가 b 보다 크다");
  }
  else  {
    Serial.print("a 는 b 보다 크지 않다");
  }
}
void loop(){  // 비어있음, 반복되는 코드가 없음
}
```

if... else if 로 추가 판단하기

아마 “a가 b보다 크면”의 메시지가 필요한 경우, else 부분을 생략할 수 있다. setup() 함수 내에 if() 조건문을 달랑 하나만 두어도 된다.

```
void setup(){
  Serial.begin(9600);
  int a = 89;
  int b = 42;
  if(a > b)  {
    Serial.print("a is greater than b");
  }
}
```

만약 스케치에서 세 가지 조건이 필요하다면, 큰가, 적은가 아니면 같은가? 이 경우 if…else if…else 문을 활용할 수 있다.

```
if(a > b)
  {
    Serial.print("a is greater than b");
  }
  else if(a < b)
  {
    Serial.print("b is greater than a");
  }
  else
  {
    Serial.print("a is equal to b");
  }
```

스케치에서는 관계 연산자 &&나 || 등을 활용하여 많은 다중 조건을 활용할 수 있다. && 연산자는 AND를 의미하고 ||는 OR 연산자를 의미한다. 예로서 a 가 10보다 크고 b가 10보다 작다면

```
if((a > 10) && (b < 10))  {
    Serial.print("적당한 범위 속에 있습니다.");
  }
```

다른 예로 a가 100보다 크거나 b가 0보다 적으면 경고 신호를 보낸다.

```
if((a > 100) || (b < 0)) {
    Serial.print("위험 상황 !");
  }
```

마지막의 예로 두 값이 같은 경우를 찾고 싶다면 =을 두 개 해서 == 활용한다.

```
if(a == b) {
    Serial.print("a와 b는 같다.");
  }
```

- 이러한 변형을 스케치에 적용해 보자.

!	하나 이상의 else if 문을 if문 다음에 사용하면 참을 만날 때까지 계속하여 실행된다. if문에서 코드 블록이 참으로 판단되면 남은 else 블록은 실행하지 않는다.

실습6: 계산과 제어의 반복

로보틱 작업에서 동작이 여러 차례 반복되는 일들이 자주 발생된다. 반복을 위해서 for 반복과 while 반복을 선택할 수 있다. 정해진 수만큼 반복할 때는 대체로 for 반복 블록을 많이 사용한다. while 반복은 조건이 참이 될 때까지만 코드 블록을 반복한다.

수를 세기 위한 for 루프

어떤 횟수만큼 블록내의 문장을 반복하려면 for 반복문을 사용한다. 예로서 ABOT이 거리를 감지하기 위해서 5개의 서로 다른 값을 사용할 경우, 어떤 코드 블록을 5회 반복해야 한다. 이 작업에서 우리는 for 반복문을 사용한다. 여기 예제로 1부터 5까지를 반복하여 시리얼 모니터에 값을 출력해 보자.

- CountToFive에 입력하고 저장한 후 업로드 하자.
- 시리얼 모니터를 열고, 1에서 5까지 세는지 확인하자.

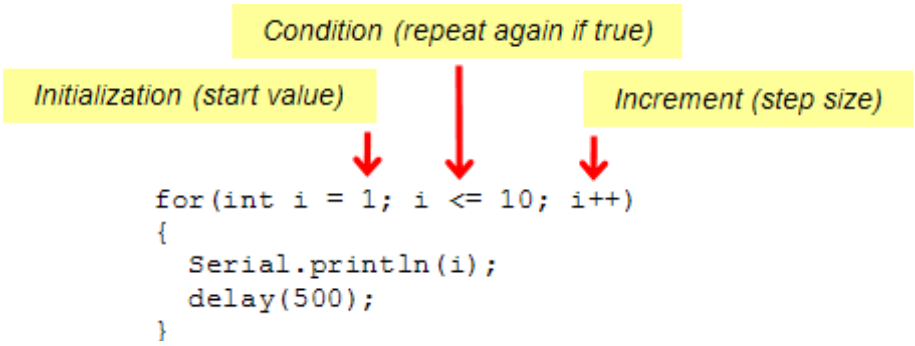
// ABOT - CountToFive

```
void setup(){
  Serial.begin(9600);
  for(int i=1;i<=5;i++){
    Serial.println(i);
```

```
  }
}
void loop(){  // 비어있음, 반복되는 코드가 없음
}
```

반복의 작동 원리

예제 CountFiveTimes 스케치의 for 반복에 대한 그림은 다음과 같다. for 반복문의 괄호 () 에는 3개의 요소가 있는데, 어떻게 제어가 되는지 살펴보자.



- 초기화: 처음 시작하는 숫자 값. 괄호 속에 int i=1; 로 지역 변수로 선언 된다. i를 인덱스(index)라 한다.
- 조건: 매번 반복될 때 마다 조건을 검사하여 참이면 반복을 계속 한다. 참이 아니면 for 반복 블록 다음 문장을 수행한다. 이 경우 만약 i가 5보다 작거나 같을 때까지 반복한다.
- 증분: 다음 반복이 될 때 i 값을 얼마만큼 변경시킬지 정한다. i++문장은 i = i + 1과 동일하다. i++를 축약된 형태라고 하고, 후 증가 연산자라 한다. i를 사용한 이후에 증가시킨다.

반복이 시작되는 처음에는 i 값이 1이다. 그래서 Serial.println(i) 는 시리얼 모니터에 1을 출력한다. 다음 번에는 i가 1 증가한다. delay(500)은 0.5초간 기다린 후에 for 문은 조건 i<=5 인지 검사를 한 후 i가 2가 되므로 참이 되고 5보다 작으므로 코드 블록을 반복한다. 이 과정이 계속 반복되고 i가 6이 되면 6<=5 코드 블록을 더 이상 실행하지 않는다.

초기 값 설정, 조건, 증가

이전에 언급한 것처럼 매번 마다 i++증가 연산자를 사용하여 i변수에 1을 더한다. 복합 연산자 --는 감소에 사용된다. 복합 산술 연산자 +=, -=, *= , /=도 사용이 가능하다. += 연산자의 활용 예로서 i = i + 1000은 i+=1000과 동일하다.

- 스케치를 CountHigherInSteps에 저장하다.
- 스케치의 문장을 다음으로 바꾼다.

  for(int i = 5000; i <= 15000; i+=1000)
- 업로드 하고 시리얼 모니터를 통해서 관찰해 보자.

루프는 조건이 참이 될 때까지 반복

앞으로 뒷장에서 센서가 어떤 값을 리턴하는 동안 while() 반복문이 계속 돌아가는 스케치를 만든다. 현재는 센서가 연결되어 있지 않아서 while() 반복으로 5까지만 세어 보자.

```
int i = 0;
while(i < 5)  {
  i = i + 1;
  Serial.println(i);
  delay(500);
}
```

코드를 조금 짧게 하려면 증가연산자 (++)를 활용할 수 있다. ++를 i의 오른쪽에 두게 되면 Serial.println(i++); 함수에서는 변화가 없다. 따라서 처음에는 0이 시리얼 모니터에 찍힌다. i++의 경우 다음 줄부터 변화된 값이 적용된다. 만약 1부터 찍고 싶다면 ++i 로 바꾸어야 한다.

```
int i = 0;
while(i < 10)  {
  Serial.println(++i);
  delay(500);
}
```

loop() 함수는 계속 반복된다. 다른 방법으로 문장을 계속 반복시키는 스케치는

다음과 같다.

```
int i = 0;
while(true)  {
  Serial.println(++i);
  delay(500);
}
```

이것이 어떻게 작동 될까? while() 반복은 괄호속이 참으로 판단되면 계속 반복한다. 'true'는 미리 정의된 상수이다. true의 값이 1이므로 while(true)는 항상 참이고, 계속 반복한다. while(false)는 어떻게 실행될까?

- 이 세 가지 서로 다른 while() 반복을 CountToFive 스케치에서 실습해 보자.
- Serial.println(++i)을 사용하였을 때, 시리얼 모니터를 통해서 어떻게 변경되는지 관찰하자.
- while(true)와 while(false)도 실습하자.

실습7: 상수와 주석

다음 스케치는 CountToFiveDocumented이며 CountToFive와는 여러 면에서 다르다. 먼저, 블록 주석이 윗부분에 나온다. 블록 주석은 /* 시작해서 */로 끝난다. 그 사이의 라인은 모두 주석에 포함된다. 라인 주석은 각 줄에 // 의 오른쪽으로 코드에 대한 설명을 적는다. 마지막으로 스케치의 앞부분에 const int (정수 상수) 선언이 있는데, 이름이 startVal, endVal, baudRate로 각각 값이 1, 10, 9600 이다. 그 다음, 스케치는 이 값들이 필요한 곳에 이 정수 상수를 활용한다.

/* ABOT - CountToFiveDocumented 이 스케치는 1부터 5까지를 증가하면서 세고 시리얼 모니터를 통해 출력한다. */

```
/*
 Shield-Bot - CountToFiveDocumented
 This sketch displays an up-count from 1 to 5 in the Serial Monitor
 */

const int startVal = 1;                          // Staring value for counting
const int endVal = 5;                            // Ending value for counting
const int baudRate = 9600;                       // For setting baud rate

void setup()                                      // Built in initialization block
{
  Serial.begin(baudRate);                        // Set data rate to baudRate

    for(int i = startVal; i <= endVal; i++)      // Count from startVal to endVal
  {
    Serial.println(i);                          // Display i in Serial Monitor
    delay(500);                                  // Pause 0.5 s between values
  }
  Serial.println("All done!");                  // Display message when done
}

void loop()                                       // Main loop auto-repeats
{
```

```
  // 비어있음, 반복되는 코드가 없음
}
```

코드의 문서화

문서화:Documenting 코드는 프로그램의 한 부분에 노트를 적는 과정이다. 변수명이나 상수 이름을 잘 작명하면 자동으로 문서화 코드가 만들어져 도움이 된다. 만약 프로그래밍 관련하는 분야로 일을 한다고 생각하면 문서화를 잘 하는 좋은 습관을 지금부터 가질 필요가 있다.

- 생계를 위해서 코드를 작성하는 소프트웨어 개발자나 로봇 프로그래머는 코드를 문서화하는 지시를 받는다.
- 다른 사람들이 여러분의 코드를 업데이트하거나 다른 과제에서 이용하거나 할 경우 코드가 어떤 작업을 하는지 이해할 수 있도록 해야 한다.
- 코드에 대한 설명은 시간이 많이 지난 다음에 이 코드가 어떤 작업을 하는 것인지 기억나게 하는데 도움을 주어 개발자의 시간을 많이 줄여준다.

추가로 여러분의 코드를 읽기 쉽게 하기 위해서 상수는 자주 사용하는 값을 빠르고 정확하게 조절하는데 활용할 수 있다. 각 경우에 이름이 붙여지지 않은 변수를 손으로 일일이 값을 정해 주다 보면 스케치에 버그를 만들기 쉽다.

- 스케치를 보면서 상수와 주석이 어떻게 사용된 지 살펴보자.
- CountToFiveDocumented를 예제로 해서 SimpleDecisions 스케치를 열어보고 문서화 해보자.
- 스케치에 대한 자세한 설명을 추가하고 제목을 포함하여 주석 블록을 만들자.
- 숫자값 89와 42 대신 상수를 선언하자.
- setup() 함수내에서 89나 43와 같은 숫자 대신에 상수 선언을 하고 이름을 붙여 주자.
- 각 줄의 오른쪽에 라인 주석을 달자.

제1장 요약

아두이노 사이트에서 설치하고 소프트웨어를 테스트하고 프로그램을 연결하고, 몇 가지 실습을 하였다. 이 예제 스케치를 통해서 마이컴이 어떻게 작동되는지 살펴보았고, 프로그래밍에 대한 개념과 여러분들이 컴퓨터 다루는 좋은 기술과 습관에 대해서 몇 가지 추천을 하였다.

마이컴이 하는 일

- 출력을 위해서 메시지를 시리얼 모니터로 보낸다.
- 메모리에 값을 저장하고 읽는다.
- 수학 문제의 답을 계산한다.
- 제어 프로그램의 흐름에 따라 선택을 결정한다.
- 수를 세는 것과 제어를 반복한다.

프로그래밍 개념

- 함수란 무엇이며, 함수 매개변수가 어떻게 값을 전달하는가.
- 아두이노의 setup()과 loop() 함수가 어떤 일을 하는가.
- 전역(global)변수와 지역(local) 변수의 차이점.
- char, int 및 float 데이터 형을 선언하고 사용하기.
- 수식을 이용하여 어떻게 수학 문제를 푸는가.
- if, if…else, and if…else if를 이용해서 어떻게 결정을 하는가.
- if문장에서 비교 연산자와 관계 연산자를 사용하는가.
- for와 while 루프에서 어떻게 수를 세고 제어를 반복하는가.

컴퓨터 스킬(Skills)

- 보드레이트란, 시리얼 모니터로 스케치에서 어떻게 보내는가.
- ASCII 문자가 무엇이며 어떻게 사용되는가.
- 마이크로컨트롤러 언어 참고를 활용하기.
- 코드의 문서화가 중요하며, 코드에 스스로 문서화를 어떻게 하는지.

제1장 도전

질문(Questions)

1. ABOT의 두뇌는 어떤 MCU로 되어 있을까?
2. 아두이노가 문자를 PC로 보낼 때, 어떤 형의 숫자들이 프로그래밍 케이블을 통해서 보내 질 때 사용될까?
3. setup() 함수와 loop() 함수의 차이는?
4. 변수명과 변수형 간의 차이점은?
5. 전역변수와 지역변수의 차이는?
6. 연산자는 무엇이며 각각 어떤 일을 하는가?
7. 21.5를 저장하려면 어떤 변수형이 필요할까?
8. for() 반복문의 괄호속에 들어가는 3가지 요소는?
9. 블록코멘트와 라인코멘트의 차이는?

예제(Exercises)

1. “the value of i = ”가 시리얼 모니터에 출력 되도록 하고 이어서 변수 i의 값이 출력되도록 코드를 작성하세요. 모두 같은 줄에 출력되고 커서가 다음 줄의 시작 지점에 이동하여 추가해서 메시지를 출력 할 수 있도록 하세요.
2. bigVal이라는 이름의 long 형 변수를 선언하고 초기 값으로 8천만을 저장하세요.
3. 변수를 2로 나누고 이것을 0과 비교하는 if…else 문장을 작성하라. 만약 결과가 0이면 “변수는 짝수”를 출력하고 그렇지 않으면 “변수는 홀수”를 출력하라.
4. 21부터 39까지를 3씩 증가하면서 세는 for() 반복문을 작성하라.
5. 문자 변수를 ASCII 값으로 저장하고 출력하는 코드를 작성하세요.
6. 숫자를 세는 대신 알파벳 ‘A'부터 ’Z'까지를 출력하는 for() 반복문을 작성하세요.

프로젝트(Projects)

1. 프린트 가능한 ASCII 문자를 출력하도록 스케치를 작성하세요. 출력 가능한 첫 문자는 빈칸 ‘ ’ 으로 키보드의 스페이스 바를 누르면 생성되는 문자로 따옴표로 빈칸을 감싸고 있다. 마지막 인쇄가능한 문자는 틸트 문자 ‘~’ 이다. 반면, 반복의 시작에서 32를 사용하고 끝으로 126

을 사용하면 된다.

2. 변수가 홀수 인지 짝수 인지 말하도록 스케치를 작성하라. 힌트: 숫자가 짝수이면 2로 나눌 때 나머지 값이 0이다. 힌트: 변수%2==0활용.

제1장 해답

질문해답(Question Solutions)

1. 아두이노 모듈 또는 ATmega328.
2. 바이너리 숫자, 0이나 1로 된 바이너리 숫자. ??= 'K'처럼 예제로 ASCII 코드가 숫자로 표현되는지 배웠다.
3. setup()함수의 문장은 프로그램이 시작될대 단 한번만 실행된다. setup()함수가 끝나면 스케치는 loop()함수로 이동한다. loop()함수 블록은 계속 동일하게 반복된다.
4. 변수의 이름은 스케치에서 대입되거나 비교될 때 사용된다. 변수의 형은 값을 저장하는 범위를 설정한다.
5. 전역변수는 스케치 전체에서 접근하고 변경할 수 있다. 지역변수는 선언된 블록 안에서 만 접근하고 수정할 수 있다.
6. 산술 연산자 덧셈 +, 뺄셈 -, 곱셈 *, 나눗셈 /, 나머지 %.
7. 이것은 소수점이 있으므로 float형의 데이터로 취급된다.
8. 초기값, 조건, 증가분.
9. 블록 주석문은 /*로 시작하여 */로 끝나며 그 사이여 여러 칸을 적을 수 있다. 라인주석문은 //로 시작하며 오른쪽에 설명을 적는다.

연습의 해답(Exercise Solutions)

1. 해답:

```
Serial.print("값 i = ");
Serial.println(i);
```

2. 해답:

```
long bigVal = 80000000;
```

3. 해답:

```
if(myVar % 2 == 0){
  Serial.println("변수는 짝수 ");
}
else{
  Serial.println("변수는 홀수 ");
}
```

4. 해답:

```
for(int i = 21; i <= 39; i+=3)
{
  Serial.print("i = ");
  Serial.println(i);
}
```

5. 해답:

```
char c = "a";
Serial.print("Character = ");
Serial.print(c);
Serial.print("    ASCII value = ");
Serial.println(c, DEC);
```

6. 해답:

```
for(char c = 'A'; c <='Z'; c++){}
```

프로젝트 해답

1. 이 스케치는 CountToTen의 변형된 버전으로 연습문제 5의 ASCII 문자를 출력해 주는 해답을 제공해 준다.

```
// ABOT - Chapter 1, 프로젝트 1
void setup(){
  Serial.begin(9600);
  for(char c = ' '; c <= '~'; c++){
    Serial.print("Character = ");
    Serial.print(c);
    Serial.print("    ASCII value = ");
    Serial.println(c, DEC);
  }
  Serial.println("All done!");
}
void loop()
{
```

```
  // 비어 있음, 반복하는 코드가 없음
}
```

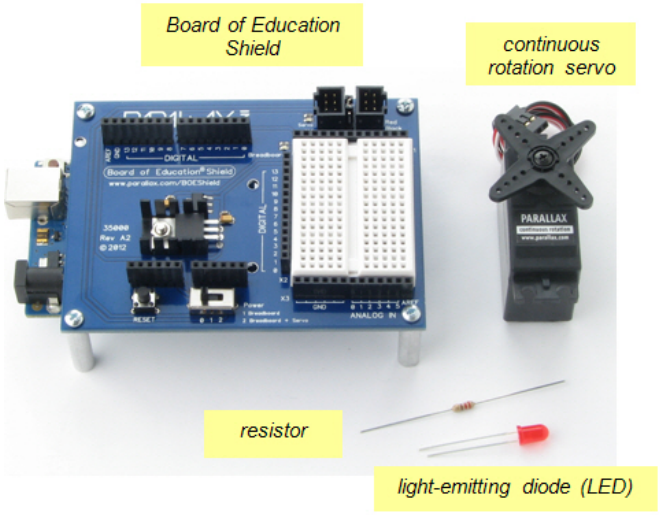
2. 이 스케치는 SimpleDecisions의 변형된 버전으로 연습문제 3의 변수 값이 홀수 짝수 인지 출력하는 해답을 제공한다.

// ABOT - Chapter 1, 프로젝트 2

```
void setup(){
  Serial.begin(9600);
  int a = 20;
  if(a%2 == 0){
    Serial.print("짝수");
  }
  else{
    Serial.print("홀수");
  }
}
void loop(){
  // 비어 있음, 반복하는 코드가 없음
}
```

제2장 쉴드, 광신호 그리고 서보모터들

여러분들이 Parallax사의 무한 회전 서보모터, 저항 그리고 광다이오드(LED)를 설치하고 테스트할 수 있는 쉴드(Board of Education Shield)의 사용법을 제2장에서 소개할 것이다. 그리고 회로를 설치하고 아두이노 보드와 교신하는 기초를 배우게 될 것이다. 제2장의 마지막에서, 여러분들은 두 개의 서보모터를 연결하고 각 서보모터의 지시신호용 불빛과 함께 서보모터의 속도 및 방향을 제어하는 스케치를 구현해볼 것이다.



- 2장 아두이노 코드 다운로드
- 데스크톱에 파일을 저장하고, 압축을 푼다. 이 파일은 예제 스케치들을 포함하고 있다.
- 시작을 위해 아래 링크를 따라 가세요.

실습1: 쉴드의 설치

쉴드(Board of Education Shield)는 아두이노 모듈로 전자 회로를 설치하고 서보모터를 연결하기 쉽도록 해준다. 이 장에서 여러분들은 서보모터와 광신호를 테스트하기 위하여 쉴드를 사용할 것이다. 다음 장에서 여러분들은 BOE쉴드와 서보모터를 ABot이라는 로봇 자동차의 섀시 위에 장착할 것이다.

부품 목록(괄호 안은 개수) :

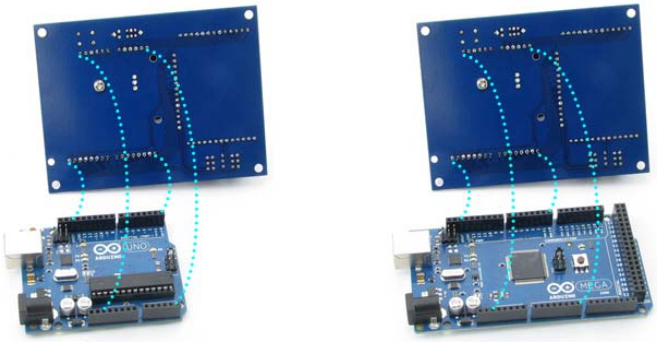
(1) 아두이노 모듈
(1) 교육용 쉴드 보드
(4) 1″ 둥근 알루미늄 받침대
(4) 납작머리 나사, 1/4″ 4-40
(3) 1/2″ 둥근 나일론 지지대
(3) 나일론 너트, 4-40
(3) 납작머리 나사, 7/8″, 4-40

소개 :

쉴드 아래 4개의 핀 그룹이 아두이노 소켓으로 연결된다. 또한, 스크루와 나일론 지지대로 아두이노 보드와 쉴드 보드를 연결하도록 설계된 3개의 구멍들을 일치시켜 연결한다.

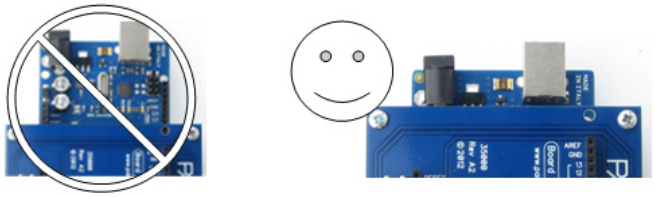
아두이노 보드의 뒷면에 UNO R3는 3번의 개선이 있었다는 것을 나타낸다. R3 보드는 두 쌍의 빈 소켓을 가지고, USB및 전원과 가깝게 위치하도록 연결할 것

이다. 이전 버전의 아두이노 보드는 쉴드의 핀 소켓 수와 동일하고, 그래서 추가로 남겨지는 빈 소켓이 없다. 만약 아두이노 메가를 가지고 있다면, 4개의 핀 그룹들이 USB및 전원과 가깝게 위치하도록 아래 그림처럼 맞춰질 것이다.

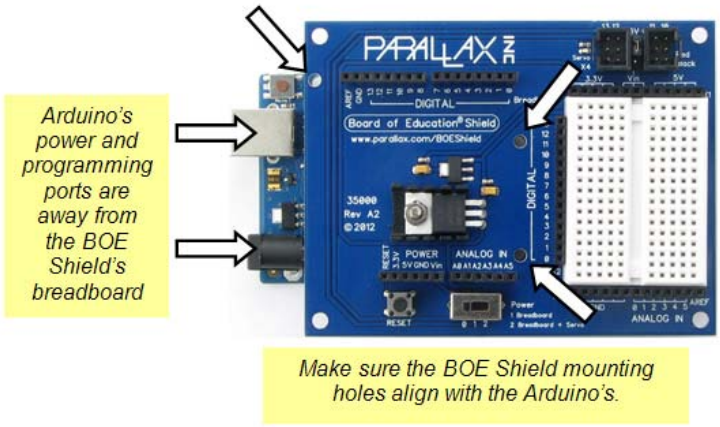


Uno R3 Mega R3

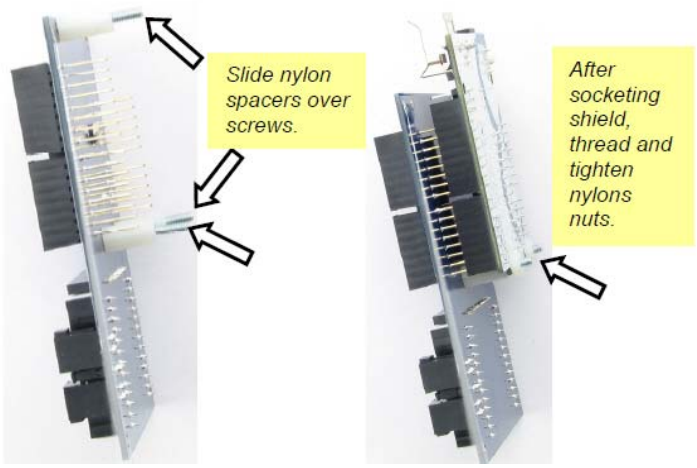
- 프로그래밍 케이블을 아두이노 모듈과 분리하시오.
- 소켓과 핀이 당신의 보드들에서 어떻게 정렬되는지 살펴보기 위해 쉴드 보드(Board of Education Shield)의 핀과 아두이노 보드를 가까이 맞춰보세요. 만약 아두이노 메가를 가지고 있다면, 아래 우측 이미지처럼 USB포트와 파워잭이 쉴드 보드의 가장자리에 더 가깝게 위치할 것이다.



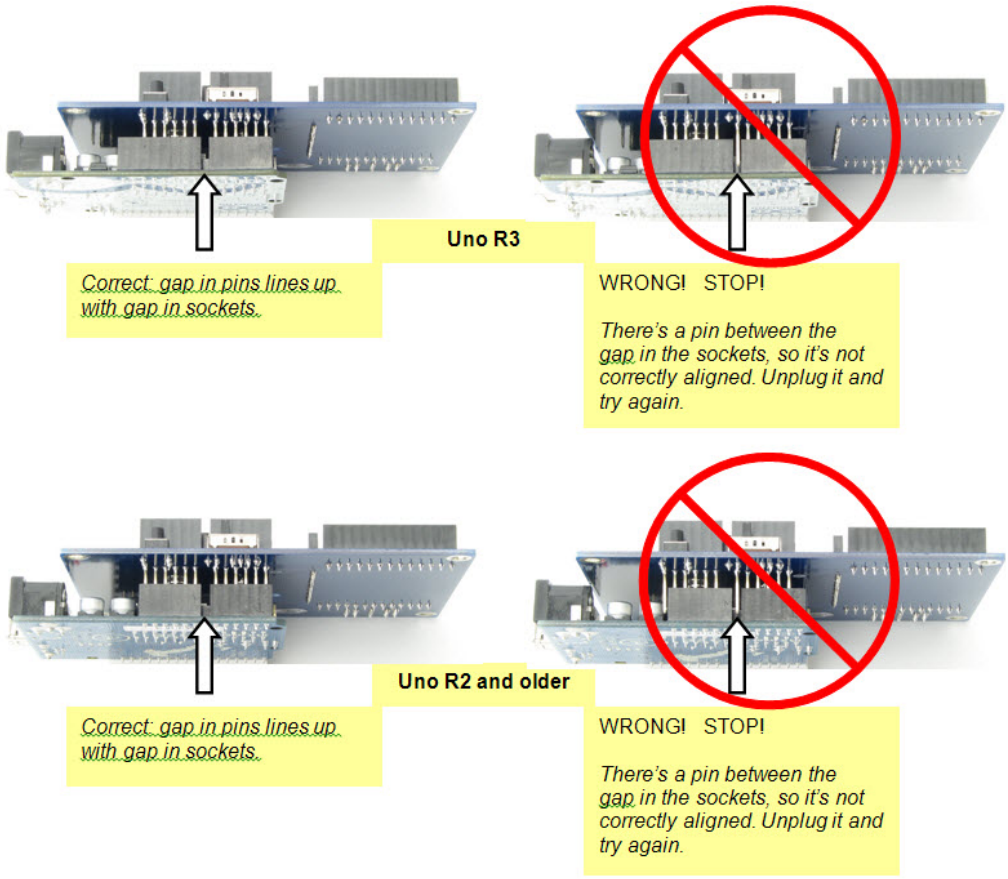
부품 위치가 아두이노 모델마다 조금씩 다르다; 어떤 것은 보드를 지지하기 위한 지지대를 1개 또는 2개만 사용할 수 있다. 이것으로 충분하지만 어느 구멍이 보드를 지지하는데 사용할 수 있는지를 찾을 필요가 있다.



- 나일론 지지대를 아두이노 모듈 구멍위에 가져가서 구멍과 완전히 일직선이 되도록 맞춰보세요.
- 아두이노 모듈의 각각의 구멍에 대하여, 7/8″나사못이 쉴드의 구멍을 통과하도록 끼워 넣으시오.

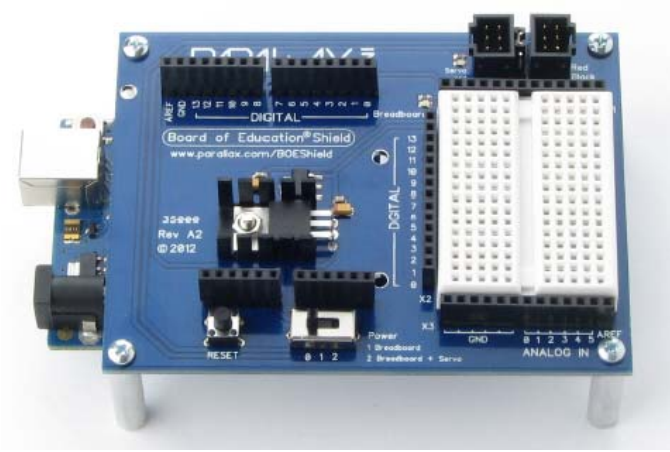


- 쉴드 핀과 아두이노 모듈 소켓을 일치시키시오.
- 또한 아두이노 보드의 구멍으로 7/8″나사못을 정렬하시오.
- 핀이 소켓에 단단히 장착되도록 두 개의 보드를 부드럽게 누르시오.



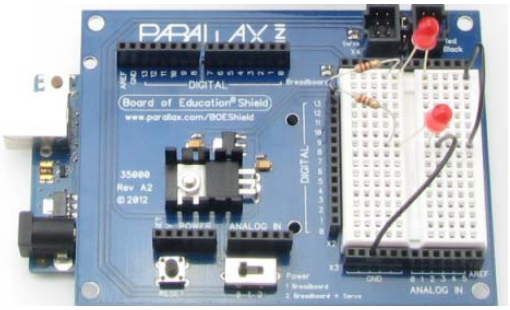
서로 연결된 보드들을 테이블과 연결하기 위하여, 우리는 쉴드의 모서리에 테이블탑 지지대(standoffs)들을 장착할 것이다.

- 쉴드의 모서리 구멍으로 1/4″나사못을 체결하시오.
- 1″ 알루미늄 지지대를 나사못으로 단단히 체결하시오.
- 4개의 지지대를 모두 체결될 때까지 반복하시오.


Board of Education® Shield
www.parallax.com/BOEShield
Rev A2
© 2012
DIGITAL
ANALOG IN
RESET
Power
GND
AREF

실습2: LED광소자 설치 및 테스트

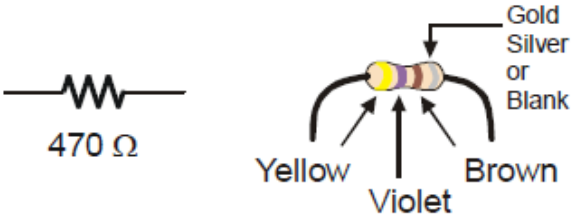
지시등은 사람들에게 소자내부가 어떤 동작 상태에 있는지를 알 수 있도록 해주거나 또는 두 개 소자 사이의 통신 패턴을 알 수 있도록 해준다. 이어서 여러분은 아두이노가 서보로 보낼 통신 신호를 표시할 지시등을 설치할 것이다. 당신이 이전에 회로를 설치해본 적이 없더라도 걱정할 필요가 없으며, 이번 과제를 통하여 여러분은 그 방법을 천천히 배울 것이다.



저항(Resistor) 이란?

저항은 전기의 흐름을 방해하는 소자이다. 이와 같은 전기의 흐름을 우리들은 전류라고 부른다. 각각의 저항은 전류의 흐름을 얼마나 강하게 방해하는지를 표시하는 값을 가지고 있다. 저항값은 옴(ohm)이라 부르고, 그리스 문자(Ω)로 표시한다.(이후에 kΩ기호를 보게 된다면 이것은 1000배의 옴을 나타내는 것이다.)

저항은 두 개의 선(또는 “리드(leeds)”라고 부른다)을 가지고 있다. 두 개 리드 사이의 세라믹 포장은 전류의 흐름을 방해하는 부분이다. 대부분의 회로도는 들쭉날쭉한 선과 어떤 값의 저항을 표시하는 숫자 표시를 사용한다. 여기서는 470 Ω저항을 개략적인 기호로 표시하고 있다. 오른쪽 그림은 여러분의 키트 회로에 사용할 저항을 알아볼 수 있도록 도움을 주는 초보자 수준의 표시 그림이다.



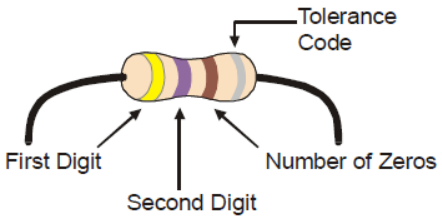
여러분 키트에서의 저항들은 어떤 저항값을 가지고 있는지를 표시할 색띠를 가

지고 있다.
저항 값을 나타내기 위해 색띠의 조합이 사용되는데, 예를 들어 470 Ω 저항을 위한 색띠는 노랑-보라-갈색 순서를 사용한다.

4번째 색띠는 저항의 오차를 표시할 수 있다. 오차는 백분율로 표시하는데, 실제 저항의 크기로부터 얼마나 벗어나는가를 말하는 것이다. 네 번째 색띠가 금색(5%) 은색(10%)일 수 있다. 이 책의 과제에서는 저항오차는 중요하지 않지만 저항 값은 중요하게 다루어질 것이다.

저항에서 각각의 색띠는 아래 표와 같이 10진수 숫자로 대응된다.

저항 색상 코드표										
숫자	0	1	2	3	4	5	6	7	8	9
색상	흑색	갈색	빨강	주황	노랑	초록	파랑	보라	회색	백색

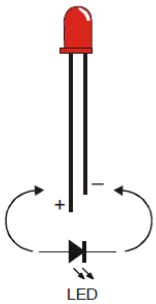


위 그림의 색상이 노랑-보라-갈색인 경우 저항값은 470Ω이 된다.
•첫째 색띠가 노랑이므로 좌측 자리숫자가 4가 된다.
•두번째 색띠가 보라이므로 다음 자리 숫자가 7이 된다.
•세번째 색띠가 갈색이므로 갈색과 대응되는 숫자는 1이 된다. 세 번째 숫자는 10진수의 지수를 의미하는 숫자이므로 10이 된다. 따라서 47의 숫자에 10을 곱한 값인 470Ω이 저항값이 된다.

LED란 무엇인가?

다이오드는 전류를 한 방향으로 흐르게하는 스위치이고 LED(*light-emitting diode*)는 전류가 흐르는 동안 광선을 방출한다. LED가 단일 방향의 전류 스위치이므로 여러분은 다이오드가 동작하기 위해 올바른 방향으로 연결되었는지를 확인해야 한다.

LED는 두 개의 단자를 가지고 있으며, 각 단자는 애노드와 캐소드로 부른다. 애노드는 +기호로 표시하고, 다이오드 심볼의 삼각형 넓은 부분에 해당한다. 캐소드는 -기호로 표시되고, 심볼 라인과 삼각형 꼭대기 부분에 해당한다.



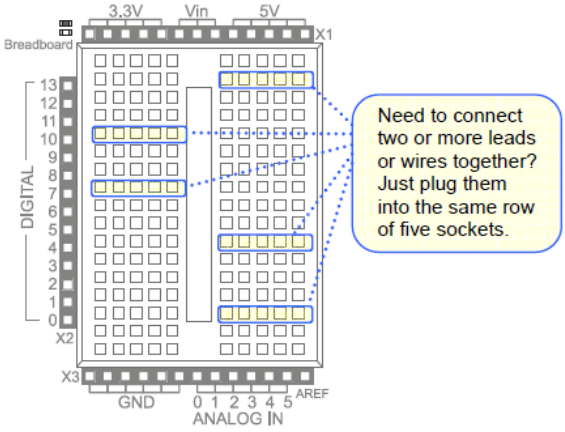
LED회로를 만들 때, 여러분은 애노드와 캐소드가 회로에 맞게 연결되도록 확인해야할 것이다. 여러분은 LED 플라스틱 모양으로 단자를 구별할 수 있다. 자세히 살펴보면 거의 둥글지만 단자의 한 쪽 가까이에서 평평한 면이 있고, 그것이 캐소드이다. 또한 LED는 서로 다른 길이의 단자를 가지고 있다. 보통 짧은 길이의 단자가 캐소드이다.

항상 플라스틱 모양을 체크하세요.

보통 긴 단자가 애노드이고 짧은 단자가 캐소드이다. 때때로 두 단자가 같은 길이로 잘려지거나, 또는 제조사가 이와 같은 규칙을 지키지 않을 수 있다. 그래서 항상 플라스틱의 평평한 면을 살펴보는 것이 최선이다. 만약 거꾸로 LED를 연결하더라도, 다이오드를 다치게 하지는 않고 바르게 연결될 때까지 광선을 방출하지 않을 것이다.

시험 제작하기

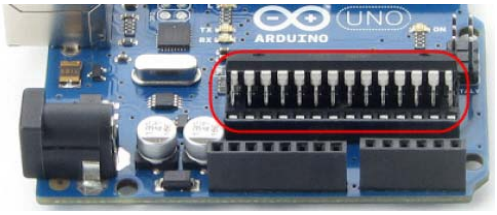
많은 소켓으로 구성된 흰색 보드를 브레드보드(*solderless breadboard*)라고 부른다. 여기 브레드보드는 17개의 행을 가지고, 각각의 행은 5개 소켓 그룹 2개가 중앙에서 분리되어 구성된다. 5개 소켓 그룹의 모든 소켓들은 아래쪽에서 금속선으로 연결되어 전기적으로 연결된 상태이다. 그래서 두 개의 전선이 5개 소켓 그룹 내에서 끼워져 있으면 전기적으로 연결된 상태이다. 이러한 방식으로 우리는 LED 저항과 같은 부품들을 연결하여 회로를 구성할 것이다. 중앙 트렌치 양쪽 동일 행을 서로 연결한 두 개의 전선은 전기적으로 연결되지 않는다.



시험제작 영역은 또한 상측 하측 좌측을 따라서 검은색 소켓을 가지고 있다.

- **상측:** 이 소켓들은 브레드보드에 3개의 공급전압을 가지고 있다 : 3.3V, Vin (배터리 또는 프로그래밍 입력전압), 5V.
- **아래쪽-좌측:** 처음 6개 소켓은 접지(GND)로 표시되고, 0V전압으로 생각한다. 종합하면 3.3V, 5V, Vin 전압과 GND가 전원 단자로 불리어지고, 여러분의 회로에 전기를 공급하는데 사용될 것이다.
- **아래쪽-우측**: 아날로그 입력(ANALOG IN) 소켓으로 여러 가지 전압을 측정하는데 사용된다. 이 단자들은 아두이노 모듈의 아날로그 입력(ANALOG IN)과 연결된다.
- **좌측**: 디지털 소켓으로 0부터 13까지 표시되어 있다. 여러분은 아두이노 모듈의 디지털 입력/출력 핀과 연결하는데 사용할 수 있다.

디지털과 아날로그 핀들은 아두이노 모듈의 마이크로 컨트롤러 칩의 작은 핀들이다. 이 핀들이 전기적으로 보드와 연결될 것이다.



스케치(프로그램)는 디지털 핀들이 하이(HIGH) 신호와 로(LOW) 신호를 회로로 보낼 수 있도록 한다. 이 장에서 우리는 빛을 켰다/껐다 할 것이다. 또한 스케치는 디지털 핀이 회로로부터 나오는 하이(HIGH)/로(LOW) 신호를 모니터할 수도 있다. 우리는 또 다른 장에서 스위치가 접촉했는지와 접촉하지 않았는지를 감지할 수 있을 것이다.

스케치는 또한 아날로그 핀으로 공급되는 전압을 측정할 수 있다. 우리는 또 다른 장에서 포토트랜지스터로 빛의 양을 측정할 것이다.

LED 실험 회로

(2) LEDs – 빨강

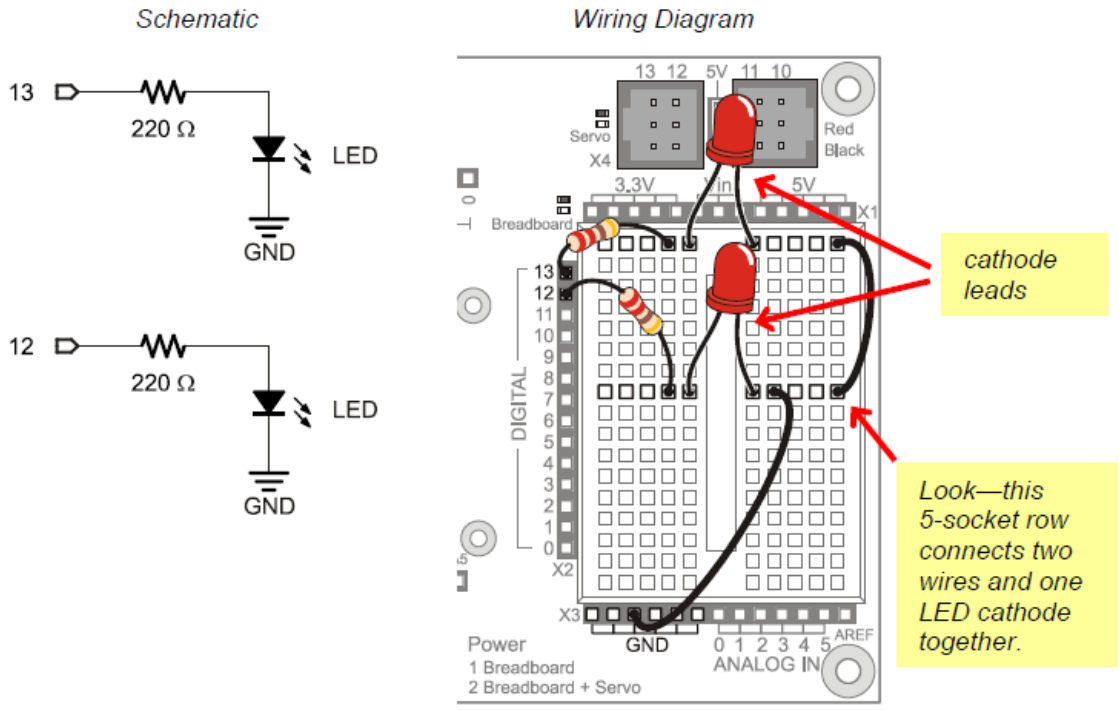
(2) Resistors, 220 Ω (red-red-brown)

회로를 만들거나 수정하기 전에는 항상 보드에 전원을 연결하지 마시오!

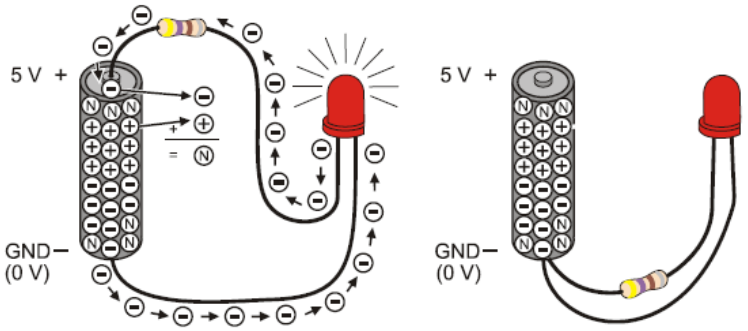
1. BOE쉴드 전원 스위치를 0으로 놓으시오.
2. 프로그램 케이블과 배터리를 분리하시오.

아래 그림 좌측은 LED회로 개략도를 나타내고, 우측은 쉴드 브레드보드에 설치된 예제 회로의 배선을 나타낸다.

- 아래 회로를 구성하세요. 처음이라면 정확하게 배선도를 따라해 보세요.
- LED캐소드 단자를 GND로 연결했는지 확인하세요. 캐소드는 LED플라스틱 케이스의 평평한 면과 가까운 짧은 길이가 캐소드 단자임을 기억하세요. 캐소드 단자는 5개 소켓 행으로 끼워져야 하고 GND 소켓으로 전선을 이용하여 연결합니다.
- 애노드 단자가 저항과 같은 5개 소켓 행에 연결되는지 확인하세요.



다음 그림은 여러분에게 LED회로를 제어할 아두이노를 프로그램할 때 어떻게 동작하는지를 여러분이 상상할 수 있도록 해준다. 5V 배터리 전원을 가지고 있다고 생각하시오. 쉴드는 5V로 표시된 소켓으로 5V를 공급하는 전압 장치를 가지고 있다. LED회로의 애노드로 5V를 연결할 때, 5V 배터리의 양극을 연결하는 것과 같다. 여러분이 회로를 GND로 연결하면 5V 배터리의 음극을 연결하는 것과 같다.



그림의 왼쪽에 의하면, LED 한 개 단자는 5V에 연결되어 있고 다른 단자는 GND에 연결되어 있다. 그래서 5V 전압은 전자를 전선 회로를 따라서 흐르도록 하고, 전류의 흐름이 LED를 빛나도록 한다. 오른쪽 회로는 GND와 연결된 LED 회로의 두 개 끝이 있다. 이것은 회로의 양쪽 끝이 똑 같이 0V가 되어서 전류가 흐르지 않고 빛이 나오지도 않는다.

여러분은 LED를 디지털 I/O핀과 연결할 수 있고, 5V와 GND사이의 핀 출력전압을 선택하도록 아두이노를 프로그램할 수 있다. 이것은 LED에서 빛이 나오게 하거나 나오지 않게 할 수 있을 것이다.

전압(Volts)의 약자 V

전압을 회로에 공급할 때, 전기적 압력을 적용하는 것이다. 보편적으로 5V는 "접지보다 5V 높다"는 것을 의미한다. 접지는 종종 GND로 줄여 말하는데 0V를 의미한다.

접지(Ground)의 약자 GND

보통의 전자 소자에서 접지는 주로 배터리 전원의 음극 단자로 연결되어 사용된다.

전류는 전자가 회로를 통과하는 비율에 관한 것이다.

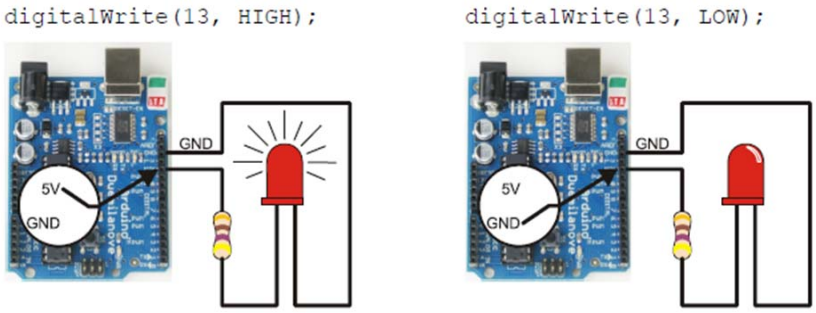
여러분은 종종 amps 또는 약자 A로 표시되는 전류 측정치를 보게될 것이다. 여러분이 사용하는 전류는 1A의 천분의 1 크기(또는 *milliamps*)로 측정된다.

LED를 켜고 끄는 스케치는 어떻게 할까?

LED회로가 디지털 13번 핀으로 연결된 LED회로가 켜지고 꺼질 수 있도록 스케치해 보자. 먼저 여러분의 스케치가 핀 13번을 출력으로 설정하기 위해 pinMode 함수를 사용한다: pinMode(*pin*, *mode*). *pin* 변수는 디지털 I/O핀의 번호이고, *mode* 는 INPUT 또는 OUTPUT 으로 설정되어야 한다.

```
void setup()                    // Built-in initialization block
{
  pinMode(13, OUTPUT);          // Set digital pin 13 -> output
}
```

이제 디지털 핀 13번이 출력으로 설정되면, 우리는 LED에서 빛이 켜지거나 꺼지도록 digitalWrite를 사용할 수 있다. 아래 그림을 살펴보자. 왼쪽 그림은 digitalWrite(13, HIGH)가 아두이노 제어부와 연결된 디지털 13번 핀에 5V를 인가하고 따라서 LED에 빛이 켜지도록 한다. 오른쪽 그림은 digitalWrite(13, LOW)가 디지털 13번 핀을 GND(0V)로 만들어서 LED가 꺼지도록 한다.



다음 스케치에서는 반복 함수를 사용한다. 먼저 digitalWrite(13, HIGH)가 LED를 켜고, delay(500)는 1/2초 동안 유지하도록 한다. 그 다음 digitalWrite(13, LOW)는 LED를 끄고, delay(500)이 1/2초 동안 유지하도록 한다. 이상의 내용이 loop 함수 내에 놓여 지면 자동으로 반복될 것이다. 결과는? 빛이 매초마다 켜지고 꺼질 것이다.

```
void loop()                          // Main loop auto-repeats
{
  digitalWrite(13, HIGH);           // Pin 13 = 5 V, LED emits light
  delay(500);                        // ..for 0.5 seconds
  digitalWrite(13, LOW);            // Pin 13 = 0 V, LED no light
  delay(500);                        // ..for 0.5 seconds
}
```

예제 스케치: HighLowLed

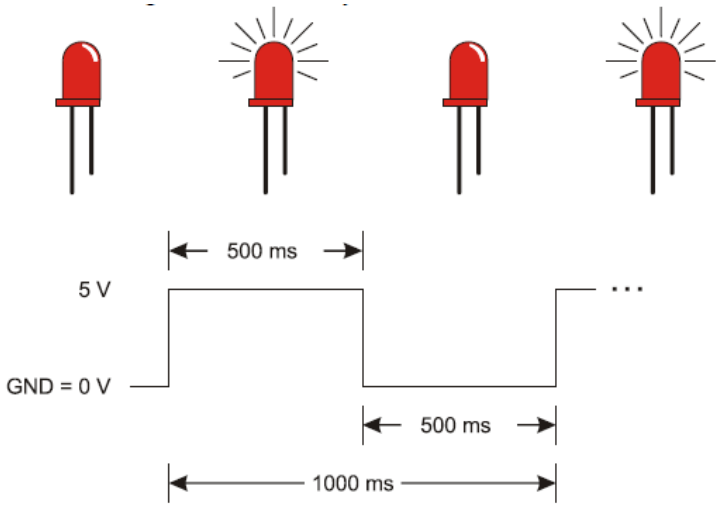
- 프로그램 케이블을 여러분의 보드로 연결하시오.
- 입력하고 저장하고 HighLowLed를 아두이노로 업로드 하시오.
- 13번 핀 LED가 매초마다 켜지고 꺼지는지 점검하시오.

/* Turn LED connected to digital pin 13 on/off once every second. */

```
void setup()                          // Built-in initialization block
{
  pinMode(13, OUTPUT);                   // Set digital pin 13 -> output
}

void loop()                           // Main loop auto-repeats
{
  digitalWrite(13, HIGH);            // Pin 13 = 5 V, LED emits light
  delay(500);                         // ..for 0.5 seconds
  digitalWrite(13, LOW);             // Pin 13 = 0 V, LED no light
  delay(500);                         // ..for 0.5 seconds
}
```

타임 챠트 만들기

타임챠트는 HIGH 신호와 LOW 신호가 시간에 따라 동작하는 그래프이다. 이 타임챠트는 HIGH (5 V)와 LOW (0 V)신호의 1000 ms 조각으로 보인다. delay(500)이 어떻게 깜박임 비율을 조절하는지를 알 수 있을까?



LED의 깜박임 비율 실험

깜박임를 두 배 빠르게 만들 수 있을까? 지연함수의 시간 파라미터를 반으로 어떻게 줄일 수 있을까?

- delay(250)을 사용하도록 여러분의 스케치를 수정하시오. 두 곳을 수정하는 것을 잊지 마시오.

12번 LED를 깜박이는 것은 pinMode에서 핀 파라미터와 호출하는 두 개의 digitalWrite 함수를 바꾸는 간단한 작업이다.

- setup 함수에서 pinMode가 13번 핀을 대신하여 12번을 사용하도록 스케치를 수정하시오.
- 또한 반복문 내의 digitalWrite문장이 12번 핀을 사용하도록 수정하시오.
- RUN 동작하고, 12번 LED가 깜박이는지 확인하시오.

여러분은 또한 동시에 두 개의 LED가 깜박이도록 할 수 있다.

- pinMode 두 개를 사용하도록 스케치에 아래 문장을 추가하시오.

```
pinMode(13, OUTPUT);          // Set digital pin 13 -> output
pinMode(12, OUTPUT);          // Set digital pin 12 -> output
```

- ….그리고 digitalWrite를 네 번 사용하시오:

```
digitalWrite(13, HIGH);       // Pin 13 = 5 V, LED emits light
digitalWrite(12, HIGH);       // Pin 12 = 5 V, LED emits light
delay(500);                    // ..for 0.5 seconds
digitalWrite(13, LOW);        // Pin 13 = 0 V, LED no light
digitalWrite(12, LOW);        // Pin 12 = 0 V, LED no light
delay(500);                    // ..for 0.5 seconds
```

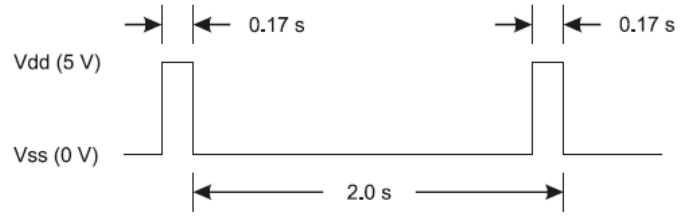
- 수정된 스케치를 RUN동작하시오. 두 개 LED가 함께 켜지고 꺼집니까?

한 개 LED가 켜지면 다른 한 개 LED가 꺼지도록 스케치를 어떻게 수정하나요? 한 개의 회로가 HIGH신호를 받으면, 다른 한 개의 회로가 LOW신호를 받아야 합니다. 시도해 보세요!

실습3: LED 서보 신호 감시

모터를 구동하는 HIGH와 LOW신호는 매우 정확한 시간 주기를 유지해야 한다. 그것은 서보모터가 HIGH상태를 얼마나 오래 동안 유지해야 하는지를 측정하기 때문이다. 그리고 얼마나 빨리 어느 방향으로 모터를 회전해야하는지에 이용하기 때문이다.

여기 타임챠트는 여러분의 쉴드 바퀴가 시계방향으로 전속 회전하도록 하는 서보 신호를 나타낸다. 그렇지만 하나의 큰 차이가 있다: 타임챠트의 모든 신호들은 서보를 제어하는 것 보다 100배 더 길게 유지된다. 이것은 어떤 것이 진행되는 것을 우리가 알 수 있을 만큼 충분히 느리다.



예제 스케치: ServoSlowMoCcw

- ServoSlowMoCcw를 아두이노에 입력하고 저장하고 업로드 하시오.
- 13번 핀 LED회로가 2초마다 펄스를 가지는지 점검하시오.

```
/*
    쉴드와 결합된 로봇 - ServoSlowMoCcw
    LED로 보이는 서보 신호의 1/100번째 속도를 전송하시오.
 */
```

```
void setup()                                // Built in initialization block
{
  pinMode(13, OUTPUT);                        // Set digital pin 13 -> output
}
```

```
void loop()                                // Main loop auto-repeats
{
  digitalWrite(13, HIGH);                 // Pin 13 = 5 V, LED emits light
  delay(170);                              // ..for 0.17 seconds
  digitalWrite(13, LOW);                  // Pin 13 = 0 V, LED no light
  delay(1830);                             // ..for 1.83 seconds
}
```

2단계 서보 신호

좋아요, 1/100번째 속도 대신 1/10번째 속도는 어떻게 하나요?

- delay(170)을 delay(17)로 줄이고, delay(1830)은 delay(183)으로, 그리고 스케치를 다시 업로드 하시오.

이제 LED 깜박임이 10배 빨라집니까? 서보 신호를 다시 10으로 나누시오:

- delay(17)은 delay(2)로 바꾸고, delay(183)은 delay(18)로 바꾸고, 그 다음 수정된 스케치를 업로드 하시오.

이제 여러분은 서보신호가 LED 지시등과 비슷하게 보인다는 것을 알 수 있다. HIGH신호가 1.7ms 대신 2ms이기 때문에 LED가 실제 서보제어 신호보다 조금 더 밝아질 것이다. 우리는 서보를 제어할 신호와 프로그래밍 수단을 사용하지만, 더 쉽고 더 정확한 방법이 있다. 먼저 LED로 그것을 시도해보자.

아두이노 서보 라이브러리 사용법

서보 신호를 만드는 좋은 방법은 여러분의 스케치내에 아두이노 서보 라이브러리를 포함하는 것이다. 이것은 표준 라이브러리의 하나로 아두이노 소프트웨어에서 제공되는 번들 코드이다.

- 아두이노 라이브러리의 목록을 살펴보고, 아두이노 소프트웨어의 Help menu를 클릭하고 Reference를 선택하시오.
- 라이브러리 링크를 따라하시오.

우리는 서보 라이브러리를 더 자세히 살펴보고 싶다.

- 서보 링크를 따라하시오.
- 서보 라이브러리 페이지의 다음 함수들을 위한 링크를 읽고 따라하시오:

  attach()

  writeMicroseconds()

  detach()

서보는 규칙적인 시간 간격으로 HIGH 펄스 제어신호를 수신해야 한다. 만약 신호가 멈추면 서보가 동작하지 않는다. 여러분의 스케치가 신호를 만들기 위해 서보 라이브러리를 사용하면, 그것은 지연 또는 센서 체크와 같이 다른 코드로 옮겨갈 수 있다. 더구나 서보 라이브러리가 배후에서 동작하기 때문에 서보는 계속 회전한다. 그것은 실제 알 수 없을 정도로 빠른 HIGH펄스의 개시를 위한 또 다른 코드의 실행을 규칙적으로 방지한다.

서보 제어 신호를 보내기 위해 서보 라이브러리를 사용하는 것의 4단계:

7. 아두이노 저자는 여러분이 스케치 시작 선언에서 setup 함수 이전에 서보 라이브러리에 엑세스하기를 원한다고 말한다.

   ```
   #include <Servo.h>              // Include servo library
   ```

8. #include와 setup 함수 사이에 여러분이 전송하고 싶은 신호를 위한 서보 라이브러리의 사례를 선언하고 이름을 적으시오.

   ```
   Servo servoLeft;                // Declare left servo
   ```

9. setup 함수에서, 도트 이후에 서보 신호에 전달할 이름을 사용하시오. 그리고 신호 핀을 호출하는 함수를 사용하시오. 이번 예제는 servoLeft로 이름 붙여진 서보신호가 디지털 핀 13번으로 전달되어야 하는 시스템을 말하고 있다.

   ```
   servoLeft.attach(13);        // Attach left signal to pin 13
   ```

10. 펄스 시간을 설정하기 위해 writeMicroseconds함수를 사용하시오. 당신은 setup과 loop함수 어느 것의 내부에 위 함수를 사용할 수 있다.

    ```
    servoLeft.writeMicroseconds(1500);  // 1.5 ms stay-still
    ```

signal

Seconds, Milliseconds, Microseconds

millisecond 는 천분의 1초이고 약자는 ms이다.

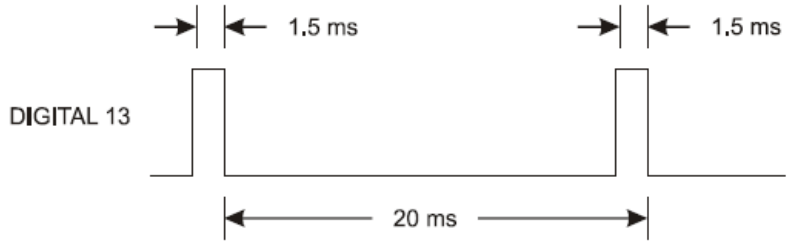
A *microsecond* is a one-millionth of a second, abbreviated µs.

1 millisecond (ms)내에는 1000 microseconds (µs)가 포함되어 있다.

1 second (s)내에는 1,000,000 microseconds가 포함되어 있다.

예제 스케치: LeftServoStayStill

서보를 계측하기 위하여, 여러분의 스케치는 1.5ms 펄스 신호를 보낼 필요가 있다. 아래 타임챠트를 살펴보시오. 신호의 HIGH펄스 시간이 1.5ms 동안 지속됩니다. 그것은 1.7ms 반시계 방향 펄스와 1.3ms 시계방향 펄스의 중간지점입니다.



- 아두이노에 LeftServoStayStill을 입력하고 저장하고 업로드 하시오. 핀 13 LED 는 이전에 관찰했던 두 개의 밝기 수준사이의 중간에 해당합니다.

```
/*
 쉴드에 장착된 로봇 - LeftServoStayStill
 Generate signal to make the servo stay still for centering.
 */
```

```
#include <Servo.h>                          // Include servo library
Servo servoLeft;                            // Declare left servo
```

```
void setup()                                  // Built in initialization block
{
  servoLeft.attach(13);                      // Attach left signal to pin 13
  servoLeft.writeMicroseconds(1500);          // 1.5 ms stay still signal
}
void loop()                                   // Main loop auto-repeats
{                                              // Empty, nothing needs repeating
}
```

핀 12 LED에 두 번째 제어신호 점검하기

여러분은 이번 코드를 많이 사용할 것이다. 그래서 신호를 핀으로 연결하고 펄스 기간을 설정하는 서보의 인스턴스 선언 방법을 연습하는 것이 좋다.

- BothServosStayStill처럼 LeftServoStayStill를 저장하시오.
- 두 번째 서보를 선언하여 추가하고 servoRight라고 이름 붙이시오.

```
Servo servoRight;                    // Declare right servo
```

- 여러분의 servoRight신호를 디지털 핀 12번에 연결하시오.

```
servoRight.attach(12);               // Attach right signal to pin 12
```

- servoRight 신호의 펄스를 1.5 ms (1500 µs)로 설정하시오.

```
servoRight.writeMicroseconds(1500); // 1.5 ms stay still signal
```

- 이제 여러분의 스케치는 BothServosStayStill처럼 보여야 한다.
- 아두이노에 저장된 스케치를 업로드 하시오.
- 두 개의 LED가 비슷한 밝기 수준을 나타내는지 점검하시오.

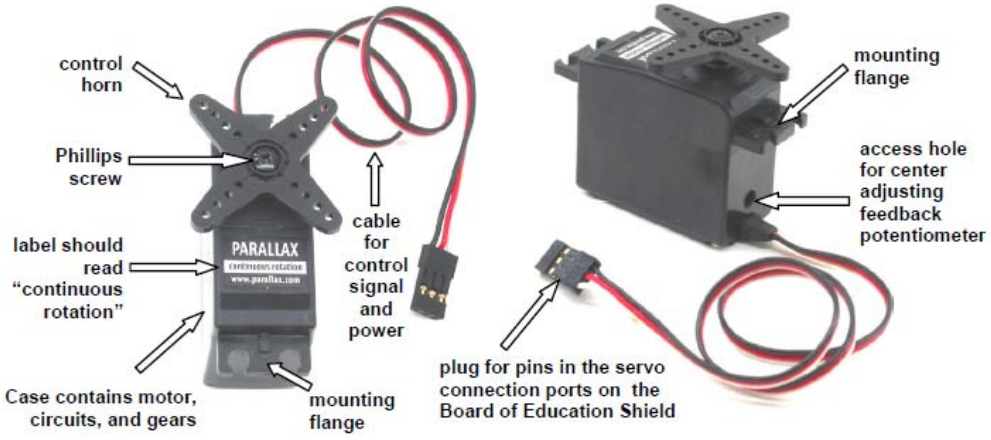
```
/*
 쉴드에 장착된 로봇 - BothServosStayStill
 Generate signals to make the servos stay still for centering.
 */
```

```
#include <Servo.h>                                  // Include servo library
```

```
Servo servoLeft;                          // Declare left servo signal
Servo servoRight;                         // Declare right servo signal

void setup()                              // Built in initialization block
{
  servoLeft.attach(13);                   // Attach left signal to pin 13
  servoRight.attach(12);                  // Attach left signal to pin 12

  servoLeft.writeMicroseconds(1500);      // 1.5 ms stay still sig, pin 13
  servoRight.writeMicroseconds(1500);     // 1.5 ms stay still sig, pin 12
}
void loop()                               // Main loop auto-repeats
{                                         // Empty, nothing needs repeating
}
```

실습4: 서보모터와 배터리 연결

로봇을 조종하는 관점에서 연속회전 서보는 간단함의 조합이고 저렴한 가격과 유용함을 의미한다. Parallax의 연속회전 서보는 ABOT 바퀴를 아두이노 컴퓨터로 회전하게 만드는 모터이다.



이번 과제에서, 여러분은 서보를 쉴드의 서보 포트로 연결하고, 이어서 전압 공급과 접지와 신호핀을 연결할 것이다. 또한 여러분은 어떤 상황에서 USB가 공급하는 전류보다 더 많은 요구에 처하게 되면 아두이노에 배터리 전원을 연결하게 될 것이다.

표준 서보와 연속회전 서보

표준 서보는 어떤 위치를 나타내는 전자신호를 수신하도록 설계된다. 서보들은 비행기 조종날개 배의 방향타 자동차 조종장치의 제어범위 내에서 위치를 제어한다. 연속회전 서보는 동일한 전자신호를 수신하지만, 어떤 속도와 방향으로 회전한다. 연속회전 서보는 바퀴와 도르래를 제어하기 쉽다.

서보 제어 혼(servo Control Horn), 4-point Star vs. Round

4-point Star와 Round는 차이가 없다. 연속회전이면 그것은 ABOT을 위한 서보이다. 여러분은 control horn을 제거하고 제거된 자리에 바퀴를 장착할 것이다.

쉴드에 서보 연결하기

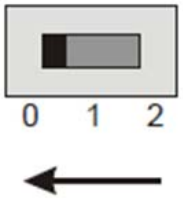
여러분 보드의 마지막 과제에서 LED회로를 남겨두시오. 아두이노가 서보의 움직임을 제어하기 위해 서보로 보내는 신호를 모니터하는데 사용될 것이다.

부품 목록

(2) 패럴렉스(Parallax) 연속회전 서보
이전 과제에서 LED 지시등을 점검하고 설치한 BOE 쉴드

사용 설명

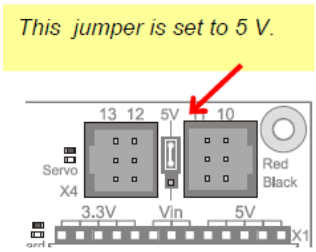
- 쉴드의 전원 스위치를 0으로 놓으시오.

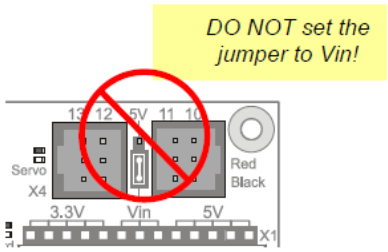


- USB 케이블을 포함하는 아두이노로부터 모든 전원을 분리하시오.

쉴드의 서보 헤더들 사이에서 Vin 또는 5V로 서보 전원을 공급할 수 있는 점퍼가 있다. 점퍼를 제거하려면 위로 들어 올리고 이후 다른 핀 쌍으로 점퍼를 눌러 끼운다. ABOT 배터리는 7.5V를 공급한다. 서보가 4-6V 사이에서 변화하기 때문에, 우리는 점퍼가 5V가 되도록 확인할 것이다. 또한 일정한 5V 전압원은 지속적인 서보 속도를 유지시키고 배터리 방전과 같이 변화하는 전압보다 더 정확한 조종을 지원한다.

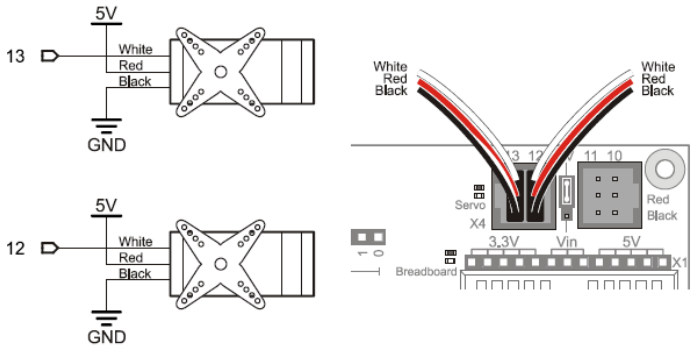
- 쉴드 전원 점퍼가 5V에 맞춰졌는지 확인하시오. 그렇지 않다면 지금 바르게 하시오.





아래 그림은 쉴드에 포트 13과 12로 서보를 연결한 회로의 개요도를 보여준다. 여러분이 연결한 케이블의 전선 색상을 주의 깊게 살펴보시오. 검은색은 아래쪽에 있어야 하고, 흰색은 위쪽에 있어야 한다.

- 그림 2-20에 보이는 것처럼 쉴드로 서보를 연결하시오.
- 그림에 보이는 케이블 색상처럼 보드에 검은색 전선이 가깝고 보드 가장자리에 흰선이 가깝도록 되어 있는지 확인하세요.



배터리 팩을 쉴드에 연결하기

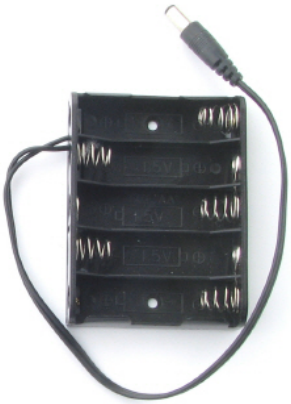
서보에 전원을 공급하기 위하여, 여러분은 지금 외부 배터리 팩으로 바꿀 필요가 있다. 서보가 갑자기 방향을 바꿀 때 또는 회전 저항을 밀고나갈 때, USB 포트보다 더 많은 전류가 사용되도록 설계된다. 또한 ABOT이 컴퓨터와 영원히 함께 동작하지는 않는다. 그래서 우리는 1.5V AA배터리 5개로 구성된 배터리 팩을 사용할 것이다. 이것은 여러분의 장치를 7.5V로 공급하고 전압 조절장치와 서보를 위해 더 많은 전류를 공급한다. 여기서부터 2가지를 기억하시오.

- **여러분이 실험을 마쳤다면 항상 배터리 팩을 뽑아 놓으세요.** 심지어 BOE 쉴드의 전원 스위치가 꺼진 상태(위치-0)일 때, 아두이노 모듈은 배터리로부터 전력을 사용하지 않는다.
- **또한 배터리 팩을 뽑을 때마다 프로그래밍 케이블을 뽑으세요.** 그것은 여러분이 우연히 서보를 동작시키지 않을 것이다.

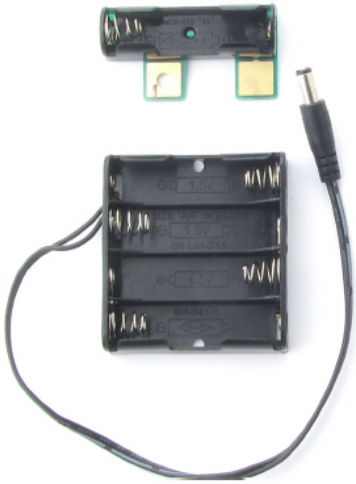
여러분은 어떤 배터리 팩을 가지고 있나요?

- 여러분이 어떤 배터리 팩을 가지고 있는지를 아래 그림과 비교하세요.
- 5셀 팩은 다음 페이지로 가세요.
- 4셀 팩은 4-Cell Pack + Boe-Boost Setup.으로 가세요.

5-cell pack



4-cell pack
+ Boe-Boost



충전 옵션

절약형 Boe-Boost (#30078) 는 4셀에 직렬로 하나의 셀을 더 추가하거나 5셀 팩을 허용한다. 5셀 팩에 6번째 1.2V AA 충전 셀을 더하면 7.2V를 공급할 수 있다.

Li-ion Boe-Bot Power Pack-Charger (#28988) 는 리튬이온 배터리 팩과 충전기를 보드에서 결합한 것이다.

주의 : 교류(AC)로 직류(DC) 전원을 ABOT에 공급하지 않는다.

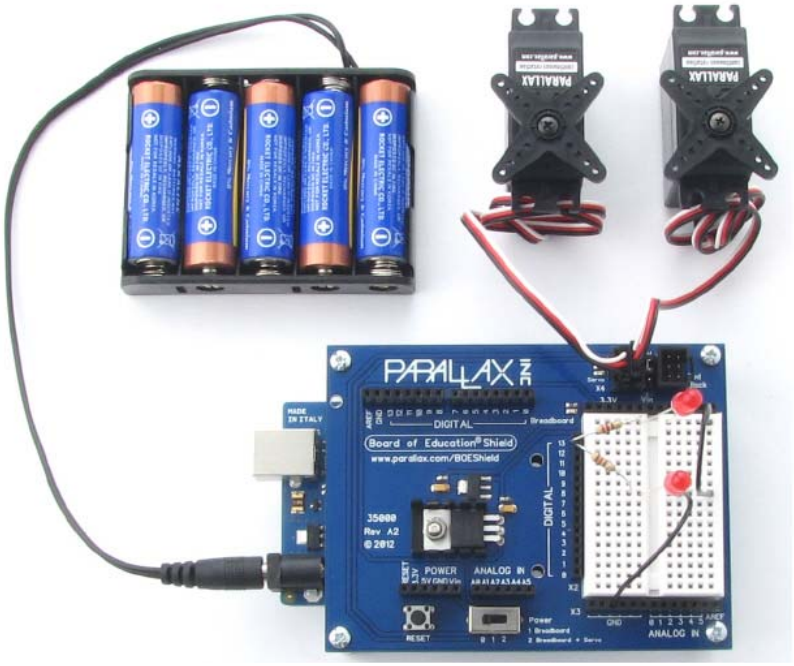
어떤 DC전원은 설계보다 더 높은 전압을 공급한다. ABOT은 7.2-7.5 V 배터리 공급전압을 사용하도록 설계되었다. 그것은 낮은 부하에서 더 높은 공급전압으로 동작하지만 서보 부하는 조절장치가 멈출 때까지 조절장치를 가열시킬 수 있다.

5셀 팩 설치

부품 목록

(5) AA 알카라인 배터리
(1) 5셀 배터리 팩

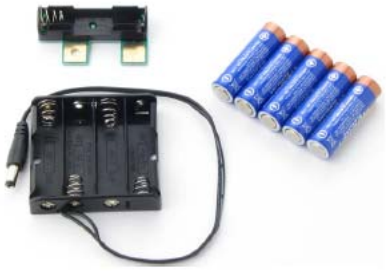
- 배터리를 배터리팩으로 끼우세요.
- 배터리팩을 아두이노 전원 잭으로 끼우시오. 마쳤다면 아래 그림과 같아야 한다.
- Centering the Servos.로 가세요.



4셀 팩과 보부스터 설치

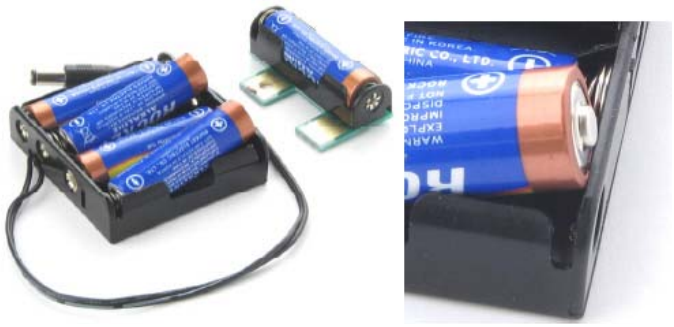
부품 목록

(1) 보부스터(Boe-Boost)
(1) 4셀 배터리 팩
(5) AA 알카라인 배터리



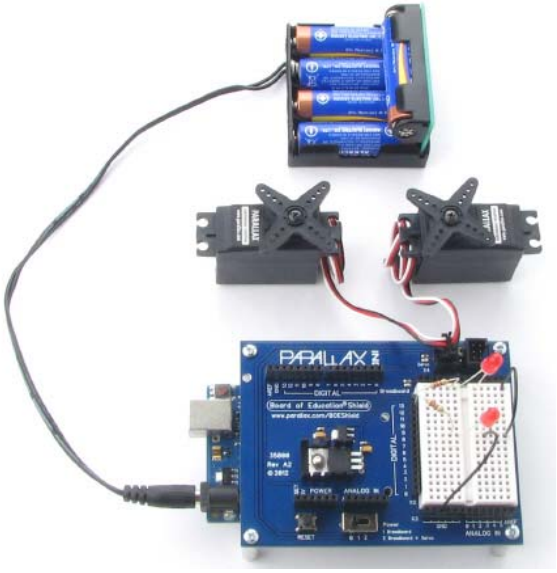
- 배터리 팩으로 배터리와 보부스터를 끼워 넣으세요. 알카라인 배터리의 +/-마크를 배터리 케이스의 +/-마크와 맞추시오.

- 아래 그림을 사용하고 두 개 배터리의 양극 끝을 위로 들어 올리시오.



- 보부스터를 양극 끝부분으로 밀어 넣고, 배터리 홀더로 밀어넣기 위해 누르시오.

- 아두이노 파워 잭으로 배터리 팩을 끼우시오.
- 당신의 작업을 점검하시오; 여러분이 장치 조립을 마쳤을 때 아래 그림과 같아야 한다.



실습5: 서보 중심(centering) 맞추기

이번 과제에서, 여러분은 “stay-still”신호를 서보로 보내는 스케치를 동작시킬 것이다. 여러분은 서보가 여전히 멈춘 상태가 되도록 서보를 맞추기 위하여 스크루 드라이버를 사용할 것이다. 이것이 서보 중심 맞추기이다. 조정 이후에 여러분은 서보가 시계방향으로 반시계방향으로 회전하는 스케치를 테스트할 것이다

필요한 도구

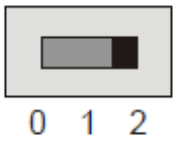
여러분은 1/8″ (3.18 mm) 드라이버가 필요하다.

- 

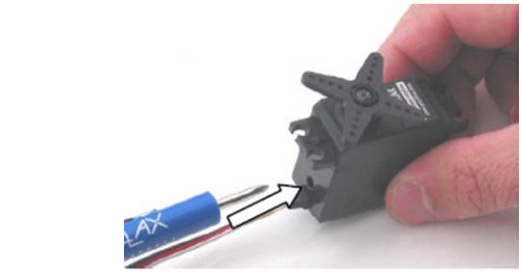
중심 신호 전송

서보 중심이 맞지 않다면, “stay-still”신호를 수신할 때 회전하거나 진동하거나 소음을 일으킬 수 있다.

- 프로그래밍 케이블을 분리하고 LeftServoStayStill를 다시 동작시키시오.
- BOE쉴드 전원 스위치를 2에 놓고, 전원을 서보에 공급하시오.



- 서보의 전위차계를 조정하기 위해 스크루 드라이버를 사용하시오. 너무 세게 누르지 마시오. 서보가 회전하거나 소음을 내거나 진동하지 않도록 만드는 조정점을 찾을 때까지 가볍게 전위차계를 조정하시오.



- 13번 핀 LED 신호 모니터 회로가 동작하는지를 점검하시오.

전위차계(Potentiometer)가 무엇인가?

전위차계(*potentiometer*)는 손잡이와 슬라이딩 바와 같은 움직임 부분을 포함하는 일종의 가변저항이다. 패럴렉스 연속회전 서보 전위차계는 작은 스크류 드라이버 팁으로 조정할 수 있는 손잡이이다.

핀 12번으로 연결된 서보의 중심 맞추기

- 스케치 RightServoStayStill를 사용한 핀12번에 대하여 과정을 반복하시오.

```
/*
 Robotics with the BOE Shield - RightServoStayStill
 Transmit the center or stay still signal on pin 12 for center adjustment.
 */
```

```
#include <Servo.h>                          // Include servo library

Servo servoRight;                           // Declare right servo
```

```
void setup()                                // Built-in initialization block
{
  servoRight.attach(12);                    // Attach right signal to pin 12
  servoRight.writeMicroseconds(1500);       // 1.5 ms stay still signal
}

void loop()                                 // Main loop auto-repeats
{                                           // Empty, nothing needs repeating
}
```

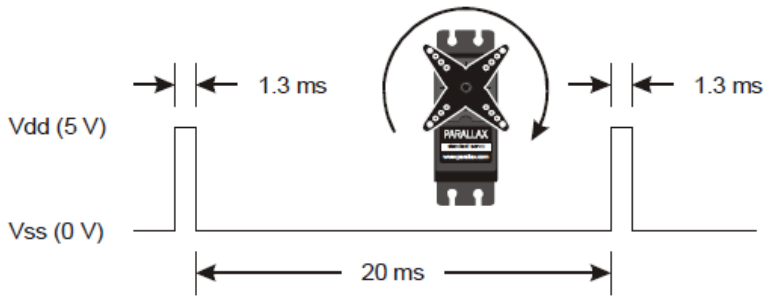
실습6: 서보 검사

ABOT을 조립하기 전에 마지막 남은 한 가지가 있다. 그것은 서보 검사이다. 이번 과제에서 여러분은 다른 속도와 방향으로 서보가 회전하도록 하는 스케치를 실행할 것이다. 다음이 *subsystem testing*의 예제이다.

Subsystem testing은 더 큰 소자로 가기 전에 각각의 부품들을 점검하는 연습이다. 그것은 로보틱 시합을 승리하도록 도와주는 유용한 전략이 된다. 또한, 기술자가 장난감 자동차 그리고 비디오 게임에서 우주선 화성 탐사 로봇까지 모든 것을 개발하는데 이용하는 기본 기술이다. 특히 더 복잡한 장치에서 각각의 부품이 점검되지 않는다면 문제점을 찾아내는 것이 거의 불가능할 수 있다. 예를 들어 우주 프로젝트에서 한 개의 문제를 수정하기 위해 시험 제작품을 분해하는 것은 수십만 수백만 달러의 비용을 필요로 한다.

펄스 폭은 속도와 방향을 제어

다음 타임챠트는 1.3ms 펄스를 보낸 때 패럴렉스 연속회전 서보가 시계방향으로 어떻게 회전하는지를 보여준다. 일반적으로 전속은 50~60 RPM 범위에 있다.



RPM이란? 분당 회전수—1분에 완전 회전하는 회전수.

pulse train이란? 기차는 일련의 차량을 의미하는 것처럼, *pulse train* 은 일련의 펄스를 의미한다.

예제 스케치: LeftServoClockwise

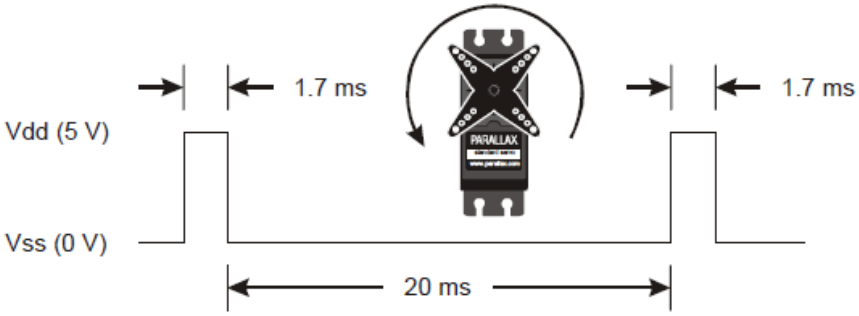
- LeftServoClockwise입력하고 저장한후 업로드 하시오.
- 서보 혼(horn)이 50-60 RPM 시계방향으로 회전하는지를 점검하시오.

```
/*
 Robotics with the BOE Shield - LeftServoClockwise
 Generate a servo full speed clockwise signal on digital pin 13.
 */
```

```
#include <Servo.h>                          // Include servo library

Servo servoLeft;                            // Declare left servo

void setup()                                // Built in initialization block
{
  servoLeft.attach(13);                     // Attach left signal to pin 13
  servoLeft.writeMicroseconds(1300);        // 1.3 ms full speed clockwise
}

void loop()                                 // Main loop auto-repeats
{                                           // Empty, nothing needs repeating
}
```

Left Servo Counterclockwise

- LeftServoClockwise를 LeftServoCounterclockwise로 저장하시오.
- servoLeft.writeMicroseconds에서 (1300) 을 (1700)으로 변경하시오.
- 수정된 스케치를 저장하고 아두이노로 업로드 하시오.
- 핀 13번에 연결된 서보가 다른 방향으로 회전하는지를 점검하시오. 이것은 약 50-60RPM의 반시계 방향이어야 합니다.



예제 스케치: RightServoClockwise

- LeftServoClockwise 를 RightServoClockwise로 저장하시오.
- servoLeft의 모든 인스턴스를 servoRight로 교체하시오.
- 13의 모든 인스턴스를 12로 교체하시오.
- 스케치를 실행하고 12번 핀 서보가 50-60 RPM 시계방향 회전하는지를 점검하시오.

```
/*
 Robotics with the BOE Shield - RightServoClockwise
 Generate a servo full speed clockwise signal on digital pin 12.
 */
```

```
#include <Servo.h>                          // Include servo library

Servo servoRight;                           // Declare left servo

void setup()                                // Built in initialization block
{
  servoRight.attach(12);                    // Attach left signal to pin 12
  servoRight.writeMicroseconds(1300);       // 1.3 ms full speed clockwise
}

void loop()                                 // Main loop auto-repeats
{                                           // Empty, nothing needs repeating
}
```

Right Servo Counterclockwise

- servoRight.writeMicroseconds에서 (1300)을 (1700)으로 바꾸시오.
- 스케치를 저장하고 아두이노로 업로드 하시오.
- 핀 12번 서보가 반시계 방향으로 50-60RPM 회전하는지를 점검하시오.

서보 속도와 방향을 제어하기

ABOT 조종을 위하여 우리는 한 번에 두 개의 서보를 제어할 필요가 있다.

- ServosOppositeDirections을 아두이노로 입력하고 저장하고 업로드 하시오.
- 핀13에 연결된 서보는 반시계방향으로 핀12에 연결된 다른 서보는 시계방향으로 회전하는지를 점검하시오.

예제 스케치: ServosOppositeDirections

```
/*
Robotics with the BOE Shield - ServosOppositeDirections
핀 13번으로 반시계방향 신호를 생성하고 핀 12번으로 시계방향 신호를 생성하시오.
*/
```

```
#include <Servo.h>                          // Include servo library
Servo servoLeft;                            // Declare left servo signal
Servo servoRight;                           // Declare right servo signal

void setup()                                // Built in initialization block
{
  servoLeft.attach(13);                     // Attach left signal to pin 13
  servoRight.attach(12);                    // Attach right signal to pin 12

  servoLeft.writeMicroseconds(1700);        // 1.7 ms -> counterclockwise
```

```
  servoRight.writeMicroseconds(1300);          // 1.3 ms -> clockwise
}

void loop()                                    // Main loop auto-repeats
{                                              // Empty, nothing needs repeating
}
```

반대방향 제어는 곧 중요해질 것이다. 생각해보자: 서보가 섀시에 장착되었을 때 BOE 쉴드-봇이 직선을 구르기 위해서는 하나의 서보가 시계방향으로 회전하면 다름 한 개는 반시계방향으로 회전해야 할 것이다. 이상하게 생각되는가? 상상이 안된다면 시도해보자.

- 스케치가 실행되는 동안 여러분의 서보를 함께 들어보자.

펄스폭 변조(*modulation*)

정보를 전달하는 신호의 양을 조정하는 것을 변조라고 부른다. 우리는 서보 제어 신호가 일련의 HIGH 신호이고 나머지는 LOW 상태로 분리된 상태라는 것을 알고 있다. HIGH 펄스가 얼마나 오래 지속되는가가 서보 회전 속도와 방향을 결정한다. 조정 가능한 펄스 폭이 서보 조작 정보이다. 그러므로 우리는 서보가 펄스 폭 변조로 제어된다고 말할 수 있다.

ABOT의 동작을 프로그램하기 위하여 writeMicroseconds의 다른 조합들 *us* 파라미터가 반복적으로 사용될 것이다. 빈칸을 채우고 여러 가지 가능한 조합을 테스트함으로써 여러분은 친숙해지고 기초를 다지게 될 것이다.

- TestServosTable2_2처럼 ServosOppositeDirections 스케치의 복사본을 저장하시오.
- servoLeft.WriteMicroseconds(*us*)에서 각각의 조합 값을 테스트하고 Table 2-2—a에 그 결과를 기록하시오.

Table 2-2: writeMicroseconds(us) Combinations			
Pin 13 servoLeft	Pin 12 servoRight	Description	Behavior
1700	1300	Full speed, pin 13 servo counter-clockwise, pin 12 servo clockwise.	
1300	1700		
1700	1700		
1300	1300		
1500	1700		
1300	1500		
1500	1500	Both servos should stay still (see Activity #5, page 64.)	
1520	1480		
1540	1460		
1700	1450		
1550	1300		

서보 실행 시간 제어방법

서보 라이브러리를 사용할 때 서보가 얼마나 오래동안 실행되는가를 제어하는 것은 쉽다. 일단 한번 정해지면, 서보는 새로운 조작이 수신될 때까지 원래의 동작을 지속할 것이다. 그래서 서보가 어떤 시간 길이로 실행하기 위해서는 여러분이 각각의 조작 후에 지연시간을 삽입해야만 한다.

예제 스케치: ServoRunTimes

- ServoRunTimes를 입력, 저장하고 아두이노로 업로드 하시오.
- 3초 동안 서보가 시계방향으로 회전하고 그 다음 반시계방향으로 3초 동안 회전 후 멈추는지를 점검하시오.

```
/*
 Robotics with the BOE Shield - ServoRunTimes
 Generate a servo full speed counterclockwise signal with pin 13 and
 full speed clockwise signal with pin 12.
 */
```

```
#include <Servo.h>                          // Include servo library

Servo servoLeft;                            // Declare left servo signal
Servo servoRight;                           // Declare right servo signal

void setup()                                // Built in initialization block
{
  servoLeft.attach(13);                     // Attach left signal to pin 13
  servoRight.attach(12);                    // Attach right signal to pin 12

  servoLeft.writeMicroseconds(1300);        // Pin 13 clockwise
  servoRight.writeMicroseconds(1300);       // Pin 12 clockwise
  delay(3000);                              // ..for 3 seconds
  servoLeft.writeMicroseconds(1700);        // Pin 13 counterclockwise
  servoRight.writeMicroseconds(1700);       // Pin 12 counterclockwise
  delay(3000);                              // ..for 3 seconds
  servoLeft.writeMicroseconds(1500);        // Pin 13 stay still
  servoRight.writeMicroseconds(1500);       // Pin 12 stay still
}

void loop()                                 // Main loop auto-repeats
{                                           // Empty, nothing needs repeating
}
```

제2장 요약

이 장의 요지는 서보의 계측과 점검이고, 서보 신호를 모니터하기 위한 지시등을 설치하는 것이다. 몇몇 하드웨어 제작에 덧붙여 전자 프로그램 그리고 몇 개의 좋은 엔지니어링 개념과 관계된 것들이 소개되었다.

하드웨어 제작

- 아두이노 모듈에 BOE 쉴드를 어떻게 설치하는가?
- 장치에 외부 배터리 전원을 어떻게 왜 공급하는가?
- BOE쉴드에 서보를 어떻게 연결하는가?

전자 공학

- 저항이 무엇인지, 심볼은 무엇인지, 칼라 색띠 값을 어떻게 바꾸는지에 대한 것
- 저항의 오차는 무엇인지
- LED는 무엇인지 그리고 단일방향 스위치처럼 동작하는 심볼은 무엇인지 그리고 애노드와 캐소드를 구분하는 방법에 대한 것
- 왜 브래드보드는 납땜이 필요 없는가? 그리고 전자소자를 서로 어떻게 연결하는가?
- LED지시등 회로를 어떻게 설치하는가 ?
- BOE쉴드에 서보 모터 부품을 어떻게 연결하는가 ?
- 전위차계가 무엇인가 그리고 전위차계를 조정하여 패럴렉스 연속회전 서보를 어떻게 계측하는가 ?
- pulse train이 무엇인가 그리고 펄스폭 변조로 서보를 어떻게 제어하는가?

프로그래밍

- HIGH와 LOW 출력 신호를 전송하기 위하여 아두이노 pinMode와 digitalWrite 함수를 어떻게 사용하는가?
- 서보 라이브러리를 스케치에 더하기 위하여 #include를 어떻게 사용하는가?

- 서보 라이브러리 함수가 서보를 제어하기
- 서보 속도 방향 그리고 실행 시간을 제어하기

엔지니어링

- 보드에서 회로 설치하기
- 소자 사이의 통신을 모니터하기 위하여 지시등 사용하기
- subsystem 검사가 무엇인지, 그리고 왜 중요한가?
- 입력 변수 표를 만들어 보고, 그것들이 장치 출력에 어떤 영향을 끼치는가?

제2장 도전

질문(Questions)

11.브레드보드를 사용하여 두 개 단자를 어떻게 연결하는가?
12.어떤 함수가 디지털 핀 방향을 설정합니까?
13.어떤 함수가 핀13번에 5V를 설정합니까? 어떤 함수가 핀13번을 0V로 설정합니까?
14.스케치가 5V 신호구간을 어떻게 제어할 수 있습니까?
15.연속회전 서보를 회전하도록 하는 펄스 구간이 무엇입니까?

16.전속 시계방향으로,
17.전속 반시계방향으로,
18.여전히 멈추어 있는.

19.어느 호출이 서보를 더 빠르게 회전하도록 합니까?

20.servoLeft.writeMicroseconds(1440) 또는
21.servoLeft.writeMicroseconds(1420). 왜?

22.어떤 서보신호 구간을 스케치가 어떻게 제어할 수 있습니까?

예제(Exercises)

23. 비작동시간의 1/3시점에서 1초마다 LED를 5번 깜박이도록 하는 반복 함수를 작성하시오. (이번 연습을 위하여 서보들을 분리하시오.)
24. 1.2초동안 13번 핀 서보는 전속 시계방향 회전하고 12번 핀은 멈춰있

도록 하는 setup 함수를 작성하시오. 그다음 두 개의 서보가 멈추도록 설정하시오.

25. 3초동안 하나의 서보를 회전시키는 setup함수를 작성하시오. 다른 서보는 처음 1.5초동안 반대 방향으로 회전해야 하고, 1.5초동안 동일방향으로 회전해야 합니다. 그다음 두 개의 서보가 멈추도록 합니다.

프로젝트(Projects)

26. 서보 라이브러리 detach 함수를 살펴보고 그것을 servoLeft의 위치에서 사용하시오. 그리고 3초동안 회전한 후에 서보를 멈추기 위하여 servoRight.writeMicroseconds(1500)를 사용하시오.
27. 12번 핀 서보가 시계방향으로 회전하는 동안 13번 핀 서보를 반시계방향으로 회전하게 하는 프로그램을 작성하시오. 3초후에 두 개의 서보를 0.6초동안 반시계방향으로 회전하게 하시오. 그다음 두 개의 서보를 0.6초동안 시계방향으로 회전시키시오. 이후 13번 핀 서보를 시계방향으로 회전시키고 12번 핀 서보를 반시계방향으로 3초동안 회전시키시오.

제2장 해답

질문 해답(Question Solutions)

1. 브레드보드 동일 행에 두 개 단자를 꽂으시오.
2. pinMode 함수
3. digitalWrite 함수가 value 변수에 따라 두 개 모두 설정합니다.

```
digitalWrite(13, HIGH) // 5 V
digitalWrite(13, LOW)  // 0 V
```

4. 어떤 핀이 HIGH로 설정된다면, 지연 호출이 어떤 시간동안 HIGH로 유지할 수 있다. 그 다음 digitalWrite호출이 그것을 LOW로 설정할 수 있다.
5. (a)전속 시계방향 회전동안 1.3 ms 펄스, (b)전속 시계방향 동안 1.7 ms 펄스, 그리고 (c)멈춘 상태동안 1.5 ms 펄스.
6. (b) servoLeft.writeMicroseconds(1420). 전속 시계방향은 servoLeft.writeMicroseconds(1300), 그리고 멈춤은 servoLeft.WriteMicroseconds(1500). 1420이 정지에서 더 나아가고 전속회전에 더 가깝기 때문에, 그 값은 비록 더 작은 값이라고 하더라도 더 빠른 시계방향 회전을 위하여 옳은 값이다.
7. Servo.writeMicroseconds(*newValue*)에 뒤이은 delay(*ms*)에 뒤이은 Servo.writeMicroseconds(*value*) 또는 or Servo.detach(*pin*)가 서보를 ms밀리세컨드 동안 회전하게 할 것이다.

연습의 해답(Exercise Solutions)

1. 전체 + off time이 200 ms이어야 하고, 이것은 1초의 1/5이다. 그래서 50ms 동안 동작하고, 150ms동안 동작하지 않는다.

```
void loop()                     // Main loop auto-repeats
{                               // 200 ms -> 5 blinks/second
  digitalWrite(13, HIGH);       // Pin 13 = 5 V, LED emits light
  delay(50);                    // ..for 0.05 seconds
  digitalWrite(13, LOW);        // Pin 13 = 0 V, LED no light
  delay(150);                   // ..for 0.15 seconds
}
```

2. 13번 핀 서보를 전속 시계방향으로 놓고, 핀12번 서보를 정지상태에 놓으시오. 그 다음 1200동안 기다리시오. servoRight가 벌써 멈추었기 때문에 모든 코드가 해야할 것은 servoLeft를 멈추는 것이다.

```
void setup()                      // Built in initialization block
{
  servoLeft.attach(13);          // Attach left signal to pin 13
  servoRight.attach(12);         // Attach right signal to pin 12

  servoLeft.writeMicroseconds(1300);  // 1.3 ms -> clockwise
  servoRight.writeMicroseconds(1500); // 1.5 ms -> stop
  delay(1200);                        // ..for 1.2 seconds
  servoLeft.writeMicroseconds(1500);  // 1.5 ms -> stop
}
```

3. 예제에서, 핌 13 서보는 반시계방향으로 시작하고 핀 12 서보는 시계방향으로 시작한다. 이과정은 1.5초동안 진행한다. 그 다음 핀 12 서보는 반시계방향으로 바뀌고, 또 다른 1.5초동안 계속한다. 그 다음 두 개의 서보가 멈춘다.

```
void setup()                      // Built in initialization block
{
  servoLeft.attach(13);          // Attach left signal to pin 13
  servoRight.attach(12);         // Attach right signal to pin 12

  servoLeft.writeMicroseconds(1700);  // 1.7 ms -> cc-wise
  servoRight.writeMicroseconds(1300); // 1.3 ms -> clockwise
  delay(1500);                        // ..for 1.5 seconds
  servoRight.writeMicroseconds(1700); // 1.7 ms -> cc-wise
  delay(1500);
  servoLeft.writeMicroseconds(1500);  // 1.5 ms -> stop
  servoRight.writeMicroseconds(1500); // 1.5 ms -> stop
}
```

프로젝트 해답

1. detach함수는 핀으로부터 서보의 인스턴스를 분리한다. 다음 스케치는 3초 동작한 후 서보를 멈추도록 한다는 것을 확인해 준다. 이 장 예제는 서보에게 여전히 기다리라고 말하는 펄스를 전송한다. 반대로, detach는 서보에게 신호 전송을 멈춘다. - 핀은 서보에게 어떤 것을 하라고 말하지 않는다. 그래서 "속도 멈춤"을 유지하는것 대신에 휴면 상태가 된다. 그 결과는 같고 서보 모터는 멈춘다. detach의 장점은 서보가 정확하게 계측되지 않더라도 서보가 천천히 돌지 못하도록 한다.

```
/*
 Robotics with the BOE Shield - Chapter 2, Project 1
 Generate a servo full speed counterclockwise signal with pin 13 and
 full speed clockwise signal with pin 12.
 */

#include <Servo.h>                      // Include servo library

Servo servoLeft;                        // Declare left servo signal
Servo servoRight;                       // Declare right servo signal

void setup()                            // Built in initialization block
{
  servoLeft.attach(13);                 // Attach left signal to pin 13
  servoRight.attach(12);                // Attach right signal to pin 12

  servoLeft.writeMicroseconds(1700);    // Pin 13 counterclockwise
  servoRight.writeMicroseconds(1300);   // Pin 12 clockwise
  delay(3000);                          // ..for 3 seconds
  servoLeft.detach();                   // Stop servo signal to pin 13
  servoRight.detach();                  // Stop servo signal to pin 12
}

void loop()                             // Main loop auto-repeats
{                                       // Empty, nothing needs repeating
}
```

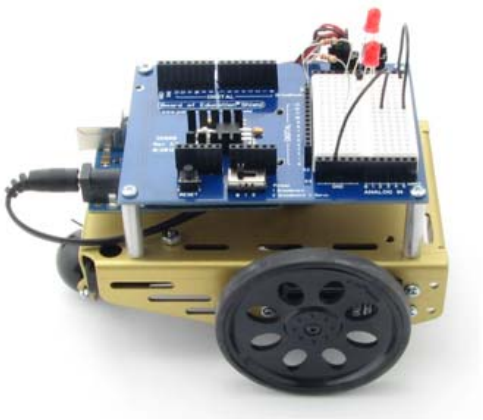
2. 해답이 이 장 끝의 스케치 ForwardLeftRightBackward이다.

제3장 ABOT 조립하고 검사하기

이 장은 ABot을 만들고 테스트하는 과정을 안내한다. 이 장은 특히 다음 장으로 넘어가기 전에 테스트를 완성하기 위해서 매우 중요하다. 이 장을 실습함으로써 일반적으로 ABot에서 흔히 하는 실수를 줄일 수 있다.

ABot 조립과 테스트 과정 요약

1. ABot 조립하기
2. 적합하게 연결되었는지 확인하고 다시 시작하기
3. ABot 배터리가 방전되었을 때를 알 수 있게 하는 스피커를 연결하고 테스트
4. 서보의 속도를 제어하고 테스트하기 위해 시리얼 모니터 사용



- 아두이노 코드 다운로드 받기 http://learn.parallax.com/node/192
- 데스크 탑에 저장하고 스케치를 사용하기 전에 압축을 푼다.
- 다음 과정을 따라하시오.

실습1: ABOT 조립

이 실습에서는 ABot 조립을 단계별로 소개한다. 각 단계는 몇 개의부품을 이용하고 그림을 참고하여 부품들을 조립한다. 각 그림들과 순서들이 있으니 주의해서 잘 따라하자.

서보 모터의 도구와 부품들

모든 필요한 부품들은 대부분 집이나 학교에서 찾을 수 있다. 또한 근처의 공구상에서 구입할 수도 있다.

도구들

(1개) 드라이버 (필립스 #1 point 드라이버 1/8“(3.18mm)
(1개) 1/4 “ 렌치 (선택사항)
(1개) 바늘코 플라이어 (펜치) (선택 사항)

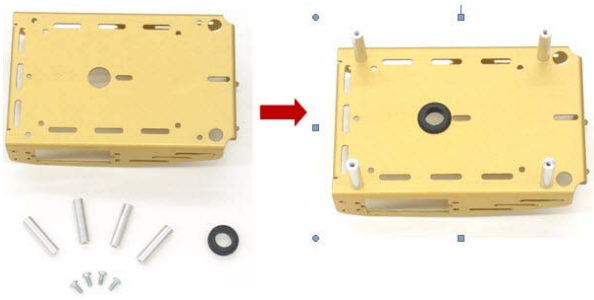
✓ 프로그래밍 케이블과 배터리 팩을 아두이노에서 분리한다.
✓ ABot에서 서보를 분리한다.
✓ ABot의 3지점 전원 스위치를 0으로 설정한다.

하드웨어 조립하기

부품 목록

(1개) ABot 차체
(4개) 1“ 지지대 (ABot에서 분리된 것)

(4개) 냄비 머리 작은 나사, 1/4″ 4-40
(1개) 고무 고리, 13/32“



지시 사항들

- ABot에서 1인치 알루미늄 지지대를 분리하고 나사를 보관한다.
- 13/32“ 고무링을 ABot의 차체 중앙에 끼워 준다.
- 차체 중앙에 테두리가 고무 고리로 되어 있는 구멍에 정확히 끼워 넣는다.
- 1/4″ 4-40 나사 네 개를 이용하여 그림처럼 네 개의 지지대를 붙인다.

서보 혼(Servo Horn) 분리하기

부품 목록

(2개) 패럴랙스 미리 중심을 맞추어 논 서보.

순서

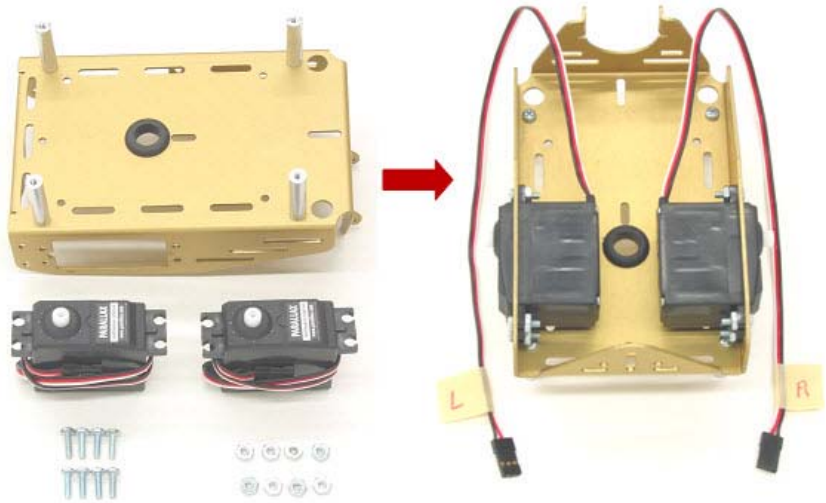
✓ 서보 날개위에 있는 서보 Control Horn에서 나사를 분리하기 위해 드라이버를 사용하라.
✓ 각각의 날개를 위쪽 방향으로 당겨서 빼낸다.
✓ 나사는 다시 사용하기 위해 보관한다.

주의 : 다음 단계로 가기 전에 다음 활동을 완성해야 한다. 2장의 실습 4: 서보 모터와 배터리 연결하기와 2장의 실습 5: 서보 센터 맞추기.

몸체에 서보 연결하기

부품 목록

(2개) ABot 몸체, 부분 조립된 것.
(2개) Parallax 연속 회전 서보
(8개) 냄비 머리 작은 나사, 3/8″ 4-40
(8개) 너트, 4-40
이름 테이프
펜



순서:

✓ 드라이버와 너트를 이용하여 서보를 몸체에 연결한다.
✓ 이름 테이프를 사용하여 위의 그림과 같이 서보의 왼쪽은(L) 오른쪽은 (R)을 적어서 붙인다.

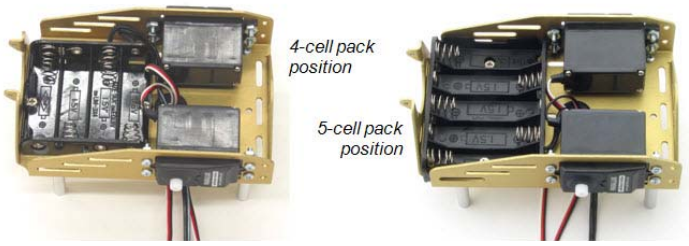
건전지 팩 조립하기

부품 목록

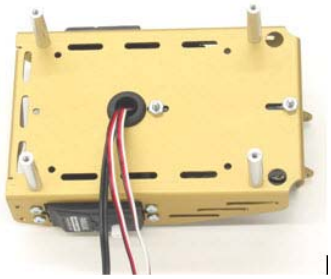
(2개) 접시 머리 작은 나사, 3/8″ 4-40
(2개) 너트, 4-40
(1개) 4개짜리 또는 2.1mm 가운데가 양극인 5개짜리 건전지 팩

순서

✓ 그림과 같이 4개짜리 또는 5개짜리 팩을 아래처럼 놓는다.



✓ 드라이건전지 팩이 차체를 통과해서 나사를 끼워 차체의 아래쪽에 빈 건전지 팩을 붙이고 스크류 위에 너트를 돌려서 꼭 맞춘다.
✓ 만약 Boe-Boost 모듈을 한 개 사용한다면 건전지를 다시 끼운다.
✓ 건전지 팩의 전원 선과 서보 선들을 차체의 중앙에 있는 고무로 된 구멍을 따라서 선을 그림과 같이 끼운다.



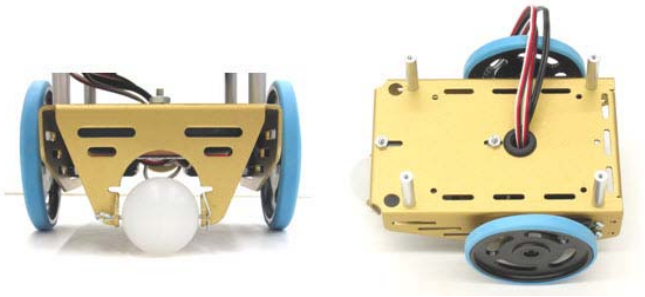
바퀴 조립하기

부품 목록

(1개) 1/16″ 이음 핀
(1개) 꼬리 바퀴 공 I
(2개) 고무 밴드 타이어
(2개) 플라스틱 휠
(2개) 서보 보터의 십자 뿔에서 제거한 나사

순서

ABot의 꼬리 바퀴 공은 가운데를 통과 할 수 있는 플라스틱 공이다. 이음 핀은 몸체에 고정시키고 기능은 바퀴의 축으로 사용된다.



✓ 차체의 끝 부분의 구멍과 꼬리 바퀴 공의 구멍의 위치를 일치시킨다.
✓ 고정 핀을 세 개의 구멍(차체 왼쪽, 꼬리 바퀴공, 차체 오른쪽)을 관통시켜 끼워 준다.
✓ 고정 핀의 마지막 부분을 공이 빠지지 않도록 구부려 줍니다.
✓ 바퀴용 고무 밴드를 잡고 늘려서 각 플라스틱 휠에 끼워 넣는다.
✓ 바퀴의 공간에 꼭 맞게 들어가서 몸체와 일직선이 되게 서보 날개에 있는 각 플라스틱 휠을 누른다. 그런 다음 서보 스크류를 보관한다.

ABot을 몸체에 조립하기

부품 목록

(4개) 접시 머리 작은 나사, 1/4″ 4-40

(1개) ABot 보드

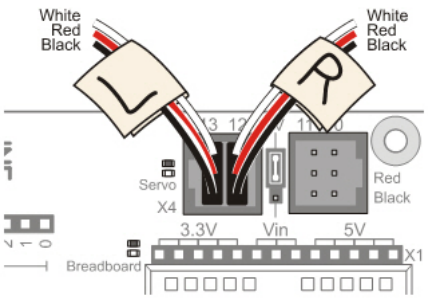
순서

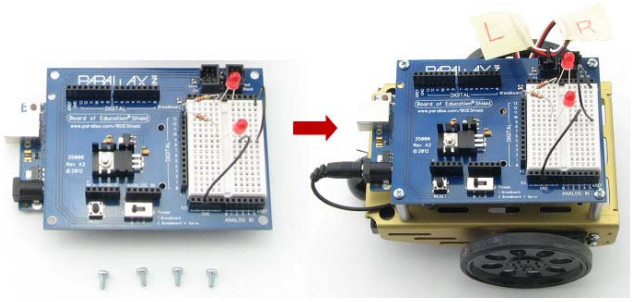
✓ 지지대 네 군데에 ABot을 놓고 보드의 바깥쪽 모서리에 네 개의 지지대 구멍에 세운다.

✓ 흰색 브레드보드가 바퀴에 더 가깝게 오게 한다. (꼬리 바퀴가 아님)

✓ 접시 머리 작은 나사로 지지대를 보드에 조립한다.

✓ 서보의 헤더를 서보에 다시 연결한다.

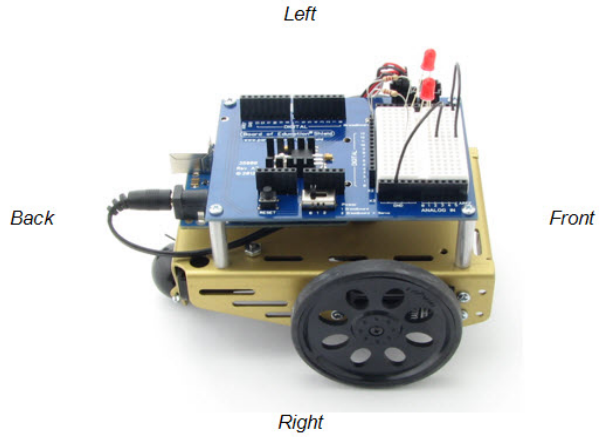




✓ 몸체의 아래에서 부터 서보와 전원의 초과된 케이블을 고무로 된 구멍을 통과하여 밀어 넣고, 서보와 몸체 사이의 케이블 길이를 맞춘다.

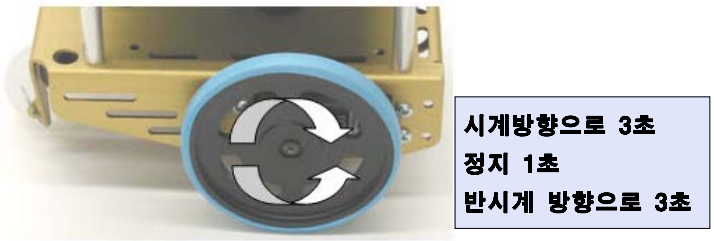
실습2: 서보 모터 다시 테스트하기

이 실습에서는 보드와 서보가 사이에 정확히 전기적으로 연결되어 있는지 테스트한다. 아래 그림은 ABot의 앞, 뒤, 왼쪽과 오른쪽을 보여준다. 핀 12번으로부터 신호를 받을 때 서보가 오른쪽으로 돌고 핀 13으로부터 신호를 받을 때 왼쪽으로 돌아야 한다.



오른쪽 바퀴와 왼쪽 바퀴 테스트하기

다음 예제는 오른쪽 바퀴가 서보에 그림과 같이 정확이 연결되었는지 테스트 한다. 이 스케치는 3초간 시계방향으로 바퀴가 회전하고 그다음 정지 한다. 그리고 1초간 멈춘 후 시계 반대 방향으로 3초간 회전한다.



예제 스케치: RightServoTest

✓ 드라이브 바퀴가 바닥위에 오도록 하고 ABot이 앞을 향하게 둔다.

✓ 프로그래밍 케이블과 전원 팩을 아두이노에 연결한다.
✓ 코드를 입력하고 저장한 다음 RightServoTest 를 아두이노에 업로드 시킨다.
✓ 3-지점 스위치를 2 위치에 놓고 RESET 버튼을 누르고 뗀다.
✓ 오른쪽 바퀴가 시계 방향으로 3초간 돌고 1초간 쉬었다가 다시 3초간 반시계 방향으로 회전하는지 확인한다.
✓ 만약 오른쪽 바퀴/서보가 예상한 대로 작동하지 않는다면 Servo Troubleshooting 페이지를 참고한다.
✓ 만약 오른쪽 바퀴/서보가 예상대로 작동하면 계속 진행한다.

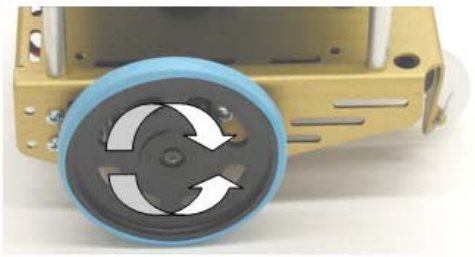
```
/*
 * ABot - RightServoTest
 * 오른쪽 서보가 시계 방향으로 3초간 회전하고, 1초간 정지한 다음
 * 3초간 반 시계 방향으로 회전한다.
 */
```

```
#include <Servo.h>                          // Include servo library
Servo servoRight;                           // Declare right servo

void setup()                                // Built in initialization block
{
  servoRight.attach(12);                    // Attach right signal to pin
12
  servoRight.writeMicroseconds(1300);       // Right wheel clockwise
  delay(3000);                              // ...for 3 seconds
  servoRight.writeMicroseconds(1500);       // Stay still
  delay(1000);                              // ...for 3 seconds
  servoRight.writeMicroseconds(1700);       // Right wheel
counterclockwise
  delay(3000);                              // ...for 3 seconds
  servoRight.writeMicroseconds(1500);       // Right wheel
counterclockwise
}
void loop()                                 // Main loop auto-repeats
{                                           // Empty, nothing needs
repeating
```

```
}
```

여러분의 차례- 왼쪽 바퀴 테스트하기

자 이제 아래 그림처럼 같은 방법으로 왼쪽 바퀴를 테스트 할 차례이다. 이것은 RightServoTest 스케치와 관련이 있다.



시계 방향으로 3초
정지 1초
반시계 방향으로 3초

- ✓ RightServoTest를 LeftServoTest로 저장한다.
- ✓ Servo servoRight 를 sServo servoLeft로 변경한다.
- ✓ servoRight.Attach(12)를 servoLeft.Attach(13)로 변경한다.
- ✓ 나머지 servoRight를 servoLeft.로 수정한다.
- ✓ 스케치를 저장하고 아두이노에 업로드 시킨다.
- ✓ 왼쪽 바퀴가 3초간 시계방향으로 돌고 1초간 쉬었다가 다시 시계 반대 방향으로 도는지 확인한다.
- ✓ 만약 왼쪽 바퀴/서보가 예상대로 작동하지 않는다면 Servo Troubleshooting를 보시오.
- ✓ 만약 왼쪽 바퀴/서보가 예상대로 움직인다면 ABot은 적당하게 작동하는 것이다. 이제 다음 장으로 넘어가시오.

✓ 서보 문제 해결

다음은 일반적인 증상과 어떻게 고치는지에 대한 목록이다. 첫 번째 단계는 여러분의 스케치를 이중으로 확인하고 코드가 정확한지 확인 하는 것이다.

- ✓ 여러분의 스케치를 RightServoTest 에 대해서 두 번 확인 한다.
- ✓ 만약 코드가 정확하다면 아래 목록에서 증상을 찾고 지시하는 순서를 따른다.

서보가 돌지 않는다.

- ✓ 서보가 이 다이어그램 가이드를 사용하고 있는지 이중으로 확인한다.(몸체에 정확히 연결하기)

✓ ABot 의 전원스위치가 2-지점에 있는지 확인한다. RESET 버튼을 눌렀다가 떼면 프로그램을 다시 실행시킬 수 있다.
✓ 스케치가 정확하게 입력되었는지 확인한다.

왼쪽 서보가 오른쪽 서보가 돌 때 돈다.

이 의미는 서보가 뒤 빠뀐 것이다. 서보는 12번 핀을 13번 핀에 번에 연결하여야 하고, 13번 핀을 12번 핀에 연결하여야 한다.

✓ 전원 선과 패터리 팩을 분리한다.
✓ 서보도 뺀다.
✓ 서보 핀 12번을 핀 13번에 연결시킨다.
✓ 서보 핀 13번을 12번 핀에 연결시킨다.
✓ 전원을 다시 연결시킨다.
✓ RightServoTest 스케치를 다시 실행시킨다.

바퀴가 완전히 멈추지 않고 여전히 천천히 움직인다.

만약 바퀴가 시계방향, 정지 ,반시계 방향 순서 후에 천천히 계속 돈다면 이것은 정확히 중심이 맞지 않는다는 의미이다. 다음에 해결방법이 두 가지 있다.

✓ 하드웨어 접근 : 2장으로 다시 돌아가서 서보의 중심을 맞춘다. 서보가 전위차계에 접근하기 쉽지 않다면 다시 조립하는 것을 고려한다.
✓ 소프트웨어 접근 : 만약 반시계 방향으로 천천히 돈다면 1500보다 좀 더 적은 값을 사용한다. 만약 시계 방향으로 천천히 돈다면 1500보다 더 큰 값을 사용해본다. 이 새 값은 writeMicroseconds 호출에서 1500을 수정한다.

바퀴가 멈추지 않고 계속 빨리 움직인다.

만약 여러분의 스케치가 정확하다면 아마도 서보가 중심이 맞지 않는다는 의미이다. 2장으로 돌아가서 서보의 중심을 맞춘다.

- 바퀴를 분리하고 서보를 몸체로부터 분리한 다음 2장으로 돌아가서 서보의 중심을 맞추는 연습을 반복한다. 다시 조립하는 것을 고려해야 한다.

실습3: 시작-리셋 지시자

이 장에서는 ABot 프로토타입 영역에서 작은 소리를 발생시키는 회로를 만들 것이다. 이것은 ABot의 건전지가 너무 적게 소모되면 소리를 발생시킬 것이다.

기기가 충분한 기능을 하기 위한 필요 전압이 떨어질 때 이것을 전압저하(Brownout)라고 한다. 기기(아두이노)가 정상전압으로 돌아올 때까지 일반적으로 꺼져버린다. 그러면 스케치가 실행중이더라도 재시작 된다.

전압저하는 건전지가 얼마 남지 않았을 때와 서보가 갑자기 더 많은 전원을 요구할 때 일반적으로 발생한다. 예를 들어 만약에 ABot이 최대 속도로 전진하다가 변환하여 최대 속도로 후진한다면 서보는 서보가 정지하고 다시 전진해야하는 추가적인 더 많은 작업을 해야 한다. 그래서 더 많은 전원이 필요하고 건전지로부터 최대 전원을 끌어 쓴다. 출력 전압은 전압강하를 발생시키는 원인이 된다.

자! 이제 여러분의 ABot이 일반적인 주행 중에 갑자기 멈추고 완벽히 기대되지 않은 방향으로 간다고 생각해보자. 코드가 실수거나 아니면 전압강하가 있다면 어떻게 할 것인가? 간단하고 충분한 해결방법은 ABot에 스피커를 추가하고 매 스케치가 실행될 때 마다 “시작” 신호를 실행시키는 것이다. 이 방법은 ABot이 주행 중에 전력강하가 된다면 시작 신호 때문에 여러분이 알아차릴 수 있을 것이다.

우리는 아두이노로부터 받는 high/low 신호의 주파수에 따라 서로 다른 소리를 만드는 피에조스피커 (*piezoelectric* 스피커)라고 하는 기기를 사용한다. 구조도와 부품은 아래에서 보이는 것과 같다.

주파수(Frequency)는 주어진 기간 동안 어떤 일이 얼마나 자주 발생하는지 측정한 것이다.

압전소자는 전압이 들어올 때 모양이 변하는 수정체가 있다. high/low 전압이 작용해서 압전성 수정체 소자가 재빨리 모양을 바꾸고 그 결과 진동이 일어난다. 이 때 진동되는 물체 주변 공기도 함께 진동하는데 이것을 우리가 들을 수 있는 소리와 음으로 나타나는 것이다. 모든 진동하는 물체는 서로 다른 음을 만든다.

압전소자는 다양한 용도가 있다. 압전소자에 힘이 적용될 때 전압을 생성 할 수 있다. 어떤 압전소자는 고유의 진동수가 있는데 이것을 통해서 만드는 전압으로 많은 컴퓨터와 마이크로컨트롤러를 위한 시계 발진기로의 기능을 할 수 있다.

피에조 스피커 회로 구성

필요한 부품들

(1개) 피에조 스피커(만약 한 개가 있다면 “Remove seal after washing"스티커를 벗겨낸다.

(기타) 연결 선들

출발/리셋(Start/Reset) 지시 회로 만들기

아래 그림은 피에조스피커를 브레드보드에 추가하는 배선도이다.

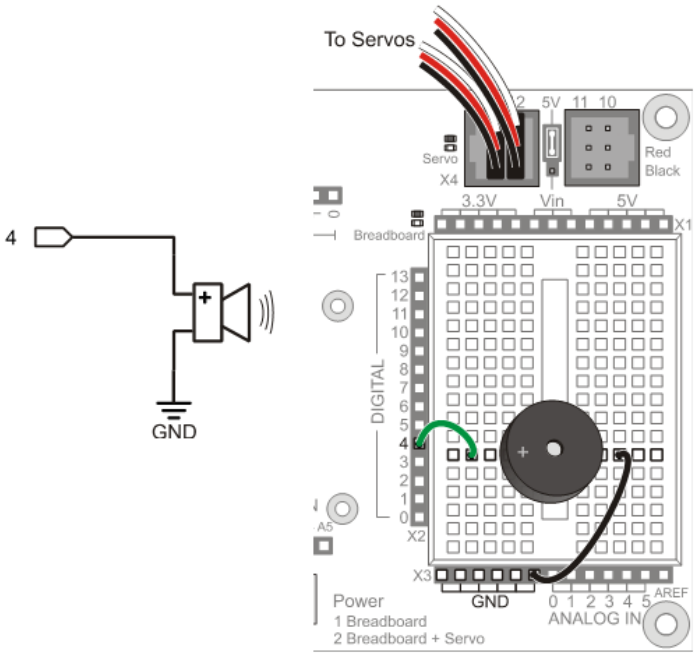
주의

회로를 조립하거나 수정하기 전에 항상 전원선을 분리하라.!

스위치를 0에 맞춘다.
건전지 팩을 분리시킨다.

프로그래밍 케이블을 분리시킨다.

✓ 그림과 같이 회로를 조립한다. 브레드보드위에서 그림과 같이 만든다. 스피커는 책의 나머지 부분에서 계속 남아있고 반면에 다른 회로는 계속 추가되거나 제거된다. 다른 회로에 추가되거나 제거되는 동안에 책의 나머지 자리에 남아 있다.


4
GND
To Servos
Servo
X4
Red
Black
3.3V
Vin
5V
Breadboard
X1
DIGITAL
13
12
11
10
9
8
7
6
5
4
3
2
1
0
A5
X2
X3
GND
AREF
0 1 2 3 4 5
ANALOG IN
Power
1 Breadboard
2 Breadboard + Servo

시작과 리셋 표시기 프로그래밍

다음 예제는 피에조스피커를 사용하여 아두이노의 tone() 함수를 호출하는 스케치 테스트이다. 함수의 이름에 대한 사실은 이 함수는 tone 신호를 스피커에 보낸다.

tone() 함수를 호출하기 위해 두 가지 선택사항이 있다. 하나는 tone의 pin 과 *frequency 를 지정해 주는 것이다. 다른 하나는 pin*, *frequency*, 그리고 *duration*(milliseconds)을 지정해주는 것이다. 우리는 tone을 구별 할 필요가 없기 때문에 두 번째 선택사항을 사용한다.

tone(*pin* , *frequency*)
tone(*pin*, *frequency*, *duration*)

이 피에조스피커는 4.5kHz 로 실행되게 설계되어 있다. 그러나 들을 수 있는 다양한 소리도 낼 수 있다. 그리고 대개 1kHz ~ 3.5kHz 범위에서 일반적으로 가장 좋은 소리이다. 시작-알림 음은 아래와 같이 사용할 것이다.

```
tone(4, 3000, 1000);
delay(1000);
```

이것은 핀 4번이 high/low 신호를 3kHz(초당 3000번)로 반복해서 보내는 것을 말한다. tone(소리)는 1000ms 동안 즉 1초 동안 지속된다. **tone** 함수는 스케치가 다음 명령으로 이동할 때 까지 배경으로 지속된다. 우리는 tone 이 실행될 때는 서보가 시작하지 않게 하고 싶다. 그래서 스케치가 서보를 이동시키기 전에 tone이 정지되도록 **tone 명령뒤에 delay(1000)**가 따라 나온다.

주파수 단위는 헤르츠(Hz)로 측정할 수 있다. 헤르츠는 1초 동안 자신이 신호를 몇 차례 반복하는지 나타낸다. 사람의 귀는 매우 낮은 20 Hz에서부터 매우 높은 (20kHz 또는 20,000 Hz)주파수를 감지한다. 1 킬로 헤르쯔는 1초당 1000을 간략하게 표시한 것이다.

예제 스케치:StartResetIndicator

이 예제 스케치는 프로그램이 시작될 때 삐 소리를 내게 하며 0.5초마다 시리얼 모니터를 이용해 메시지를 출력하는 프로그램이다. 이 메시지들은 loop() 함수 안에 있기 때문에 명령어들이 계속 반복된다. 아두이노의 전력에 인터럽트가 발생하면 스케치는 처음부터 다시 시작되므로, 삐 소리가 계속 나면 배터리 수명이 다 되었음을 알 수 있다.

- 보드에 전원을 다시 연결하세요.
- StartResetIndicator 코드를 작성하고 저장하고 아두이노에 업로드하세요.
- 만약 삐 소기라 들리지 않으면, 작성한 코드에 오류가 없는지 확인하고 다시 실행하자.
- 만약 삐 소기가 들리면, 시리얼 모니터를 열자(이것 역시 리셋을 함). 그 다음, ABot의 푸시 버튼을 누르자.
- 리셋 할 때 마다 1초 동안 명확히 들리는 삐 소리를 들을 수 있는 재확인하고 “Waiting for reset…” 메시지가 나오는지 보자.
- 또한 배터리 선과 프로그래밍 케이블을 빼냈다가 연결해보자. 이렇게 할 때도 시작-알림 삐 소리가 나야 한다.

```
/*
 * Robotics with theABot - StartResetIndicator
 * 피에조스피커 회로 테스트
 */
```

```
void setup()                                    // Built in initialization
block
{
  Serial.begin(9600);
  Serial.println("Beep!!!");

  tone(4, 3000, 1000);                          // Play tone for 1 second
  delay(1000);                                  // Delay to finish tone
}

void loop()                                     // Main loop auto-repeats
```

```
{
  Serial.println("Waiting for reset...");
  delay(1000);
}
```

StartResetIndicator가 어떻게 작동될까?

StartResetIndicator 는 “Beep!!!” 메시지를 시리얼 모니터에 출력하면서 시작된다. 메시지를 출력 다음 **tone** 함수는 피에조스피커로 3kHz를 1초간 울린다. 아두이노에 의해서 명령들이 매우 빠르게 실행되므로 메시지가 출력되는 것과 동시에 피에조스피커가 톤의 연주를 시작하는 것처럼 보인다.

톤이 마치면, 스케치는 loop 함수에 들어간다. 이것은 “Waiting for reset…” 메시지를 끝없이 반복한다. ABot 리셋 버튼을 누를때마다 또는 전원을 끊었다가 다시 연결 할때마다, 스케치는 “Beep!!!” 메시지와 3kHz 톤을 다시 내기 시작한다.

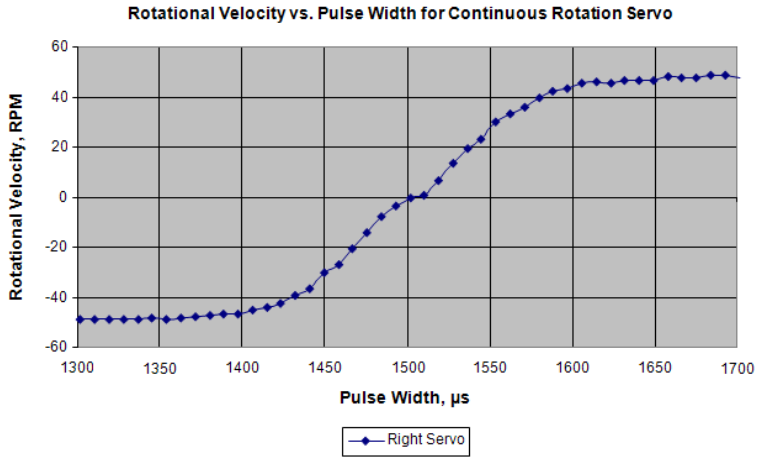
여러분의 차례–StartResetIndicator를 다른 스케치에 추가 하자.

우리는 지금부터 나오는 모든 예제 스케치에서 톤을 사용할 것이다. setup()함수의 시작 부분에 삐 소리와 잠시 멈추는 문장을 모두 추가하는 것은 좋은 아이디어이며 좋은 습관을 갖는 것이다.

- ✓ RightServoTest를 열자.
- ✓ StartResetIndicator 스케치로부터 tone()과 delay() 함수 호출을 복사하고 RightServoTest 스케치의 setup() 함수에 시작 부분에 붙여 넣자.
- ✓ 수정된 스케치를 실행하고 “sketch starting” 이라고 반응하며 아두이노를 리셋 할 때 마다 삐 소리를 내는지 살펴보자.

실습4: 속도 제어 시험

이 그래프는 서보의 속도에 대한 펄스 시간을 보여 준다. 그래프의 수평 축은 펄스의 폭을 마이크로초(µs)나타내고, 수직 축은 서보의 RPM 반응을 나타낸다. 시계방향의 회선은 음의 값으로 나타나고 반시계 방향은 양의 값이다. 이 특별한 서보 그래프는 펄스폭 1300~1700 µs의 범위에 대해서 -48에서 +48 RPM 법위를 나타내는 전달 곡선이라고도 부른다. 여러분들 서보의 전달 곡선 그래프도 이와 유사할 것 같다.



전송 커브 그래프가 유용한 세 가지 이유들

28. 여러분들은 특정한 펄스폭을 위한 여러분들의 서보로부터 무엇을 기대할 수 있을지에 대한 아이디어를 얻을 수 있다. 1500에서 그래프에서 수직으로 가로지르는 지점을 찾기 위해 수직선을 따라 가보면 회전이 0 인 지점이 바로 1500 µs 펄스임을 알 수 있다.

- 1300과 1350 µs 펄스에서 서보의 속도를 비교해 보면 별 차이가 없지 않는가?
- 1550 µs 펄스에서 서보의 기대되는 속도는 얼마나 되고 1525 µs 펄스에서는 얼마나 될까?

29. 1300과 1400 µs 펄스 사이에는 속도의 차이가 별로 크기 않다. 그래

서 1300 µs 펄스를 시계방향의 최대 속도가 난다는 것은 과도한 것이 아닐까? 동일하게 적용하면 반 시계 방향의 회전에서 1600와 1700 µs 펄스가 동일한 결과가 된다. 이와 같은 과도한 속도 설정은 1400 에서 1600 µs 범위에서 두 수를 선택하는 것으로 보다는 밀접하게 가까운 속도 내에서 같은 결과가 나타나므로 매우 유용하다.

30. 1400과 1600 µs 사이는, 속도 조절이 더 많거나 더 적은 선형이다. 이 범위에서, 특정한 펄스폭의 변경은 속도의 변화에 대응하는 결과를 가져온다. 이 범위의 펄스를 사용하여 여러분의 서보 속도를 제어하자.

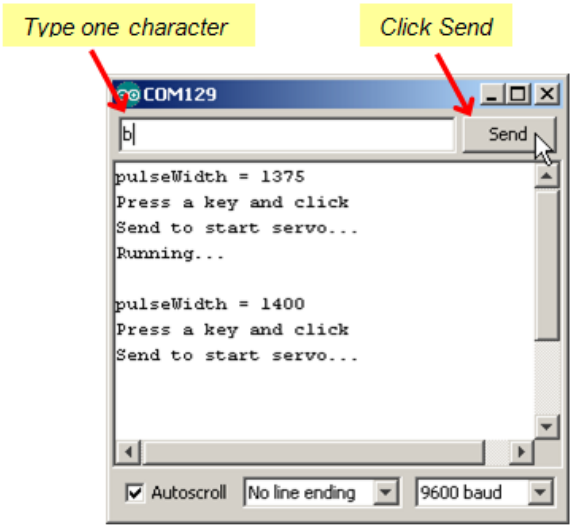
예제 스케치: 서보 속도 테스트

이 스케치에서 펄스의 값을 1375 µs 에서 1625 µs 까지 25 µs 단계로 증가 할 때 서보의 RPM 속도와 방향을 체크할 수 있다.

이 속도의 측정은 1400 – 1600 µs 범위 에서 서보가 펄스 길이를 명확히 제어하는 것을 돕는다. 이 스케치는 서보 제어 신호를 보낼 준비가 되어있다는 뜻의 펄스 기간을 출력하는 것으로부터 시작한다. 그런 다음 시리얼 모니터에서 서보를 돌리기 전에 아두이노로 문자를 보낼 때 까지 기다린다.

이것은 서보를 6초간 돌리고, 여러분은 바퀴가 몇 바퀴 도는지 셀 수가 있다. 그 다음에 for 반복은 자신이 반복을 하고 펄스폭을 다음 테스트를 위해서 25씩 증가한다.

- 바퀴에 마스킹 테이프 조각 같은 위치를 표시 해두므로 바퀴의 속도 측정을 하는 동안 바퀴가 몇 회 회전하는지 볼 수 있다.
- ABot을 코 쪽으로 세워 두어서 바퀴가 마음껏 회전할 수 있도록 한다.
- TestServoSpeed를 입력하고, 저장한 다음 아두이노에 업로드 한다.
- 시리얼 모니터를 열고, 드롭다운 메뉴에서 “No line ending” 과 “9600 baud.”로 설정을 하고 전송화면을 클릭하자.
- 창의 윗부분을 클릭하고, 어느 문자든 입력한 후 [Send] 버튼을 누른다.
- 바퀴가 몇 회 회전하는지 세어 보고 RPM을 계산하기 위해 10을 곱하라.(방향을 적어 두는 것을 잊지 말자; 5회 테스트를 하고 바꾸자)
- 만약 여러분들의 데이터를 그래프에 추가했다면, 전반적인 모양이 적합해 보이는가?



/*

ABot 로봇 - TestServoSpeed

시리얼 모니터로부터 문자 하나를 아두이노로 보내면 이것이 왼쪽 서보를 6초간 회전하게 한다. 1375us에서 시작하여 25씩 증가하면서 반복되고 1625us에 마친다. 이 스케치는 속도에 대한 펄스 폭을 그래프로 나타내는데 매우 유용하다.

*/

```
#include <Servo.h>                      // Include servo library
Servo servoLeft;                        // Declare left servo signal
Servo servoRight;                       // Declare right servo signal

void setup()                            // Built in initialization block
{
  tone(4, 3000, 1000);                  // Play tone for 1 second
  delay(1000);                          // Delay to finish tone

  Serial.begin(9600);                   // Set data rate to 9600 bps
  servoLeft.attach(13);                 // Attach left signal to P13
}
void loop()                             // Main loop auto-repeats
```

```
{

  // Loop counts with pulseWidth from 1375 to 1625 in increments of 25.

  for(int pulseWidth = 1375; pulseWidth <= 1625; pulseWidth += 25)
  {
    Serial.print("pulseWidth = ");              // Display pulseWidth value
    Serial.println(pulseWidth);
    Serial.println("Press a key and click"); // User prompt
    Serial.println("Send to start servo...");

    while(Serial.available() == 0);            // Wait for character
    Serial.read();                              // Clear character

    Serial.println("Running...");
    servoLeft.writeMicroseconds(pulseWidth); // Pin 13 servo speed = pulse
    delay(6000);                                // ..for 6 seconds
    servoLeft.writeMicroseconds(1500);        // Pin 13 servo speed = stop
  }
}
```

TestServoSpeed 작동원리

TestServoSpeed 스케치는 for() 반복을 통해서 매번 pulseWidth라는 이름의 변수 값을 25씩 증가시킨다.

```
  // 반복의 수는 pulseWidth가 1375부터 1625사이를 25씩 증가함.

  for(int pulseWidth = 1375; pulseWidth <= 1625; pulseWidth += 25)
```

for() 반복은 각각의 반복과 함께 다음 펄스폭을 사용자 프롬프트와 함께 13번 핀 서보로 보낸다.

```
    Serial.print("pulseWidth = ");              // Display pulseWidth value
    Serial.println(pulseWidth);
```

```
Serial.println("Press a key and click"); // User prompt
Serial.println("Send to start servo...");
```

setup() 함수에서 **Serial.begin()을** 한 후에, 아두이노가 시리얼 모니터로부터 들어오는 문자를 위해서 메모리를 조금 설정해 둔다. 이 메모리를 일반적으로 시리얼 버퍼()라고 부르고 시리얼 모니터로부터의 ASCII 값을 저장한다. 매번 **Serial.read()를** 사용하여 버퍼로부터 문자를 가져오고 아두이노는 대기 버퍼의 문자수를 하나 줄인다. **Serial.available()을** 호출하는 것은 얼마나 많은 문자들이 버퍼에 있는지 말해 준다. 이 스케치는 시리얼 모니터가 문자를 보낼 때 까지 while(Serial.available() == 0)을 사용하여 기다린다. 서보가 달리기 위해 이동하기 전에, Seiral.read()를 사용하여 버퍼로 부터의 문자를 지운다. 스케치는 int myVar = Serial.read( ) 사용하여 변수에 문자를 복사한다. 코드가 문자 값으로 판단을 하지 않기 때문에, 단지 Serial.read()을 호출기만 하지 결과를 아무 곳에도 저장하지는 않는다.

가장 중요한 부분은 버퍼를 깨끗하게 할 필요가 있으며 그렇게 하므로 Seral.available( ) 이 다음번 반복에서 0을 리턴 하도록 한다.

```
while(Serial.available() == 0);        // Wait for character
Serial.read();                          // Clear character
```

while() 반복 코드 블록은 도대체 어디에 있을까?

C언어에서는 while() 반복이 빈 블록 코드를 사용하는 것을 허용한다. 이 경우 문자가 도착 할 때 까지 거기서 기다린다. 여러분이 시리얼 모니터에 문자를 입력할 때 Serial.available()은 0 대신 1을 리턴 하므로 while 반복은 스케치를 다음 문장으로 이동하는 것을 허락 한다. Serial.read()는 아두이노의 시리얼 버퍼로부터 입력된 문자를 지운는데 Serial.available()가 다음 번 반복에서 0을 리턴 할 것을 확실하게 해 준다. 여러분들은 while 반복을 서로 다른 방법으로 적을 수 있다.

```
  while(Serial.available() == 0) {}
...or:
   while(Serial.available() == 0) {}; .
```

아두이노가 키보드로부터 문자를 받은 다음 “Running…”메시지를 출력하고 나서 서보를 6초간 회전시킨다. 이 코드에서 for() 반복은 pulseWidth 변수 값이 1375로 시작하여 각 반복마다 25씩 증가 된다. 그래서 첫 반복에서 servoLeft는 1375이며, 다음번에는 1400, 그리고 1625까지 동일 한 방법으로 증가 된다. 매번 반복에서 **servoLeft.writeMicroseconds(pulseWidth)**를 사용하여 **pulseWidth** 값을 서보 속도로 저장을 한다. 이것은 시리얼 모니터를 통해서 아두이노로 문자를 보내면 서보의 속도를 매번 마다 갱신한다.

```
Serial.println("Running...");
servoLeft.writeMicroseconds(pulseWidth); // Pin 13 speed=pulse
delay(6000);                             // ..for 6 seconds
servoLeft.writeMicroseconds(1500);       // Pin 13 speed=stop
```

옵션: 여러분의 전달 커브 데이터를 기록하자.

자신의 전달 곡선 데이터를 기록하기위해 아래 표를 사용할 수 있다. TestServoSpeed 스케치의 반복은 테이블에 있는 모든 값들을 변경 시킬 수 있거나 모두 동일하게 할 수도 있다.

- 아래 부분 오른쪽에 보이는 "Printer-friendly version" 링크를 클릭하자.
- TestServoSpeed에서 for() 문장을 다음처럼 변경:

for(int pulseWidth=1375; pulseWidth <= 1625; pulseWidth += 25)

…을:

for(int pulseWidth=1300; pulseWidth <= 1700; pulseWidth += 20)

- 수정된 스케치를 아두이노에 가져오고, 이것을 이용하여 테이블의 내용을 모두 채우자(테이블을 모두 채우려면 for문의 증가 파라미터를 pulseWidth += 10로 변경)
- 마음에 드는 그래프를 나타내는 소프트웨어를 선택하고 펄스폭과 바퀴의 RPM을 그려보자
- 오른 바퀴를 위해서 측정을 반복하고, 스케치에서 모든 13 인스턴스를 12로 바꾸라.

Table 3-1: Pulse Width and RPM for Parallax Continuous Rotation Servo

Pulse Width (µs)	Rotational Velocity (RPM)	Pulse Width (µs)	Rotational Velocity (RPM)	Pulse Width (µs)	Rotational Velocity (RPM)	Pulse Width (µs)	Rotational Velocity (RPM)
1300		1400		1500		1600	
1310		1410		1510		1610	
1320		1420		1520		1620	
1330		1430		1530		1630	
1340		1440		1540		1640	
1350		1450		1550		1650	
1360		1460		1560		1660	
1370		1470		1570		1670	
1380		1480		1580		1680	
1390		1490		1590		1690	
						1700	

제3장 요약

이 장에서는 ABot을 조립하고 테스트하였다. 조립은 기계적인 만들기와 회로의 구성 및 테스트와 테스트에 사용되는 프로그래밍 개념을 다루었다. 몇 가지 요점 정리를 해 보자.

하드웨어 설치

- 로봇 몸체에 서보 장착하기
- 서보 모터에 바퀴 끼워 넣기와 몸체에 꼬리 바퀴 달기
- ABot 장착하기

전자

- 피에조스피커란 무엇이며, ABot의 브레드보드 회로에 어떻게 추가할 것인가.
- 주파수가 무엇이며, 측정 단위와 사람이 들을 수 있는 주파수 범위는 어떻게 되는가.
- 배터리가 낮아 작동되지 않는 조건과 서보 모터의 회전에 어떤 영향을 미치는가?

프로그래밍

- 아두이노의 tone() 함수로 어떻게 피에조스피커로 삐 소리를 내는가?
- 시리얼 버퍼란 무엇이며, 아두이노 Serial.available()함수가 어떻게 사용하는가?
- 시리얼 모니터에서 전송부분을 어떻게 이용하며, Serial.read()함수에 대해, 아두이노로 데이터를 어떻게 보내는가?
- 서보의 RPM을 변경시기기 위해서 변수 증가를 for() 반복에서 어떻게 활용하는가?
- tone()함수를 프로그램 작업완료 하였다는 것을 알리는 방법으로 어떻게 스케치에 추가하는가?

엔지니어링 기술

- 로봇이 예상치 못한 행동을 하면 배터리가 소모된 것이지 프로그래밍

오류가 아님을 결정하는 brownout indicator strategy 방법

- 전송 커브 그래프가 무엇이며, 서보 제어 신호에 대한 서보 RPM 관계를 이해하는데 활용되는지.

제3장 도전

질문(Questions)

1. ABot의 brownout 현상은 어떤 상태일까?
2. 리셋이란 무엇인가
3. 피에조 스피커를 이용하여 brownout이 발생한 것을 알릴 수 있을까?
4. 스피커의 소리를 내는 함수는 무엇인가?
5. 헤르쯔(hertz)란 무엇이며 무슨 단어의 약어 인가?

연습(Exercises)

1. 소리를 만드는 문장을 작성하자. 시작을 알리는 음과 스케치의 마지막을 알리는 소리를 다르게 하자.
2. 스케치의 중간단계를 알리는 스피커 음이 나는 문장을 작성하라. 이것은 시작 알림 또는 마지막 음과는 다를 것이다.

프로젝트(Projects)

1. TestServoSpeed 스케치를 수정해서 이것이 각각의 시험이 끝나면 것을 확인하는 음을 만든다.
2. TestServoSpeed 스케치를 수정하여 각 각의 바퀴가 차례로 회전하는 대신 양쪽바퀴가 회전하도록 하자. 오른쪽 바퀴가 왼쪽 바퀴와 방향이 반대가 되도록 하자.

제3장 해답

질문 해답(Question Solutions)

1. 오류가 발생한 행동은 예상하지 않는 방향으로 가거나 혼란스럽게 춤을 추는 증상이 포함된다.
2. 아두이노가 스케치를 시작부분부터 다시 실행 하는 것. 리셋 버튼을 누르고/놓을 때나 아두이노가 brownout으로 충분한 전력을 공급 받지 못할 때 발생된다.
3. 피에조 스피커를 작동 시켜 음을 내는 문장을 스케치의 첫 부분에 넣는다면 전압저하(brownout)가 발생할 때 소리를 발생하게 된다. 그렇게하여 배터리를 교환해야 할지 주행 코드에 오류가 발생한지를 확인할 수 있다.
4. tone()함수
5. 해르쯔(hertz)는 1초당 신호가 몇 번 반복되는지 측정한 횟수이다. 이것의 약자는 Hz.

실습 해답(Exercises Solutions)

1. tone(4, 2000, 1500); //example, your tone may be different.
2. tone(4, 4000, 75); //example, your tone may be different.

프로젝트 해답

1. Add E2 해답의 끝의 for()문 끝에 추가를 하자.

==========

```
/*
 Robotics with theABot - Chapter 3, Project 1
*/

#include <Servo.h>                      // Include servo library

Servo servoLeft;                        // Declare left servo signal
Servo servoRight;                       // Declare right servo signal
```

```
void setup()                                // Built in initialization block
{
  tone(4, 3000, 1000);                      // Play tone for 1 second
  delay(1000);                              // Delay to finish tone

  Serial.begin(9600);                       // Set data rate to 9600 bps
  servoLeft.attach(13);                     // Attach left signal to P13
}

void loop()                                 // Main loop auto-repeats
{

  // Loop counts with pulseWidth from 1375 to 1625 in increments of 25.

  for(int pulseWidth = 1375; pulseWidth <= 1625; pulseWidth += 25)
  {
    Serial.print("pulseWidth = ");          // Display pulseWidth value
    Serial.println(pulseWidth);
    Serial.println("Press a key and click"); // User prompt
    Serial.println("Send to start servo...");

    while(Serial.available() == 0);         // Wait for character
    Serial.read();                          // Clear character

    Serial.println("Running...");
    servoLeft.writeMicroseconds(pulseWidth);    // Pin 13 servo speed = pulse
    delay(6000);                            // ..for 6 seconds
    servoLeft.writeMicroseconds(1500);          // Pin 13 servo speed = stop
    tone(4, 4000, 75);                      // Test complete
  }
}
=================================================================
```

2. 반대 방향으로 같은 속도를 만들기 위해서Servo servoRight;와 servoRight.attach(12);를 추가하자.

servoRight.writeMicroseconds(1500 + (1500 - pulseWidth))

6초간 실행된 후 servoRight.writeMicroseconds(1500)가 추가된다.

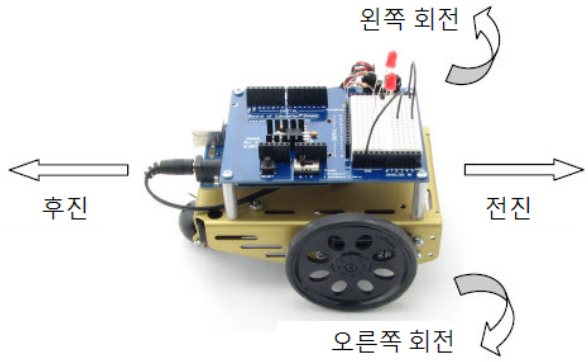
==========

```
/*
 Robotics with theABot - Chapter 3, Project 2
*/

#include <Servo.h>                          // Include servo library

Servo servoLeft;                            // Declare left servo signal
Servo servoRight;                           // Declare right servo signal

void setup()                                // Built in initialization block
{
  tone(4, 3000, 1000);                      // Play tone for 1 second
  delay(1000);                              // Delay to finish tone

  Serial.begin(9600);                       // Set data rate to 9600 bps
  servoLeft.attach(13);                     // Attach left signal to P13
  servoRight.attach(12);                    // Attach right signal to P12
}

void loop()                                 // Main loop auto-repeats
{

  // Loop counts with pulseWidth from 1375 to 1625 in increments of 25.

  for(int pulseWidth = 1375; pulseWidth <= 1625; pulseWidth += 25)
  {
    Serial.print("pulseWidth = ");          // Display pulseWidth value
    Serial.println(pulseWidth);
```

```
    Serial.println("Press a key and click"); // User prompt
    Serial.println("Send to start servo...");

    while(Serial.available() == 0);             // Wait for character
    Serial.read();                               // Clear character

    Serial.println("Running...");
    servoLeft.writeMicroseconds(pulseWidth);    // Pin 13 servo speed = pulse
    // Pin 12 servo opposite direction of pin 13 servo.
    servoRight.writeMicroseconds(1500 + (1500 - pulseWidth));
    delay(6000);                                 // ..for 6 seconds
    servoLeft.writeMicroseconds(1500);           // Pin 13 servo speed = stop
    servoRight.writeMicroseconds(1500);          // Pin 12 servo speed = stop
    tone(4, 4000, 75);                           // Test complete
  }
}
```

제4장 ABOT 주행

이 장에서는 ABOT 동작을 조정하기 위한 다른 프로그래밍 기법을 소개한다. 먼저 로봇이 기본적으로 어떻게 움직이는지를 이해하고, 각 주행 경로에 대한 함수를 만들 것이다. 이 장 후미에 ABOT 스스로 주행하기 위하여 로봇센서 입력 값에 대한 응답으로 함수들을 호출하는 프로그램을 스케치할 것이다.



이 장의 실습 목표는 다음과 같다.

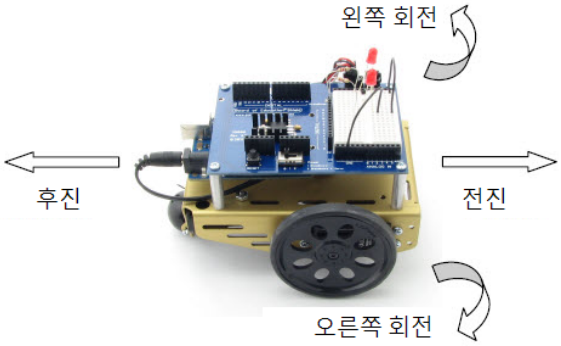
1. 기본 동작 수행하기 : 전진, 후진, 좌회전, 우회전, 좌·우 축회전 그리고 정지
2. 실습1로부터 주행 동작을 좀 더 정밀하게 조절하기.
3. ABOT이 미리 예정된 거리를 주행하도록 만들기 위해 서보(servo) 동작 시간을 계산할 수 있는 수학 사용하기
4. ABOT이 점진적으로 가속 및 감속 주행하도록 스케치하기
5. 기본적인 주행을 실행하기 위해 함수를 만들 것이며, 그 함수가 스케치 내 필요한 곳 어디에서라도 포함될 것이다.
6. 사용자 주행을 빠르게 정의하는 함수를 디자인하기
7. 복잡한 주행들은 배열로 저장하고, 이러한 주행들을 다시 호출할 수 있도록 스케치하기.

이 장에서 배울 내용들

실습1 : ABOT 기본 주행
실습2 : 기본 주행 조정
실습3 : 거리 계산
실습4 : 램프 주행
실습5 : 함수를 이용한 간단 주행
실습6 : 사용자 주행 함수
실습7 : 배열을 이용한 주행 시퀀스
제4장 종합

실습1: ABOT 기본 주행

첫 번째 단계는 회전! 아래 그림은 ABOT의 한 쪽을 기준으로 전진, 후진, 좌회전, 우회전을 나타낸다.



앞으로 움직이기

차동차 바퀴를 앞으로 움직이기 위해 무엇을 해야 하는지 생각해본 적이 있는가? 바퀴들은 자동차의 양쪽에서 서로 상반되는 방향으로 회전한다. 마찬가지로 ABOT도 앞으로 전진하기 위해 왼쪽 바퀴는 반시계 방향으로 회전해야 하고, 오른쪽 바퀴는 시계 방향으로 회전해야 한다.

스케치는 각 서보의 속도와 방향을 제어하기 위해 서보(Servo) 라이브러리의 writeMicroseconds 함수를 사용할 수 있다는 것을 기억하라. 그래서, 로봇은 새로운 속도와 방향을 선택하기 전에 임의의 시간에 대한 서보 동작을 유지하기 위해 지연(delay) 함수를 사용할 수 있다. 다음의 예제는 ABOT이 3초간 앞으로 굴러간 후 정지하는 것을 나타낸다.

예제 스케치 : ForwardThreeSeconds

√ ABOT의 전원 스위치를 1로 놓고, 배터리 팩을 아두이노에 연결한다.
√ 프로그램을 작성, 저장 그리고 아두이노에 ForwardThreeSeconds를 업로드 한다.
√ 프로그램 케이블을 분리하고 ABOT을 바닥에 내려놓는다.
√ Reset 버튼을 누르고 있는 동안, 스위치는 위치 3으로 움직이고, 그리고 가도록 놓는다. ABOT가 3초 동안 앞쪽으로 전진한다.

// Robotics with the BOE Shield – ForwardThreeSeconds
// Make the ABOT roll forward for three seconds, then stop.

```
#include <Servo.h>                      // Include servo library

Servo servoLeft;                        // Declare left and right servos
Servo servoRight;

void setup()                            // Built-in initialization block
{
  tone(4, 3000, 1000);                  // Play tone for 1 second
  delay(1000);                          // Delay to finish tone

  servoLeft.attach(13);                 // Attach left signal to pin 13
  servoRight.attach(12);                // Attach right signal to pin 12

  // Full speed forward
  servoLeft.writeMicroseconds(1700);    // Left wheel counterclockwise
  servoRight.writeMicroseconds(1300);   // Right wheel clockwise
  delay(3000);                          // ...for 3 seconds

  servoLeft.detach();                   // Stop sending servo signals
  servoRight.detach();
}
```

```
void loop()                              // Main loop auto-repeats
{                                         // Empty, nothing needs
repeating
}
```

<ForwardThreeSeconds 작동 원리>

첫째, 서보(Servo) 라이브러리가 포함되어야만 여러분들의 스케치가 여러 함수들을 사용할 수 있다.

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

다음, 서보(Servo)의 인스턴스가 선언되어야 하고, 각 바퀴에 대해 각각의 이름이 선언되어져야 한다.

```
Servo  servoLeft;
Servo servoRight;
```

객체의 인스턴스

객체는 하나의 스케치 안에서 복사와 재사용이 여러 번 가능한 미리 작성된 코드 블록이다. 객체 인스턴스(Instance)라 불리는 각 복사는 다르게 설정될 수 있다. 예를 들면, 두 개의 선보(Servo) 선언은 servoLeft와 servoRight 이름으로 객체 코드의 두 개 인스턴스를 만든다. 그래서 각 인스턴스 내에 있는 함수들은 각각 다르게 불리고 설정된다. 그래서 servoLeft.attach(13)는 servoLeft 객체 인스턴스가 서보 제어 신호를 핀 13번으로 보내기 위한 설정이다. servoRight.attach(12)는 servoRight 객체 인스턴스가 12번 핀으로 신호를 보내기 위한 설정을 의미한다.

스케치는 setup 함수 안에서 자동으로 시작한다. 이 함수는 자동으로 반복적인 동작을 유지하는 반복(loop) 함수에서 작동하기 전에 일단 setup 함수에서 코드를 실행한다.
그러므로 우리는 ABOT이 한번만 앞으로 전진과 정지하기를 원하면, 모든 코드가 setup함수 내에 위치해도 된다. 루프 함수를 빈 공간으로 남겨두어도 괜찮다.

모든 움직임을 스케치하는데 있어서, 첫 번째 행동은 셋업이 피에조스피커 비프(beep) 소리를 만드는 것이다. tone 함수 호출은 3 kHz에서 동작하는 피에조 스

피커가 1초 동안 지속되는 신호를 디지털 핀 4번으로 전송한다. 그러므로 tone 함수는 코드가 실행되는 동안 배경소리로서 작동하고, delay(1000)는 tone함수가 완료될 때까지 ABOT이 움직이는 것을 방지하기 위한 것이다.

```
 void setup()                               // Built-in initialization
  {
    tone(4, 3000, 1000);                    // Play tone for 1 second
    delay(1000);                            // Delay to finish tone
```

다음, servoLeft 객체 인스턴스는 디지털 핀 13으로 할당되고 servoRight instance는 핀 12로 할당된다. 이것은 servoLeft.writeMicroseconds가 서보 제어 신호를 핀 13 으로 보내는 것에 영향을 미친다. 또한, servoRight.writeMicroseconds가 핀 12로 신호를 보내는 것에 영향을 미칠 것이다.

```
    servoLeft.attach(13);                   // Attach left signal to pin 13
    servoRight.attach(12);                  // Attach right signal to pin 12
```

ABOT이 앞으로 움직이기 위해 좌와 우의 바퀴는 서로 반대로 움직인다는 것을 상기하자. servoLeft.writeMicroseconds(1700)는 왼쪽 서보를 반시계방향으로 전속력으로 회전하는 것이다.
그리고 servoRight.writeMicroseconds(1300)는 오른쪽 바퀴를 전속력으로 시계방향으로 회전하는 것이다. delay(3000)은 서보가 3초 동안 계속 동작을 유지한다는 것을 의미한다. 지연 후, servoLeft.detach and servoRight.detach는 서보 신호를 끊고, 로봇은 정지하게 된다.

```
    // Full speed forward
   servoLeft.writeMicroseconds(1700);  // Left wheel counterclockwise
   servoRight.writeMicroseconds(1300); // Right wheel clockwise
   delay(3000);                        // ...for 3 seconds

    servoLeft.detach();                // Stop sending servo signals
    servoRight.detach();
}
```

setup 함수가 완료된 후, 스케치는 자동으로 반복(loop) 함수를 실행할 것이다. 이번 스케치에서는 빈 공간으로 남겨 놓았으며, 무한 반복으로 동작하지는 않는다.

indefinitely.

```
 void loop()                    // Main loop auto-repeats
{
}
```

사용자 차례 - 거리 조정

동작하는 거리를 변화시켜 보고 싶은가? 이것은 delay(3000)내의 값을 변화시키면 된다. 예를 들면, delay(1500)은 ABOT을 절반만 움직일 것이다. 마찬가지로 delay(6000)은 두 배의 시간을 가지므로 두 배의 거리로 이동한다.

√ 위의 두 가지 경우에 대하여 스케치를 한 후 다시 업로드시켜 변화를 살펴보자. ABOT의 이동 거리가 변하였습니까?

<후진, 회전 그리고 중심축 회전 움직임>

여러분들의 ABOT의 다른 모든 움직임은 servoLeft와 servoRight writeMicroseconds 호출 안에 있는 us 파라미터의 다른 조합들이다.
예를 들면, 다음의 두 가지 호출은 ABOT을 후진하도록 만들 것이다.

```
  // Full speed backwards
 servoLeft.writeMicroseconds(1300);    // Left wheel clockwise
 servoRight.writeMicroseconds(1700);   // Right wheel counterclockwise
```

다음의 두 가지 호출 함수는 ABOT이 좌회전하도록 만들기 위해 위치할 것이다:

```
  // Turn left in place
 servoLeft.writeMicroseconds(1300);    // Left wheel clockwise
 servoRight.writeMicroseconds(1300);   // Right wheel clockwise
```

다음 두 가지 호출 함수는 ABOT이 우회전하도록 만들기 위해 위치할 것이다:

```
  // Turn right in place
 servoLeft.writeMicroseconds(1700);    // Left wheel counterclockwise
 servoRight.write Microseconds(1700); // Right wheel counterclockwise
```

ABOT이 앞으로 이동, 좌회전, 우회전 그리고 후진하도록 하나의 스케치에 앞에서 정의한 모든 명령들을 조합해 보자.

예제 스케치 : ForwardLeftRightBackward

√ 입력, 저장 그리고 ForwardLeftRightBackward를 업로드한다.

√ ABOT이 앞으로-왼쪽-오른쪽-뒤쪽으로 움직이는 것을 확인한다.

```
// Robotics with the BOE Shield - ForwardLeftRightBackward
// Move forward, left, right, then backward for testing and tuning.

#include <Servo.h>                    // Include servo library

Servo servoLeft;                      // Declare left and right servos
Servo servoRight;

void setup()                          // Built-in initialization block
{
  tone(4, 3000, 1000);                // Play tone for 1 second
  delay(1000);                        // Delay to finish tone

  servoLeft.attach(13);               // Attach left signal to pin 13
  servoRight.attach(12);              // Attach right signal to pin 12

  // Full speed forward
  servoLeft.writeMicroseconds(1700);     // Left wheel counterclockwise
  servoRight.writeMicroseconds(1300);    // Right wheel clockwise
  delay(2000);                           // ...for 2 seconds

  // Turn left in place
  servoLeft.writeMicroseconds(1300);    // Left wheel clockwise
```

```
  servoRight.writeMicroseconds(1300);  // Right wheel clockwise
  delay(600);                          // ...for 0.6 seconds

   // Turn right in place
  servoLeft.writeMicroseconds(1700);   // Left wheel counterclockwise
  servoRight.writeMicroseconds(1700);  // Right wheel counterclockwise
  delay(600);                          // ...for 0.6 seconds

  // Full speed backward
  servoLeft.writeMicroseconds(1300);   // Left wheel clockwise
  servoRight.writeMicroseconds(1700);  // Right wheel counterclockwise
  delay(2000);                         // ...for 2 seconds

  servoLeft.detach();                  // Stop sending servo signals
  servoRight.detach();
}

void loop()                            // Main loop auto-repeats
{                                     // Empty, nothing needs repeating
}
```

팁 - 이 스케치를 빨리 입력하기 위해, 주행을 만드는 4개 라인에 대하여 4개를 복사하기 위해 아두이노 소프트웨어 에디터 메뉴 툴(복사와 붙이기)를 이용하라(참고, servoLeft.writeMicroseconds, servoRight.writeMicroseconds 그리고 delay). 그런 다음, 각각에 대하여 수정하라.

사용자 차례 - 중심축 회전

여러분들은 ABOT이 하나의 바퀴를 축으로 중심회전을 할 수 있다. 그 방법은 하나의 바퀴는 돌고 있는 동안 다른 하나의 바퀴는 정지해 있는 것이다. 다음은 전진과 후진의 축회전에 대한 4가지 루틴이다.

```
  // Pivot forward-left
 servoLeft.writeMicroseconds(1500);    // Left wheel stop
```

```
servoRight.writeMicroseconds(1300);  // Right wheel clockwise

// Pivot forward-right
servoLeft.writeMicroseconds(1700);   // Left wheel counterclockwise
servoRight.writeMicroseconds(1500);  // Right wheel stop

// Pivot backward-left
servoLeft.writeMicroseconds(1500);   // Left wheel stop
servoRight.writeMicroseconds(1700);  // Right wheel counterclockwise

// Pivot backward-right
servoLeft.writeMicroseconds(1300);      // Left wheel clockwise
servoRight.writeMicroseconds(1500);     // Right wheel stop
```

√ ForwardLeftRightBackward를 PivotTests로 저장하라.
√ 각각의 writeMicroseconds 호출 내에 있는 us 파라미터를 변화시키면 스케치는 4가지 축회전을 실행할 것이다.: 전진, 좌회전, 우회전 그리고 후진
√ 수정된 스케치를 실행해보자. 그리고 여러 가지의 중심축 회전 움직임이 작동되는지 확인하라.
√ 각 주행에 대해 delay 호출과 함께 실험해 보면, 각각의 움직임이 90도 회전하기 위해 충분히 긴 시간동안 달릴 것이다.

실습2: 기본적인 주행 조정

여러분의 ABOT이 전속력으로 15초간 앞쪽으로 달리도록 지시하는 스케치를 상상하라. 로봇이 앞쪽으로 똑바로 주행한다고 가정했을 경우, 만약 여러분의 로봇이 주행하는 동안 왼쪽이나 오른쪽으로 방향이 약간 틀어진다면 무슨 일이 일어나겠는가? 더 이상 ABOT을 분리할 필요가 없고 이것을 고정하기 위해 드라이버로 서보들을 재조정할 필요도 없다. 여러분들은 양쪽 바퀴 주행을 같은 속도로 맞추는 간단한 스케치를 통하여 조정 작업을 해결할 수 있다. 드라이버는 하드웨어적인 조정을 고려할 때 사용되고, 프로그램은 소프트웨어적인 조정을 수행할 수 있다.

ABOT 경로 수정

그래서 ABOT이 직선 대신에 곡선으로 주행하는가? 하나의 서보로부터 다른 서보까지 최고 속도가 변화하므로, ABOT에서 이 점진적인 회전을 유발하는 것은, 하나의 바퀴가 다른 반대편의 바퀴보다 약간 빠르게 회전하는 것이다.

이것을 올바르게 하기 위해 가장 먼저 해야 할 것은 여러분의 ABOT이 꾸불꾸불하게 진행할 경우 방향성과 얼마나 많이 그렇게 되었는지를 알아보기 위해 오랜 시간 동안 지속적인 앞쪽방향 진행을 실험하는 것이다.

예제 스케치: ForwardTenSeconds

√ ForwardThreeSeconds을 열고 ForwardTenSeconds로 다시 저장한다.
√ delay(3000)을 delay(10000)으로 변경하고 제목과 참고사항을 변경한다.
√ 파워 스위치를 1로 설정하고 스케치를 업로드한 후, USB 케이블을 분리한다.
√ ABOT을 부드럽고 깨끗한 긴 바닥위에 놓는다.
√ 전원 스위치를 2로 옮기는 동안 Reset 버튼을 누른 상태로 유지하고, 그런 다음 로봇을 놓는다.
√ Reset 버튼을 누르자. 그 다음에 ABOT이 10초 동안 앞으로 전진하면서 오른쪽과 왼쪽으로 휙휙 돌아가는지를 유심히 살펴보시오.

```
// Robotics with the BOE Shield - ForwardTenSeconds
```

// Make the BOE Shield-Bot roll forward for ten seconds, then stop.

```
#include <Servo.h>                  // Include servo library

Servo servoLeft;                    // Declare left and right servos
Servo servoRight;

void setup()                        // Built-in initialization block
{
  tone(4, 3000, 1000);              // Play tone for 1 second
  delay(1000);                      // Delay to finish tone

  servoLeft.attach(13);             // Attach left signal to pin 13
  servoRight.attach(12);            // Attach right signal to pin 12

  // Full speed forward
  servoLeft.writeMicroseconds(1700);  // Left wheel counterclockwise
  servoRight.writeMicroseconds(1300); // Right wheel clockwise
  delay(10000);                       // ...for 10 seconds

  servoLeft.detach();                 // Stop sending servo signals
  servoRight.detach();
}

void loop()                           // Main loop auto-repeats
{                                     // Empty, nothing needs repeating
}
```

사용자 차례 – ABOT이 경로를 따라 똑바로 가도록 서보 속도 조정

만약 여러분의 ABOT이 앞쪽으로 똑바로 진행하기를 원할 때 약간 회전을 한다면, 그 해결책은 아주 간단하다. 단지 빠른 바퀴의 속도를 늦추기만 하면 된다. servo변환 곡선 그래프에서 가장 알맞은 서보의 속도 제어는 1400 ~ 1600 µs 범위라는 것을 기억하라.

표 4-1: writeMicroseconds(us) 함수내의 us 파라미터

최고 스피드 시계방향	선형 스피드 영역 출발	전체 정지	선형 스피드 영역 끝	최고 스피드 반시계방향
1300	1400	1500	1600	1700

여러분의 ABOT이 서서히 왼쪽으로 회전하도록 해보자. 이것은 오른쪽 바퀴가 왼쪽 바퀴보다 빠르게 회전한다는 것을 의미한다. 왼쪽 바퀴는 가능한 빠르게 이미 회전하고 있으므로, 로봇을 똑바른 방향으로 진행하도록 하기 위해 오른쪽 바퀴의 속도를 늦춰줄 필요가 있다. 속도를 늦추기 위해 servoRight.writeMicroseconds(us)함수내의 파라미터를 1500에 가깝게 변화시킨다. 먼저 1400을 적용한다. 아직 너무 빠르게 움직이는가? 값을 1410으로 변화시킨다. ABOT이 더 이상 왼쪽으로 회전하지 않을 때까지 파라미터 값을 10씩 증가시켜 가면서 실험한다. 만약 어떤 초과 "straight" 조정과 여러분의 ABOT이 오른쪽으로 돌아가기 시작한다면, 보다 적은 값에 의해 us파라미터를 감소시켜라. 여러분의 ABOT이 똑바로 앞으로 갈 때까지 us 파라미터를 계속 조정하라. 이것을 반복 과정이라 부른다. 그리고 이것은 올바른 값을 가지기 위해 시도와 개선의 반복 작업을 의미한다.

만약 ABOT이 **왼쪽 방향 대신 오른쪽 방향으로 회전한다면**, 왼쪽 바퀴를 천천히 회전시켜야 한다. servoLeft.writeMicroseconds(1700)으로 시작해서 서서히 파라미터 값을 감소시켜야 한다. 1600을 적용해 보라. 로봇이 똑바로 움직이든가 다른 방향으로 회전할 때까지 10씩 감소시켜 본다. 그리고 너무 지나치게 변화된다면 2씩 감소하라. ForwardTenSeconds를 수정하면 여러분의 ABOT이 앞으로 똑바로 진행할 것이다.

√ ForwardTenSeconds을 수정하라. 그리고 여러분의 ABOT이 똑바로 가도록 만들어라.

√ 여러분의 writeMicroseconds(us)함수 호출에 대해 각 서보가 최상의 us 파라미터를 가지는 표시를 테이프나 스티커로 만들어 사용하라.

√ 만약 ABOT이 이미 똑바로 움직인다면, 그 효과를 보기 위해 약간의 수정이 필요하다. 이것은 ABOT이 직선 라인 대신 커브로 주행하는 원인이다.

여러분의 ABOT이 후진하기 위해 프로그램할 때, 완전히 다른 상황을 발견할 것이다.

√ BoeBotForwardTenSeconds 을 수정하여 ABOT이 10초동안 뒤로 움직이도록 만든다.
√ 똑바른 라인에 대한 테스트를 반복한다.
√ ABOT이 똑바로 뒤로 움직이기 위해 writeMicroseconds함수 호출에 대한 us 파라미터 수정 과정을 반복한다.

회전 조절

ABOT이 일정한 장소에서 회전할 때 소비되는 시간의 양은 얼마나 멀리 회전 하는지를 결정한다. 그래서 회전 조절하는 것은 여러분들이 해야할 모든 것은 다른 양의 시간에 대한 회전을 만들기 위해 지연 함수의 ms파라미터를 조절하는 것이다.

ABOT이 90도(1/4 바퀴) 이상 회전을 한다고 하자. delay(580) 혹은 delay(560)을 시도하라. 만약 충분하게 회전하지 않는다면, 지연 함수의 ms 파라미터를 20ms씩 증가시키면서 진행한다.

실질적인 작은 변화의 차이는 20 정도이다. 서보제어 신호는 매 20ms마다 보내지며, 사용자의 지연 함수 호출의 ms파라미터는 20를 곱해서 조정한다.

만약 여러분들이 약간 90도가 조금 넘거나 약간 90도 아래에 해당하는 값을 알고자 한다면, 아주 멀리 회전을 만들 수 있는 값을 선택하고, 그런 다음, 서보를 약간씩 천천히 아래로 내린다. 왼쪽으로 회전하는 경우, 양쪽 writeMicroseconds의 us 파라미터들은 1300에서 1500가까이 변화될 것이다. 1400으로 시작하고 양쪽 서보가 천처히 움직이도록 값을 점진적으로 올려간다. 오른쪽 회전에 대해, us 파라미터를 1700부터 1600까지 변화시키면서 출발한다. 그리고 10씩 그 값을 증가시켜 속도를 줄이게 된다.

사용자 연습 - 90도 회전과 스케치 업데이트

정확하게 90도 회전하도록 ForwardLeftRightBackward를 수정하시오.
각 서보의 라벨을 90도 회전을 위한 지연함수 ms 파라미터 표시로 갱신하시오.
전진과 후진 주행을 위해 결정된 값으로 ForwardLeftRightBackward 내의 지연

함수 ms 파라미터를 갱신하시오.

√ 정밀한 90도 회전을 만들기 위해 ForwardLeftRightBackward를 수정하시오.
√ 90도 회전을 위해 적당한 지연 함수의 ms 파라미터 표기를 각 서보의 레이블로 업데이트 하시오.
√ 똑바른 전진과 후진을 주행하기 위한 값으로 ForwardLeftRightBackward 안에 있는 delay 함수의 ms 파라미터 값을 업데이트하시오.

카페트는 로봇 운행의 에러를 유발할 수 있다.
만약 여러분이 ABOT을 카페트 위에서 주행하고 있다면, 완벽한 결과는 기대하지 마라. 털이 놓여있는 카페트 길은 여러분의 ABOT 주행에 영향을 미칠 수 있고, 특히, 장거리에서도 마찬가지이다. 좀 더 정밀한 주행을 위해서는 부드러운 면이 있는 곳을 사용하시오.

실습3: 거리계산

많은 로봇 경진대회에서, 좀 더 정밀한 로봇 주행이 좋은 결과를 나타낸다. 어떤 초보자 단계 로봇 경진대회는 추측항법(dead reckoning)이라 불리어진다. 이 경진대회의 목표는 여러분 로봇이 하나 혹은 그 이상의 장소로 움직이고 그리고 정확히 출발 지점으로 되돌아오는 것이다.

여러분은 휴가 장소나 친척 집으로 가는 동안 내내 여러분의 부모님께 이 질문을 반복해서 물어야 한다는 것을 기억하라.

"우리가 언제 도착하지?"

아마 여러분이 어렸을때 학교에서 나눗셈에 관하여 배웠을 것이고, 여러분은 목적지의 도시까지 얼마나 멀리 있는 가를 알아보기 위해 도로 표지판을 살펴볼 것이고, 그 다음은, 여러분은 차 속도계를 살펴보았다. 여러분은 거리를 속도로 나누어서 도착지까지 얼마의 시간이 소요될지 예측할 것이다. 여러분은 이러한 항목들을 생각으로 하는 것이 아니라 여기에 여러분이 사용할 수 있는 식으로 소개한다.

$$Time = \frac{Distance}{Speed}$$

미국 관습 예 : 만약 여러분이 여러분의 목적지로부터 140마일 떨어져 있고, 여러분이 시간당 70마일을 주행할 수 있다면, 그곳에 도착하는데 소요되는 시간은 약 2시간 정도일 것이다.

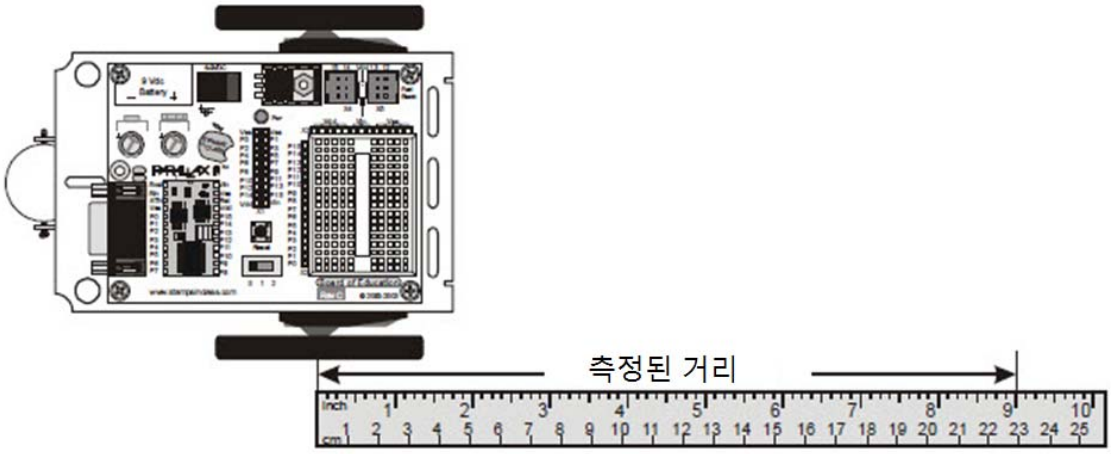
$$\begin{aligned} time &= \frac{140\,miles}{70\,miles/hour} \\ &= 140\,miles \times \frac{1\,hour}{70\,miles} \\ &= 2\,hours \end{aligned}$$

미터법 계산 : 만약 여러분의 목적지가 200킬로미터에 떨어져 있고 시간당 100킬로미터로 운행 한다면, 목적지에 도착하는데 2시간이 걸릴 것이다.

여러분은 목적지가 많이 떨어져 있는 것을 제외하고는 ABOT에 대해서도 동일하게 적용할 수 있다. 다음 식을 사용한다.

$$servo\ run\ time = \frac{BOE Shield\_Bot\ distance}{BOE Shield\_Bot\ speed}$$

√ 입력, 저장 그리고 ForwardOneSecond를 실행하라.
√ 여러분의 ABOT을 눈금자 옆에 놓는다.
√ 지면에 닿는 바퀴 부분이 눈금자의 0 인치/센티미터에 놓이도록 만든다.
√ 스케치를 다시 실행하기위해 보드위에 있는 Reset 버튼을 누른다.
√ 바퀴가 현재 지면에 닿아있는 부분을 측정하여 ABOT의 바퀴가 얼마나 멀리 주행 했는지를 기록하라 : _____ (in or cm).



// Robotics with the BOE Shield - ForwardOneSecond
// Make the ABOT roll forward for one seconds, then stop.

```
#include <Servo.h>                    // Include servo library

Servo servoLeft;                      // Declare left and right servos
Servo servoRight;
```

```
void setup()                          // Built-in initialization block
{
  tone(4, 3000, 1000);               // Play tone for 1 second
  delay(1000);                        // Delay to finish tone

  servoLeft.attach(13);               // Attach left signal to pin 13
  servoRight.attach(12);              // Attach right signal to pin 12
            // Full speed forward
  servoLeft.writeMicroseconds(1700);  // Left wheel counterclockwise
  servoRight.writeMicroseconds(1300); // Right wheel clockwise
  delay(1000);                                // ...for 1 second

  servoLeft.detach();                         // Stop sending servo signals
  servoRight.detach();
}

void loop()                                   // Main loop auto-repeats
{                                            // Empty, nothing needs repeating
}
```

여러분의 ABOT의 속도로서 기록된 거리는 unit/second 이다. 이제, 여러분은 ABOT이 특정 거리를 가기 위해 로봇을 몇 초 동안 움직여야 하는지를 알 수 있다.

1초당 인치와 센티미터
인치의 생략형은 “in” 이고 센티미터의 생략형은 “cm”이다. 마찬가지로 “인치/초”의 축약어는 “in/s”이고, “센티미터/초”의 축약어는 “cm/s”이다. 두 가지 모두 ABOT의 편리한 속도 측정 방법이다. 1인치는 2.54센티미터이다. 인치의 센티미터 변환은 인치 값 에 2.54를 곱하면 된다. 센티미터의 인치 변환은 센티미터 값에 2.54를 나누면 된다.

여러분의 계산은 초 단위라는 것을 명심하라. 그러나 지연 함수의 파라미터 값은 milisecond 값으로 되어 있다. 그래서 여러분의 계산 결과 값에 1000을 곱하여

사용해야 한다. 그리고 여러분의 지연함수 호출 때 이 값을 사용한다. 예를 들면, 여러분의 ABOT이 2.22초 동안 움직이도록 만들고자 한다면, 여러분은 writeMicroseconds함수를 호출 한 후 delay(2220) 사용해야 할 것이다.

미국식 방법 예 : 9 in/s에서 여러분의 ABOT이 20인치 만큼 주행하기 위해 2.22초 동안 움직여여야 한다.

$$\begin{aligned} time &= \frac{20\,in}{9\,in/s} \\ &= 20\,in \times \frac{1s}{9\,in} \\ &= 2.22s \end{aligned}$$

미터식 계산 예 : 23cm/s에서, 여러분의 ABOT은 51cm를 주행하기위해 2.22초 동안 움직여야 한다.

$$\begin{aligned} time &= \frac{51cm}{23\,cm/s} \\ &= 51cm \times \frac{1s}{23\,cm} \\ &= 2.22s \end{aligned}$$

같은 결과를 나타내는 양쪽 예제:

$$\begin{aligned} time\,(ms) &= time\,(s) \times \frac{1000\,ms}{s} \\ &= 2.22s \times \frac{1000\,ms}{s} \\ &= 2220\,ms \end{aligned}$$

그래서 writeMicroseconds 함수를 부른 후 delay(2220)를 사용하라:

```
servoLeft.writeMicroseconds(1700);
servoRight.writeMicroseconds(1300);
delay(2220);
```

사용자 차례 - 여러분의 ABOT의 거리

이제, 여러분이 선택한 거리를 시도해 보자.

√ 여러분의 ABOT이 얼마나 멀리 움직이기를 원하는지 거리를 결정하라.

√ 여러분이 선택한 거리에 대해 직진으로 얼마만큼의 시간 동안 로봇이 운행해야 하는지를 아래 식을 이용하여 구하라.

$$ms\ servo\ run\ time = \frac{BOE\ Shield\_Bot\ distance}{BOE\ Shield\_Bot\ speed} \times \frac{1000\,ms}{s}$$

√ ABOT이 여러분이 계산한 시간만큼 직진으로 움직이도록 ForwardOneSecond를 수정하고 실행하라. 얼마나 가깝게 갔는가?

여러분의 ABOT의 바퀴가 홀에 들어갔을 경우 이를 빠져 나올 수 있도록 엔코더(encoder)라 불리는 도구를 사용하여 거리의 정확성을 높이시오.

실습4: 경사(ramping) 주행

램핑은 갑작스런 출발과 정지 대신 servo의 속도를 점진적으로 증가하거나 감소하는 방법이다. 이 기술은 ABOT의 배터리와 서보들의 수명을 길게 할 수 있다.

램핑을 위한 프로그래밍

아래 예제는 속도가 최대가 될 때가지 어떻게 램프하는지의 예제이다. for루프에 정수 변수 이름 speed는, 100번 반복 하도록 하였다. for 루프의 증가 파라미터는 speed+=2로 표현하며, 루프를 매 반복할 때 마다 speed 값은 2씩 증가한다. speed 변수는 writeMicroseconds의 호출의 us 파라미터이다. 이것은 for 루프가 반복할 때마다 값에 영향을 미친다. 각각 반복할 때마다 20ms의 지연을 가진다. 루프 1초당 50번씩 반복한다. 이것은 2씩 증가해서 100까지 속도가 1초 소요된다는 것이며, 그 시점에서 양쪽 서보는 전속력으로 진행할 것이다.

```
for(int speed = 0; speed <= 100; speed+=2)
{
  servoLeft.writeMicroseconds(1500+speed);
  servoRight.writeMicroseconds(1500-speed);
  delay(20);
}
```

다음 순서를 통하여 for 루프의 순서를 자세히 살펴보자.

- 첫 번째 trip : 속도 0, 양쪽 writeMicroseconds 호출은 us 파라미터가 1500일때 끝난다.
- 두 번째 trip: 속도 2, servoLeft.writeMicroseconds(1502)와 servoRight.writeMicroseconds(1498)
- 세 번째 trip: 속도 4, servoLeft.writeMicroseconds(1504)와 servoRight.writeMicroseconds(1496)
- 이 방법을 계속 실행.....
- 100번째 trip: 속도 200, servoLeft.writeMicroseconds(1600) 와 servoRight.writeMicroseconds(1400). 이것은 최고 속도에 상당히 근접하고 있다. 1700과 1300는 지나친 값이다.

예제 스케치: StartAndStopWithRamping

√ 프로그램을 입력, 저장 그리고 StartAndStopWithRamping을 실행하라.

√ ABOT이 전속력일 때까지 점진적으로 가속하며, 잠시 동안 전속력을 유지하고 그리고 정지할 때까지 점진적으로 감소하는 것을 확인하라.

```
// Robotics with the BOE Shield - StartAndStopWithRamping
// Ramp up, go forward, ramp down.

#include <Servo.h>                  // Include servo library

Servo servoLeft;                    // Declare left and right servos
Servo servoRight;

void setup()                        // Built-in initialization block
{
  tone(4, 3000, 1000);              // Play tone for 1 second
  delay(1000);                      // Delay to finish tone

  servoLeft.attach(13);             // Attach left signal to pin 13
  servoRight.attach(12);            // Attach right signal to pin 12

  for(int speed=0; speed<=100; speed+=2) //Ramp up to full speed.
  {
    servoLeft.writeMicroseconds(1500+speed);
                                    // us = 1500,1502,...1598,1600
    servoRight.writeMicroseconds(1500-speed);
                                    // us = 1500,1498,...1402,1400
    delay(20);                      // 20 ms at each speed
  }

  delay(1500);                      // Full speed for 1.5 seconds

  for(int speed=100;speed>=0;speed-=2)//Ramp from full speed to stop
  {
    servoLeft.writeMicroseconds(1500+speed);
```

```
                                // us = 1600,1598,...1502,1500
    servoRight.writeMicroseconds(1500-speed);
                                // us = 1400,1402,...1498,1500
    delay(20);                  // 20 ms at each speed
  }

  servoLeft.detach();           // Stop sending servo signals
  servoRight.detach();
}

void loop()                     // Main loop auto-repeats
{                               // Empty, nothing to repeat
}
```

여러분 차례 - 다른 주행에 램핑 더하기

여러분은 다른 주행과 램핑을 결합한 루틴을 만들 수 있다. 다음 예는 전진 대신 후진을 전속력으로 어떻게 램핑하는지를 나타낸다. 이 루틴과 전진 방향 램핑 사이의 차이점은 0에서 출발하여 -100까지 speed의 값이 변한다는 것이다.

```
  for(int speed = 0; speed >= -100; speed -= 2)// Ramp stop to full reverse
  {
    servoLeft.writeMicroseconds(1500+speed);      // us = 1500,1498, 1496...1400
    servoRight.writeMicroseconds(1500-speed);     // us = 1500,1502, 1508...1600
    delay(20);                                    // 20 ms at each speed
  }
```

여러분은 안쪽 회전과 바깥쪽 회전에 대한 램핑 루틴을 만들 수 있다. 아래는 오른쪽-회전 램핑의 예이다. 한쪽의 바퀴는 1500+speed와 다른 한쪽의 바퀴는 1500-speed를 하는 대신, 둘 다 1500+speed를 사용한다는 것을 명심하라.

```
  for(int speed = 0; speed <= 100; speed += 2) // Ramp stop to right
turn
  {
    servoLeft.writeMicroseconds(1500+speed);          // us = 1500,1502,
1508...1600
    servoRight.writeMicroseconds(1500+speed);         // us = 1500,1502,
1508...1600
    delay(20);                                        // 20 ms at each speed
  }

  for(int speed = 100; speed >= 0; speed -= 2)// right turn to stop
  {
    servoLeft.writeMicroseconds(1500+speed);          // us = 1600,1598,
1597...1500
    servoRight.writeMicroseconds(1500+speed);         // us = 1600,1598,
1597...1500
    delay(20);                                        // 20 ms at each speed
  }
```

√ ForwardLeftRightBackward 스케치를 열고 ForwardLeftRightBackwardRamping 이름으로 파일을 저장하라.

√ 새로운 스케치를 수정면 여러분의 ABOT이 각각의 주행을 안쪽으로 바깥쪽으로 램프할 것이다. 힌트: 여러분은 위의 작은 부분을 사용할 것이고, StartAndStopWithRamping으로부터 비슷한 작은 부분을 사용할 것이다.

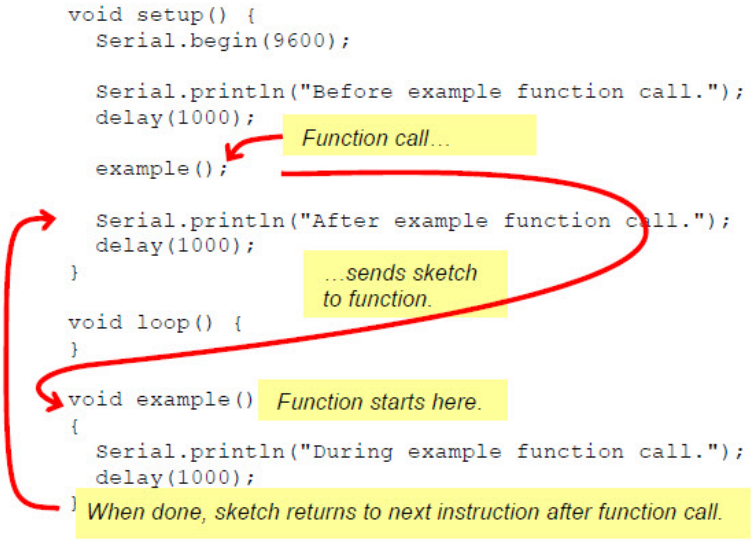
실습5: 함수를 가지는 단순 주행

사전 프로그램된 주행을 실행하기 위한 편리한 방법은 함수를 만드는 것이다. 다음 장에서, 여러분의 ABOT이 장애물을 피하기 위한 주행을 해야 할 것이다 그리고 장애물을 피하기 위한 주요한 구성요소는 미리 작성된 프로그램 주행을 실행하는 것이다.

setup과 loop 함수들은 아두이노 언어로 만들어질 것이다. 그러나 여러분은 여러분의 스케치를 위해 명확한 일을 수행하는 함수들을 추가할 수 있다. 여기서의 작업은 여러분의 스케치로 어떻게 함수를 추가하는지 뿐만 아니라 그러한 함수들을 주행에 재사용할 수 있는 여러 가지 다른 방법들을 소개한다.

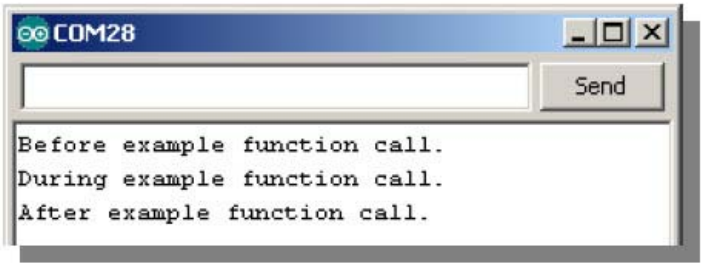
최소 함수 호출

아래 그림은 loop()함수 아래에 프로그램의 맨 마지막에 example함수를 추가로 포함하는 스케치의 한 부분을 나타낸다. 추가된 함수는 void example()함수이다. example()함수의 괄호 안이 비어 있는 것은, 이것을 수행하는데 파라미터가 필요 없다는 것을 의미한다. 그리고 void가 의미 하는 것은 반환(return)값이 없다는 것이다(다음 장에 return 값이 있는 함수를 살펴 볼 것이다). 괄호 { }의 정의는 example 함수의 코드 블록을 의미한다.



위의 그림에서, setup()함수 안에서 example()함수를 호출하고 있다. example()은 스케치가 이 이름을 가리키는 함수를 찾아가서 코드를 실행하고 그것이 완료되었을 경우 되돌아오는 것을 의미한다. 그러므로 스케치는 void example()로 점프할 것이고 괄호 { } 내에 있는 두 줄의 명령을 실행할 것이다. 그 다음은, 함수를 호출한 곳으로 되돌아가고 그 다음 명령이 계속 실행될 것이다. 아래는 여러분이 스케치를 실행할 때 볼 수 있는 일의 순서이다.

1. Serial Monitor 는 "Before example function call."를 나타낸다.
2. 1초 지연 후, 모니터는 "During example function call."를 나타낸다. 왜 그럴까? 메시지를 출력하라는 명령을 가지고 있는 example()함수 호출 때문이고, 그 다음에 1초 시간 지연이 실행된다. 그 다음에, 함수는 example()함수가 호출된 setup()함수내로 되돌아 갈 것이다.
3. Serial Monitor는 "After example function call."를 나타낸다.



예제 스케치 - SimpleFunctionCall

√ 입력, 저장 그리고 SimpleFunctionCall를 업로드한다.
√ 업로드를 진행을 살펴보고, 작업이 완료되는 순간 직렬모니터를 열어라.
√ 터미널을 보고 “Before”, “During” 그리고 “After”로 시작하는 문자열 메시지를 확인하라.

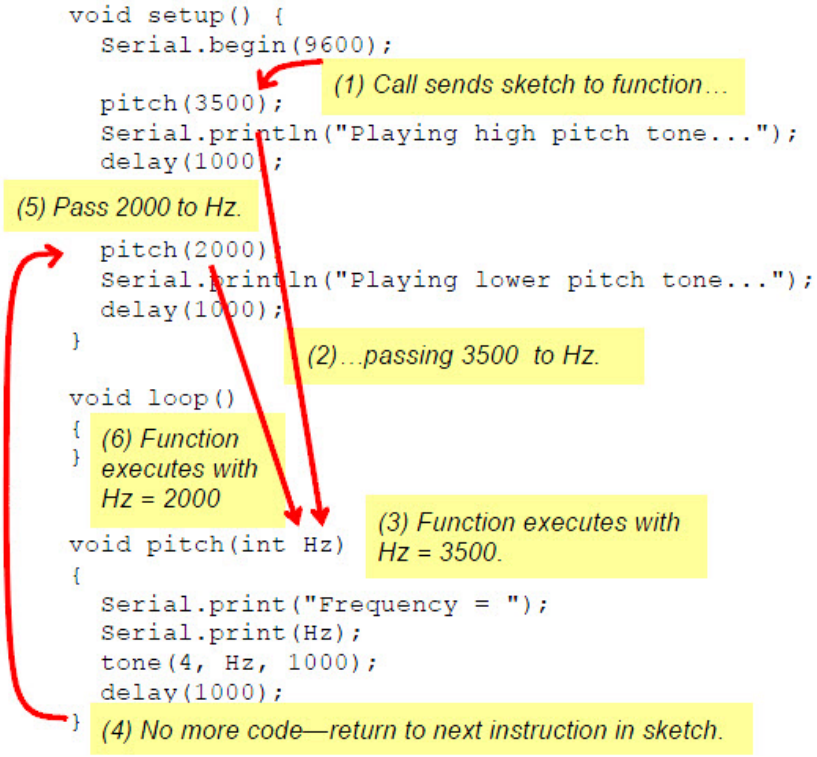
```
// Robotics with the BOE Shield - SimpleFunctionCall
// This sketch demonstrates a simple function call.

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

  Serial.println("Before example function call.");
  delay(1000);

  example();                          // This is the function call

  Serial.println("After example function call.");
  delay(1000);
}

void loop() {
}

void example()                        // This is the function
{
  Serial.println("During example function call.");
  delay(1000);
}
```

<파라미터를 가지는 함수 호출>

함수는 한 개 혹은 그 이상의 파라미터를 가질 수 있다는 것을 기억하라 - 함수가 호출 되었을 때 함수가 받고 사용하는 데이터이다. 아래 그림은 다음 스케치로부터 pitch()함수를 나타낸다. void pitch(int Hz)로 선언되어진다. 1장 활동 3의 아두이노에서 여러가지 데이터 형식으로 변수 값들을 저장하는 것을 상기하라. int는 정수 범위 -32,768부터 32,767까지 범위를 가진다. 프로그램에서 괄호 안에 있는 int Hz는 pitch()함수에 대한 파라미터이다; 이 경우, 지역 변수 Hz가 정수(int) 데이터 형식으로 선언한다.

지역 변수는 함수 내에 선언되어진다. 그리고 이것은 이 함수 내에서만 사용되어진다. 만약 지역 변수가 함수 선언할 때 파라미터로 만들어졌다면, 여기서는 void pitch(int Hz)처럼, 함수가 호출될 때마다 이 값은 초기화된다. 예를 들면, pitch(3500) 호출은 정수 값 3500을 pitch 함수의 int Hz 파라미터로 전달한다.

그래서 첫 번째 pitch 함수가 pitch(3500)으로 호출 되어질 때, 정수 값 3500이 Hz로 전달된다. 이것은 pitch 함수의 코드 블록을 통하여 이 과정이 진행되는 동안 Hz를 3500으로 초기화한다. 두 번째 호출, pitch(2000), pitch함수의 코드 블록을 통하여 스케치가 두 번째 진행되는 동안 Hz를 2000으로 초기화 한다.



예제 스케치 - FunctionCallWithParameter

√ 스케치를 읽고 여러분이 이것을 실행했을 때 보고 들리는 것이 무엇인지를 예측하라.

√ 프로그램을 입력하고 FunctionCallWithParameter를 업로드하라 그런 다음 직렬모니터를 열어라.

√ 여러분의 터미널을 보고 소리를 들어라.

√ 여러분의 예측이 맞았나요? 만약 그렇다면 잘 했어요!!! 만약 그렇지 않다면, 스케치를 자세히 살펴보고 각 함수 호출부터 함수와 되돌아오는 것까지 코드를 따라가서 확인해 보세요. 확실히 하기 위해, 위에서처럼 다른 diagram를 살펴 보자

```
// Robotics with the BOE Shield - FunctionCallWithParameter
// This program demonstrates a function call with a parameter.
```

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

  Serial.println("Playing higher pitch tone...");
  pitch(3500);          // pitch function call passes 3500 to Hz parameter
  delay(1000);

  Serial.println("Playing lower pitch tone...");
  pitch(2000);          // pitch function call passes 2000 to Hz parameter
  delay(1000);
}
void loop()
{
}

void pitch(int Hz)      // pitch function with Hz declared as a parameter
{
  Serial.print("Frequency = ");
  Serial.println(Hz);
  tone(4, Hz, 1000);
  delay(1000);
}
```

여러분 차례 – 두 개의 파라미터까지 함수 확장

다음은 두 개의 파라미터를 사용하는 수정된 pitch함수이다: Hz와 ms. 이 새로운 pitch함수는 소리를 얼마나 지속하는지를 제어하는 것이다.

```
 void pitch(int Hz, int ms)
{
  Serial.print("Frequency = ");
  Serial.println(Hz);
  tone(4, Hz, ms);
  delay(ms);
```

}

아래는 수정된 pitch함수에 대한 두 가지 호출이 있다. 하나는 3500Hz에서 0.5초 동안 지속하는 것이고 다른 하나는 2000Hz에서 1.5초 동안 지속하는 것이다.

```
pitch(3500, 500);
pitch(2000, 1500);
```

이러한 각각의 pitch 호출은 두 개의 값을 가진다는 것에 주목하자. 하나는 Hz로 전달하고, 다른 하나는 ms 파라미터로 전달한다. 함수 내에 있는 값들의 개수 함수 정의에 있는 파라미터의 개수와 일치하여야 한다. 만약 그렇지 않다면 스케치는 컴파일할 수 없을 것이다.

FunctionCallWithParameter를 FunctionCallWithTwoParameters로 저장하시오.
pitch함수를 두 개 파라미터 버전으로 교체하시오.
하나의 파라미터 pitch 호출을 두 개 파라미터 호출로 교체하시오.
수정된 스케치를 동작시키고 동작을 검증하시오.

√ FunctionCallWithParameter을 FunctionCallWithTwoParameters로 저장하라.
√ pitch함수가 두 개의 파라미터가 있는 것으로 대체 한다.
√ 한 개의 파라미터가 있는 pitch함수 호출을 두 개의 파라미터가 있는 pitch함수 호출로 대체한다.
√ 수정된 스케치를 실행한다. 그리고 확인과정을 거친다.

<함수 내에 주행 넣기>

앞으로 전진, 왼쪽 턴, 오른쪽 턴 그리고 뒤로 후진 주행 루틴들을 함수들의 안에 놓자.

예 스케치 - MovementsWithSimpleFunctions

√ 입력, 저장 그리고 MovementsWithSimpleFunctions을 업로드 하라.

```
// Robotics with the BOE Shield - MovementsWithSimpleFunctions
```

// Move forward, left, right, then backward for testing and tuning.

```
#include <Servo.h>                  // Include servo library

Servo servoLeft;                    // Declare left and right servos
Servo servoRight;

void setup()                        // Built-in initialization block
{
  tone(4, 3000, 1000);              // Play tone for 1 second
  delay(1000);                      // Delay to finish tone

  servoLeft.attach(13);             // Attach left signal to pin 13
  servoRight.attach(12);            // Attach right signal to pin 12

  forward(2000);                    // Go forward for 2 seconds
  turnLeft(600);                    // Turn left for 0.6 seconds
  turnRight(600);                   // Turn right for 0.6 seconds
  backward(2000);                   // go backward for 2 seconds

  disableServos();                  // Stay still indefinitely
}

void loop()                         // Main loop auto-repeats
{                                   // Empty, nothing needs repeating
}

void forward(int time)              // Forward function
{
  servoLeft.writeMicroseconds(1700);  // Left wheel counterclockwise
  servoRight.writeMicroseconds(1300); // Right wheel clockwise
  delay(time);                        // Maneuver for time ms
}

void turnLeft(int time)                 // Left turn function
```

```
{
  servoLeft.writeMicroseconds(1300);    // Left wheel clockwise
  servoRight.writeMicroseconds(1300);   // Right wheel clockwise
  delay(time);                          // Maneuver for time ms
}

void turnRight(int time)                // Right turn function
{
  servoLeft.writeMicroseconds(1700);    // Left wheel counterclockwise
  servoRight.writeMicroseconds(1700);   // Right wheel counterclockwise
  delay(time);                          // Maneuver for time ms
}

void backward(int time)                 // Backward function
{
  servoLeft.writeMicroseconds(1300);    // Left wheel clockwise
  servoRight.writeMicroseconds(1700);   // Right wheel counterclockwise
  delay(time);                          // Maneuver for time ms
}

void disableServos()                    // Halt servo signals
{
  servoLeft.detach();                   // Stop sending servo signals
  servoRight.detach();
}
```

여러분은 여러분의 ABOT의 움직이는 패턴을 만든 수 있다; 이것은 ForwardLeftRightBackward 스케치에 의해 만들어지는 것과 같은 것이다. 이것은 같은 움직임을 결과로 나타내는 여러 스케치 방법들 중의 두 번째 예제이다. 이 장의 끝 부분에 다수의 예제를 볼 수가 있다.

사용자 차례 - 루프에서 함수 호출

4 가지의 주행을 계속해서 실행하기 원하는가? setup 함수에서 loop 함수로 4가

지 주행 함수 호출을 옮긴다. 이것을 실행 해보라:

√ MovementsWithFunctionsInLoop와 같은 새로운 이름으로 스케치를 저장한다.
√ //disableServos()처럼 왼쪽에 두 개의 / 를 놓는 방법으로 setup함수 내에 있는 disableServos()함수 호출을 주석 처리한다.
√ loop함수로 부터 // Empty… 코멘트 부분을 제거하라. - 이것은 올바르지 않은 부분이다.

forward(2000), turnLeft(600), turnRight(600) 그리고 backward(2000) 함수 호출을 setup 함수에서 제거하라. 그리고 이것들을 loop함수 내에 붙여넣기를 한다. 완성된 결과는 아래와 같다.

```
 void setup()                      // Built-in initialization block
{
  tone(4, 3000, 1000);             // Play tone for 1 second
  delay(1000);                     // Delay to finish tone

  servoLeft.attach(13);           // Attach left signal to pin 13
  servoRight.attach(12);          // Attach right signal to pin 12

  // disableServos();               // Stay still indefinitely
}

void loop()                        // Main loop auto-repeats
{
  forward(2000);                   // Go forward for 2 seconds
  turnLeft(600);                  // Turn left for 0.6 seconds
  turnRight(600);                  // Turn right for 0.6 seconds
  backward(2000);                 // go backward for 2 seconds
}
```

√ 수정된 스케치를 업로드한 후 지속적으로 4가지의 주행이 연속적으로 반복이 이루어지는지를 확인하라.

실습6: 사용자 주행 함수

마지막 스케치 MovementsWithSimpleFunctions는 길고 모양새가 없는 프로그램이다. 그리고 로봇을 움직이기 위한 네 가지 함수는 거의 같은 것이다. TestManeuverFunction 스케치는 함수의 유사성과 코드의 유연성 장점을 가진다.

TestManeuverFunction는 3개의 파라미터를 가지는 움직임에 대한 maneuver함수를 가진다: speedLeft, speedRight, and msTime

```
void maneuver(int speedLeft, int speedRight, int msTime)
```

아래에 열거된 speedLeft 그리고 speedRight에 대한 규칙은 기억하기 쉽다. 가장 좋은 것은 이 maneuver함수와 함께 더 이상의 시계방향과 반시계방향으로 로테이션에 대한 생각은 할 필요가 없다.

- 로봇이 앞으로 움직이기 위한 양수 값
- 로봇이 뒤로 움직이기 위한 음수 값
- 앞으로 전속력의 값 200
- 뒤로의 전속력 값 -200
- 정지 값 0
- 선형적 속도 조절 범위 100부터 -100

msTime에 대한 규칙들:

- maneuver를 실행하기 위해 여러 개의 양수 ms 값
- servo 신호를 무력화시키기 위한 값 -1

아래의 함수들 호출은 전진-후진-왼쪽-오른쪽-정지 순서로 될 것이다.

```
maneuver(200, 200, 2000);          // Forward 2 seconds
maneuver(-200, 200, 600);          // Left 0.6 seconds
maneuver(200, -200, 600);          // Right 0.6 seconds
maneuver(-200, -200, 2000);        // Backward 2 seconds
maneuver(0, 0, -1);                // Disable servos
```

예제 스케치 - TestManeuverFunction

√ 입력, 저장 그리고 업로드 TestManeuverFunction
√ 전진, 왼쪽, 오른쪽, 후진 그리고 정지 순서로 되는지를 확인하라.

// Robotics with the BOE Shield - TestManeuverFunction
// Move forward, left, right, then backward with maneuver function.

```
#include <Servo.h>                  // Include servo library

Servo servoLeft;                    // Declare left and right servos
Servo servoRight;

void setup()                        // Built-in initialization block
{
  tone(4, 3000, 1000);              // Play tone for 1 second
  delay(1000);                      // Delay to finish tone

  servoLeft.attach(13);             // Attach left signal to pin 13
  servoRight.attach(12);            // Attach right signal to pin 12

  maneuver(200, 200, 2000);         // Forward 2 seconds
  maneuver(-200, 200, 600);         // Left 0.6 seconds
  maneuver(200, -200, 600);         // Right 0.6 seconds
  maneuver(-200, -200, 2000);       // Backward 2 seconds
  maneuver(0, 0, -1);               // Disable servos
}
void loop()                         // Main loop auto-repeats
{                                   // Empty, nothing needs repeating
}
void maneuver(int speedLeft, int speedRight, int msTime)
{
  // speedLeft, speedRight ranges: Backward  Linear  Stop  Linear
Forward
  //                              -200       -100......0......100       200
```

```
  servoLeft.writeMicroseconds(1500+speedLeft); // Set Left servo speed
  servoRight.writeMicroseconds(1500-speedRight);//Set right servo speed
  if(msTime===-1)                              // if msTime = -1
  {
    servoLeft.detach();                        // Stop servo signals
    servoRight.detach();
  }
  delay(msTime);                               // Delay for msTime
}
```

여러분 차례 - 사용자 속도와 지속 제어

maneuver함수를 사용하여, 0은 정지, 100부터 -100사이는 속도 제어 범위 그리고 200과 -200은 servo가 전속력으로 달리도록 하는 것이다.

TestManeuverFunction 스케치는 사용자 maneuver들을 빠르게 정의하는 것이 용이하다. maneuver 함수를 각각 호출 하는 동안 각 바퀴의 회전과 주행에 대한 새로운 파라미터를 전달한다. 예를 들면, 3초 동안 호 모양을 그리기위해 오른쪽 바퀴가 전속력으로 움직이고 있는 동안 왼쪽 바퀴는 절반의 속력으로 움직이도록 만들어 보자. 함수 호출은 다음과 같을 것이다:

```
maneuver(50, 100, 3000);
```

다른 예제는 왼쪽 바퀴를 중심축으로 회전하는 것으로써, 왼쪽 바퀴는 정지 상태를 유지하며 오른쪽 바퀴는 앞으로 움직이는 것이다.

```
maneuver(0, 200, 1200);
```

√ 여러분의 스케치에 예제 maneuver 호출 두 개를 추가하라.

√ 다른 세가지의 바퀴-축회전을 순서대로 추가하라: 전진-오른쪽 축회전, 후진-오른쪽 축회전 그리고 후진-왼쪽 축회전

실습7: 배열을 이용한 주행 순서

어떤 로봇 응용은 연속된 주행을 요구한다. 여러분들은 ABOT이 장애물에 의해 부딪쳤을 경우에 대해 다음 장에 간단한 실행 예제들을 참고할 수 있다. 이러한 것은 앞으로 전진하기 전에 뒤로 물러나야 하고 그리고 회전해야 한다. 그것은 두 개의 주행 동작으로 간단한 연속동작이다.

다른 예는 복도 주행이다. ABOT은 복도를 찾아야 할 것이다. 복도 벽을 찾기 전에 복도로 진입하기 위한 주행 동작을 진행할 것이다.

다른 연속동작이 좀 더 정교할 수 있다. 길고 복잡한 주행의 한 예는 로봇 댄스 경연대회이다.(로봇 댄스 경연대회는 최근에 많은 대중성을 얻고 있다.) 전체 노래에 대해 춤추는 동안, 로봇은 꽤 긴 주행 목록과 주행 시간이 필요할 것이다. 만약 여러분의 스케치가 주행의 기록을 저장할 필요가 있다면, 배열 변수가 이러한 리스트를 만드는데 가장 좋은 도구일 것이다.

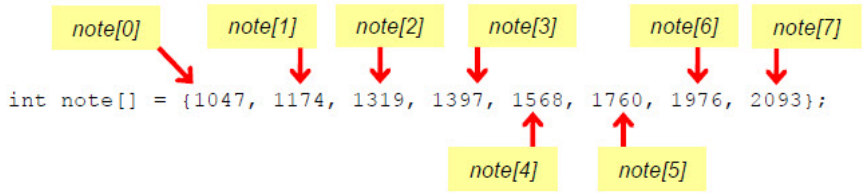
이번 활동은 피에조스피커를 이용하여 간단한 음악 응용을 가지는 배열을 소개한다. 그런 다음, 이것은 스케치가 실행하는 동안 재생하기 위한 실행동작의 시퀀스를 저장하기 위해 배열을 사용하는 두 개의 다른 접근법을 실험할 것이다.

배열이란 무엇인가?

배열은 하나의 이름을 가지는 변수들의 집합이다. 배열 속의 각 변수는 하나의 요소를 의미하게 된다. 다음은 8개의 요소를 가지는 배열 선언이다.

```
int note[] = {1047, 1174, 1319, 1397, 1568, 1760, 1976, 2093};
```

배열 이름은 note이며, 배열 내에 있는 각 요소는 차례대로 순서에 의해 접근할 수 있다. 배열순서는 0부터 시작한다. 그래서 각 요소들은 0부터 7까지 번호가 할당된다. 아래 그림은 어떻게 note[0]가 1047를 저장, note[1]이 1174를 저장, note[2]가 1319를 저장, 그리고 계속해서, 마지막의 note[7]이 2093를 저장하는가를 나타낸다.



여러분의 코드가 변수 myVar에 1397을 저장하고 있는 note[3]를 복사할 필요가 있다고 하자. 여러분의 코드는 다음과 같을 것이다:

```
myVar = note[3];
```

여러분은 스케치에서 배열 요소 값을 역시 바꿀 수 있다. 예를 들면, 다음과 같이 여러분은 1397 값을 1975값으로 바꿀 수 있다.

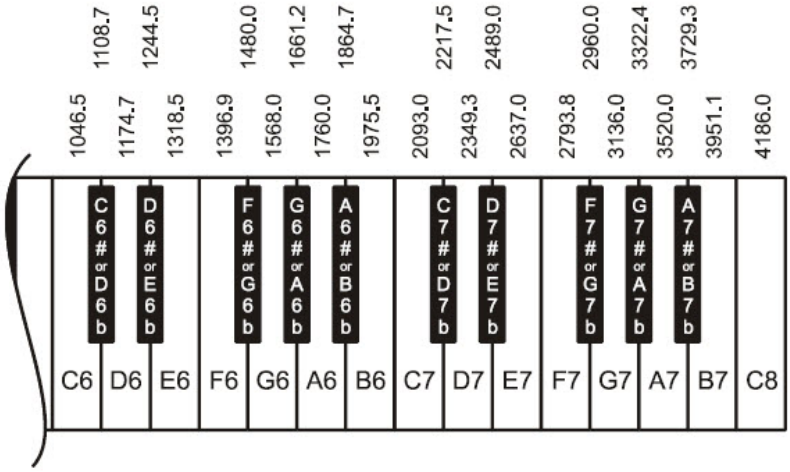
```
note[3] := 1975;
```

배열은 위 그림에 나타낸 것처럼 임의의 값을 가지는 것처럼 미리 초기화할 필요는 없다. 예를 들면, 아래와 같이 8개의 요소를 가지는 배열을 선언할 수 있다.

```
int myArray[8];
```

그런 다음, 여러분의 스케치는 센서 측정값, 직렬모니터로부터 입력되는 값 혹은 여러분이 저장하는 어떤 값이든지 나중에 각 요소의 값을 배열에 채울 수 있다. 아래 그림은 피아노 건반의 오른쪽 부분인 음악 악보를 나타낸다. 각 건반위의 키는 어떤 주파수에서 줄을 진동시킨다.

√ 맨 왼쪽에 있는 여덟 개의 흰색 건반 주파수들과 악보 배열 안에 있는 값들을 비교하라.



다음, 여러분의 ABOT이 어떤 음악 악보를 연주하는 것을 만들기 위해 배열을 사용해보자

<배열 요소 사용>

배열 요소는 그 값을 사용하기 위해 다른 변수로 복사 저장할 필요가 없다. 예를 들면, 여러분의 스케치는 아래와 같이 Serial Monitor에 note[3]에 있는 값을 화면에 나타낼 수 있다:

```
Serial.print(note[3]);
```

배열 안에 있는 값들이 음악 악보이므로, 우리는 ABOT의 피에조스피커로 악보를 연주할 것이다! 여기에 방법이 있다:

```
tone(4, note[3], 500);
```

예제 스케치 - PlayOneNote

여기에 Serial Monitor에 각각의 배열 요소 값을 나타내는 예가 있다. 그리고 이 값들은 것은 ABOT의 피에조스피커로 음악 악보를 연주하는 값으로 사용할 것이다.

√ 입력, 저장 그리고 Arduino에서 PlayOneNote를 업로드하라.
√ 스케치가 업로딩을 마치자 말자 Serial Monitor를 열어라.

√ Serial Monitor에 “note = 1397”이 나타나는지를 확인하라.
√스피커로 음악이 나오는지를 확인하라.
√ note[4]를 이용하는 1568값을 연주하고 출력하도록 스케치를 수정하라.
√ 여러분의 수정된 스케치를 테스트하라.

// Robotics with the BOE Shield – PlayOneNote
// Displays and plays one element from note array.

```
int note[] = {1047, 1147, 1319, 1397, 1568, 1760, 1976, 2093};

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

  Serial.print("note = ");
  Serial.println(note[3]);

  tone(4, note[3], 500);
  delay(750);
}
void loop()
{
}
```

예제 스케치 – PlayNotesWithLoop

많은 응용 프로그램에서 배열 요소에 접근하기 위해 변수를 사용한다. 다음의 스케치 PlayAnotherNote는 index 변수를 선언하고 그리고 index 숫자에 의해 배열 요소를 선택하기 위해 사용한다.

for 루프와 유사하게 index의 값은 자동적으로 증가할 수 있다. 악보를 연주하고 화면에 나타내기 위한 코드는 for루프 안쪽에 있고 index는 배열 요소를 선택하기 위해 사용되어 진다. for 루프를 통한 첫 번째 과정은, index가 0이 되는 것이다. 그래서 note[0]에 저장되는 값은 note[index]가 print 함수 혹은 tone 함수가 어디에 있던지 사용되어질 것이다. 루프를 통한 각 과정과 함께, index는

스케치가 배열 안에 있는 모든 악보들이 화면에 나타나고 연주될 때까지 증가할 것이다.

√ 입력, 저장 그리고 PlayNotesWithLoop를 업로드하라.
√ 스케치가 업로딩 하자마자 Serial Monitor를 열어라.
√ Serial Monitor가 스피커로 나오는 연주가 배열 안에 있는 각 악보들을 나타내는지 확인하라.

```
// Robotics with the BOE Shield - PlayNotesWithLoop
// Displays and plays another element from note array.

int note[] = {1047, 1147, 1319, 1397, 1568, 1760, 1976, 2093};

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

  for(int index = 0; index < 8; index++)
  {
    Serial.print("index = ");
    Serial.println(index);

    Serial.print("note[index] = ");
    Serial.println(note[index]);

    tone(4, note[index], 500);
    delay(750);
  }
}
void loop()
{
}
```

√ 만약 아래에 있는 것처럼 for루프를 변화시킨다면 무슨 일이 일어나겠는가? 해보십시오!!

```
  for(int index = 7; index >= 0; index--);
```

sizeof 함수 이용

좀 더 많거나 적은 악보를 가지는 음악 멜로디를 만들고자 한다. 악보의 정확한 숫자를 연주하기 위해 for 루프를 업데이트하는 것을 잊지 말아야 한다. 아두이노 라이브러리는 이러한 것을 도울 수 있는 sizeof 함수를 가지고 있다. 이것은 바이트 단위로 배열의 크기를 알려 줄 뿐만 아니라 배열 변수 형식의 크기도 알 수 있다 (int 처럼). 여러분의 코드는 배열의 바이트 수를 변수 형식의 바이트 수로 나눌 수 있다. 그 결과는 배열 내에 있는 요소의 수이다.

아래는 이 기술을 이용한 예이다. 변수 elementCount에 note 배열 안에 있는 요소의 수를 저장한다.

```
int note[] = {1047, 1147, 1319, 1397, 1568, 1760, 1976, 2093};
int elementCount = sizeof(note) / sizeof(int);
```

이후, 여러분의 for 루프는 배열 안에 있는 모든 악보를 연주하기 위한 elementCount 변수를 사용할 수 있다. 요소가 추가되든지 삭제되든지 상관없다.

```
for(int index = 0; index < elementCount; index++)
```

√ 입력, 저장 그리고 PlayAllNotesInArray를 업로드하라.
√ 스케치를 업로딩 하자 마자 직렬모니터를 열어라
√ 직렬모니터가 스피커로 나오는 연주가 배열 안에 있는 각 목록들을 나타내는지 확인하라.

```
// Robotics with the BOE Shield - PlayAllNotesInArray
// Uses sizeof to determine number of elements int he array
// and then displays and prints each note value in the sequence.

int note[] = {1047, 1147, 1319, 1397, 1568, 1760, 1976, 2093};

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

  int elementCount = sizeof(note) / sizeof(int);
```

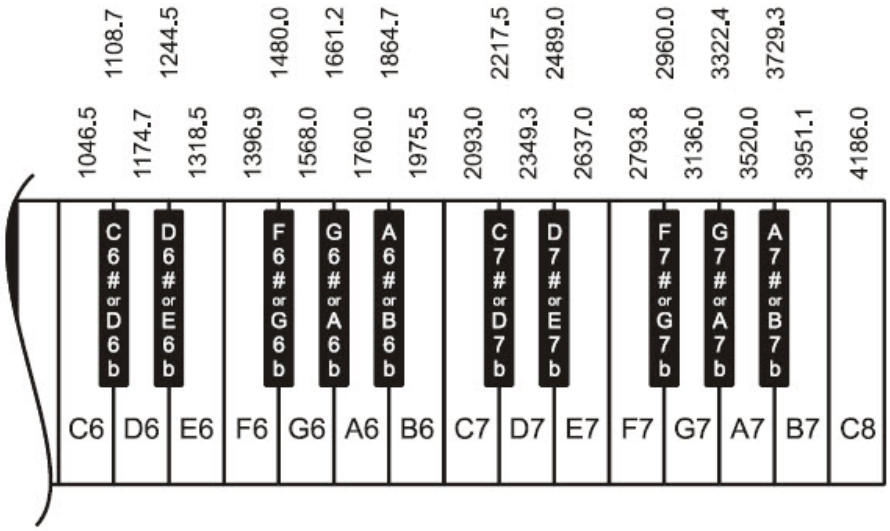
```
  Serial.print("Number of elements in array = ");
  Serial.println(elementCount);

  for(int index = 0; index < elementCount; index++)
  {
    Serial.print("index = ");
    Serial.println(index);

    Serial.print("note[index] = ");
    Serial.println(note[index]);

    tone(4, note[index], 500);
    delay(750);
  }
}
void loop()
{
}
```

여러분 차례 – 목록 추가하기

√ 다음 두 악보, D7과 E7을 추가하고 키보드 그림으로부터 주파수들을 이용한다. 여러분의 배열은 아래와 같이 한줄 이상에 걸쳐 사용 될 수 있다.

```
int note[] = {1047, 1147, 1319, 1397, 1568, 1760,
              1976, 2093, 2349, 2637};
```

√ 만약 여러분이 음악적으로 끌린다면, 매우 짧은 곡조를 연주하는 배열을 쓰도록 노력하라.



<배열을 가지는 주행>

마지막 활동으로부터 maneuver함수를 기억하는가? 여기에 3가지의 배열은 maneuver함수 내에서 각각 파라미터로 사용되는 것들이다. 이와 더불어 이것들은 이 장을 통하여 해왔던 전진-왼쪽-오른쪽-후진-정지 순서로 만들 것이다.

```
//                    Forward     left      right     backward     stop
//             index        0        1         2          3         4
int speedsLeft[]  = {200,      -200,     200,       -200,      0};
int speedsRight[] = {200,       200,    -200,       -200,      0};
int times[]       = {2000,      600,     600,       2000,     -1};
```

스케치는 모든 주행을 실행하기위한 함수들 가운데서 하나의 코드를 사용할 수 있다.

```
// Determine number of elements in sequence list.
int elementCount = sizeof(times) / sizeof(int);
```

```
// Fetch successive elements from each sequence list and feed them
// to maneuver function.
for(int index = 0; index < elementCount; index++)
{
  maneuver(speedsLeft[index], speedsRight[index], times[index]);
}
```

루프를 통하여 순서는 1씩 증가 할 것이다. 그래서 각 maneuver함수 호출과 함께, 각 배열 안에 있는 다음 요소는 maneuver함수에서 파라미터로 전달 될 것이다. 루프의 첫 번째는, index 0, maneuver 함수 호출의 파라미터들은 각 배열의 0번째 요소가 될 것이다. 이것은 다음처럼 maneuver(speedsLeft[0], speedsRight[0], times[0]) 이다. maneuver함수로 전달된 실제적인 값은 maneuver(200, 2000, 2000) 이다. 루프의 두 번째는, index 1 이며 다음과 같을 것이다. maneuver(speedsLeft[1], speedsRight[1], times[1]) 실제적인 값은 maneuver(-200, 200, 2000) 이다.

예제 스케치 - maneuver(-200, 200, 2000).

√ 입력, 저장 그리고 아두이노를 이용하여 ManeuverSequence를 업로드 하라.

√ ABOT이 전진-왼쪽-오른쪽-후진 움직임 그리고 정지가 실행되는지를 확인하라.

```
// Robotics with the BOE Shield - ManeuverSequence
// Move forward, left, right, then backward with an array and the
// maneuver function.
#include <Servo.h>                          // Include servo library

//                  Forward     left    right    backward     stop
//          index       0          1        2          3          4
int speedsLeft[]  = {200,      -200,     200,      -200,        0};
int speedsRight[] = {200,       200,    -200,      -200,        0};
```

```
int times[]          = {2000,     600,     600,        2000,      -1};

Servo servoLeft;                          // Declare left and right servos
Servo servoRight;

void setup()                              // Built-in initialization block
{
  tone(4, 3000, 1000);                    // Play tone for 1 second
  delay(1000);                            // Delay to finish tone

  servoLeft.attach(13);                   // Attach left signal to pin 13
  servoRight.attach(12);                  // Attach right signal to pin 12

  // Determine number of elements in sequence list.
  int elementCount = sizeof(times) / sizeof(int);

  // Fetch successive elements from each sequence list and feed them
  // to maneuver function.
  for(int index = 0; index < elementCount; index++)
  {
    maneuver(speedsLeft[index], speedsRight[index], times[index]);
  }
}
void loop()                               // Main loop auto-repeats
{                                         // Empty, nothing needs repeating
}
void maneuver(int speedLeft, int speedRight, int msTime)
{
//speedLeft, speedRight ranges:Backward Linear Stop Linear    Forward
//                              -200       -100.....0.....100         200
  servoLeft.writeMicroseconds(1500+speedLeft); // Set Left servo speed
  servoRight.writeMicroseconds(1500-speedRight);//Set right servo speed
  if(msTime==-1)                                  // if msTime = -1
  {
```

```
    servoLeft.detach();                    // Stop servo signals
    servoRight.detach();
  }
  delay(msTime);                           // Delay for msTime
}
```

여러분의 ABOT이 움직이는 순서가 전진-왼쪽-오른쪽-후진-정지와 유사하게 실행되었는가? 지금까지의 작업이 지루하지는 않은가요? 여러분의 ABOT이 그 밖의 다른 할 수 있는 일들이나, 여러분 자신이 만든 루틴으로 안무하는 것을 보기를 원하는가?

여러분 차례 - 리스트에 주행을 추가

다음은 여러분이 보다 긴 리스트를 시도해볼 수 있는 예제이다. 이것은 전진-왼쪽-오른쪽-후진의 순서로 진행한 뒤 4가지의 축회전을 실행하는 것이다. 이 예제에서, index가 4일 경우, 이것은 각 배열의 두 번째 줄의 첫 번째를 사용 할 것이다. 마찬가지로 index가 5일 때, 이것은 각 배열의 두 번째 줄에 있는 두 번째 수를 사용 할 것이다. { }안에 있는 각 요소는 콤마를 이용하여 구분한다. 그리고 이것은 한 줄로 표현해도 되며 혹은 여러 줄에 걸쳐 표현해도 전혀 문제가 되지 않는다.

```
int speedsLeft[]  = {200,    -200,    200,      -200,
                       0,     200,   -200,        0,       0};
int speedsRight[] = {200,     200,   -200,      -200,
                     200,       0,      0,      -200,       0};
int times[]       = {2000,    600,    600,      2000,
                     1000,    1000,   1000,      1000,     -1};
```

√ ManeuverSequence를 ManeuverSequenceExpanded로 저장하라.
√ 배열을 위에서 선언한 9개의 요소를 가지는 예제처럼 만들어라.
√ 수정된 스케치를 실행하고 전진-왼쪽-오른쪽-후진의 순서로 수행 한 후 축회전을 실행하는지 확인하라.

<문자 배열과 switch-case>

이 활동의 마지막 예제는 배열에 문자 목록을 이용하여 주행을 수행하는 스케치이다. 각 문자는 200ms 실행 시간을 가지는 어떤 주행을 나타내는 것이다. 실행 시간이 고정되어 있으므로, 이것은 유연성을 가지고 있지는 않다. 그러나 이것은 주행의 빠른 시퀀스를 만드는 것으로 간단하다.

f = forward b = backward l = left r = right s = stop
f = 전진 b = 후진 l = 왼쪽 r = 오른쪽 s = 정지

문자배열 이용은 콤마로 분리된 배열의 목록을 사용하지 않는다. 대신, 문자열을 사용한다. 다음은 문자배열 안에 이전에 했던 전진-왼쪽-오른쪽-후진-정지의 시퀀스와 같은 예제이다.

```
char maneuvers[] = "ffffffffffllrrrbbbbbbbbbbs";
```

문자배열은 10개의 f 문자를 가진다. 각 문자가 200ms의 실행 시간을 나타내므로, ABOT이 2초 동안 전진을 할 것이다. 다음은 3개의 l 문자이므로 600ms 동안 왼쪽 회전을 할 것이다. 3개의 r 문자는 오른쪽 회전을 만들고, 뒤에 오는 b 문자는 후진으로 가도록 만들 것이다. 그리고 문자 s는 이 시퀀스를 완전하게 멈추도록 한다.

예제 스케치: ControlWithCharacters

√ 입력, 저장 그리고 아두이노에서 ControlWithCharacters를 업로드하라.

√ ABOT이 전진-외쪽-오른쪽-후진 움직임 그리고 정지를 실행하는지 확인하라.

```
// Robotics with the BOE Shield - ControlWithCharacters
// Move forward, left, right, then backward for testing and tuning.

#include <Servo.h>                    // Include servo library

char maneuvers[] = "ffffffffffllrrrbbbbbbbbbbs";

Servo servoLeft;                      // Declare left and right servos
```

```
Servo servoRight;

void setup()                          // Built-in initialization block
{
  tone(4, 3000, 1000);                // Play tone for 1 second
  delay(1000);                        // Delay to finish tone

  servoLeft.attach(13);               // Attach left signal to P13
  servoRight.attach(12);              // Attach right signal to P12

  // Parse maneuvers and feed each successive character to the go
function.
  int index = 0;
  do
  {
    go(maneuvers[index]);
  } while(maneuvers[index++] != 's');}

void loop()                           // Main loop auto-repeats
{                                     // Empty, nothing needs repeating
}
void go(char c)                       // go function
{
  switch(c)                           // Switch to code based on c
  {
    case 'f':                                 // c contains 'f'
      servoLeft.writeMicroseconds(1700);  // Full speed forward
      servoRight.writeMicroseconds(1300);
      break;
    case 'b':                                 // c contains 'b'
      servoLeft.writeMicroseconds(1300);    // Full speed backward
      servoRight.writeMicroseconds(1700);
      break;
    case 'l':                                  // c contains 'l'
```

```
        servoLeft.writeMicroseconds(1300);       // Rotate left in place
        servoRight.writeMicroseconds(1300);
        break;
      case 'r':                                  // c contains 'r'
        servoLeft.writeMicroseconds(1700);       // Rotate right in place
        servoRight.writeMicroseconds(1700);
        break;
      case 's':                                  // c contains 's'
        servoLeft.writeMicroseconds(1500);       // Stop
        servoRight.writeMicroseconds(1500);
        break;
    }
    delay(200);                                  // Execute for 0.2
seconds
}
```

√ 이 배열을 시도하라 ABOT이 무엇을 하는지 예측할 수 있겠는가?

```
char maneuvers[] = "ffffffffffllllrrrrrrllllbbbbbbbbbbs";
```

문자 maneuvers 배열 변수와 사용 초기화 후, 이러한 것들은 배열로부터 문자들을 가져오고 go 함수로 이들 문자열을 전달한다.(나중에 설명)

```
  int index = 0;
  do
  {
    go(maneuvers[index]);
  } while(maneuvers[index++] != 's');
```

첫 번째, index는 초기값으로 0이 선언되었고 do-while루프가 사용되었다. 일반적인 while 루프와 마찬가지로 do-while 루프는 코드 블록 내에 있는 명령 줄들을 반복적으로 실행한다. 그러나 블록을 시작한 후 나타나는 while부분까지 최소한 한번은 항상 실행한다. 루프를 통한 시점에, go(maneuvers[index])는 maneuvers[index]에 있는 문자를 전달한다. index++에 있는 ++는 루프를 통한 다음 시간 동안 index 변수에 1를 추가한다 - 이것은 후위 증가 연산자라는 것

을 명심하자. 이것은 “maneuvers 배열이 ‘s’와 같지 않을 동안 계속된다.

이제 go 함수를 살펴보자. 이것은 c 파라미터를 통해 전달된 각 문자를 받고, switch/case문을 이용하여 그때 상황에 따라 그때 그때에 따라 평가한다. maneuver 문자 배열 안에 있는 각각의 5개의 문자에 대해, go함수에 의해 받은 문자는 switch(c) 블록 안에서 일치하는지를 묻는 case문이 있다.

만약 go 함수 호출이 c파라미터에 대해 f 문자를 전달한다면, f인 경우의 코드는 전속력-전방처럼 실행될 것이다. 만약 이것이 b를 전달한다면, 전속력-후진이 실행 될 것이다. 각각의 경우에 이어서 break는 switch 블록을 빠져나오는 것이고 스케치는 다음 명령어 delay(200)으로 이동한다. 그래서 각 go에 대한 호출은 200ms 주행을 결과로 나타낸다. 각 case문의 끝에 break가 없다면, 스케치는 다음 case문을 실행할 것이며, 이는 주행에서 요구되지 않았던 결과를 초래할 것이다.

```
 void go(char c)                          // go function
{
  switch(c)                              // Switch to based on c
  {
    case 'f':                             // c contains 'f'
      servoLeft.writeMicroseconds(1700);  // Full speed forward
      servoRight.writeMicroseconds(1300);
      break;
    case 'b':                             // c contains 'b'
      servoLeft.writeMicroseconds(1300);  // Full speed backward
      servoRight.writeMicroseconds(1700);
      break;
    case 'l':                             // c contains 'l'
      servoLeft.writeMicroseconds(1300);  // Rotate left in place
      servoRight.writeMicroseconds(1300);
      break;
    case 'r':                             // c contains 'r'
      servoLeft.writeMicroseconds(1700);  // Rotate right in place
      servoRight.writeMicroseconds(1700);
      break;
```

```
    case 's':                                   // c contains 's'
      servoLeft.writeMicroseconds(1500);        // Stop
      servoRight.writeMicroseconds(1500);
      break;
  }
  delay(200);                                    // Execute for 0.2 s
```

여러분 차례 – 배열 요소와 case문 추가하기

√ 전방으로 1/2 속력의 경우를 추가하자. 문자는 'h'를 사용하고 servo를 선형으로 움직이기 위해 1400에서 1600사이의 값을 가진다.

```
case 'h':                                    // c contains 'h'
  servoLeft.writeMicroseconds(1550);        // Half speed forward
  servoRight.writeMicroseconds(1450);
  break;
```

√ 10을 추가하거나 혹은 여러분의 maneuver 문자 배열에 h문자를 추가하라. 기능을 보이기 위해서 스케치에는 s문자의 왼쪽에 추가되어야 한다는 것을 명심하라.

√ 약간의 실험, 그리고 축회전-후진-왼쪽과 같은 다른 주행을 위한 다른 case문 추가한다. 그런 다음, 여러분의 배열 문자열에 새로운 주행을 위한 임의의 문자를 추가 하라. 주행을 연속적으로 만들기 위해 이것이 얼마나 편리한 방법인지를 알겠습니까?

제4장 종합

이 장은 로봇 주행에 대한 모든 것, 즉 여러 가지의 다른 프로그래밍 접근으로 실험을 그리고 로봇과 공학 기술의 적용을 하였다.

프로그래밍

•주행의 단순화 - 자주 사용되는 주행 코드는 사용자 함수를 만들어 주행을 단순화 한다. 그리고 어떻게 함수를 호출하는지를 설명.

• 주행 코드 내에서 스텝 증가를 가지는 for루프가 어떻게 카운터 되는지를 설명.

• 무슨 파라미터들이 있는가? 그리고 함수를 어떻게 만드는지? 그리고 만들어진 함수를 어떻게 호출하는지를 설명.

• 지역 변수는 함수 선언 안에서 파라미터로서 어떻게 생성되는지를 설명.

• 아두이노 언어의 int와 문자 배열을 어떻게 선언, 초기화 그리고 사용하는지를 설명. 아두이노 라이브러리의 sizeof함수의 사용 및 장점 설명.

• 프로그램 흐름을 어떻게 제어하는지 설명, do-while문과 switch/case문 사용.

• 조건 루프에서 후위 증가 연산자(++)를 어떻게 사용하는지를 설명.

• 루프의 조건에서 “같지않다”의 비교 연산자(!=)를 어떻게 사용하는지를 설명.

• 피에조 스피커로 연속된 악보를 연주하는 스케치하기.

• 같은 주행 순서들을 수행하기위해 여러 가지의 다른 프로그램 방법을 이용하여 스케치하기.

로봇 기술

• 기본적인 롤링-로봇주행들이 바퀴 속도와 방향 제어에 의해 어떻게 만들어지는지를 설명.

• 점진적 회전, 축회전(pivot-turns) 그리고 턴에서의 회전(rotating-in-place turns)의 차이가 무엇인지 설명. 그리고 이를 위해 바퀴의 속도와 방향 조합을 만드는 방법 설명.

• 속도 램핑이 무엇인지 설명. 여러분의 로봇을 부드럽게 움직이기 위해 어떻게 사용하는지 설명. 그리고 램핑이 여러분의 BOE Shield-Bot에 어떤 장점이 되는지를 설명.

• 초보 단계(entry-level) 로봇 주행의 맥락에 있어서, 추측항법(dead reckoning)이 무엇인지 설명.

• 사전에 결정된 거리 혹은 특별한 각도에서 회전을 하는 BOE Shield-Bot 경

로를 만들기 위해 로봇 주행 실행 시간(run-time)제어 설명.
• 직선 주행을 위한 servo 속도를 조정하는 것에 의해 하드웨어 변화를 보상하는 것을 설명.

엔지니어링 기술

• 시스템 입력과 출력 사이의 원인과 효과 관계의 특징을 나타내는 간단한 형식에 대해 일관성을 유도하기위한 관측과 측정을 만드는 것.
• 하드웨어 조정과 소프트웨어 조정 사이의 차이
• 반복적인 작업이 무엇인지, 그리고 테스트를 하는 소프트웨어 조정을 위해 사용하는 것은 무언지.

제4장 도전

질문

1. ABOT이 앞으로 가도록 만들기 위해 왼쪽 바퀴를 무슨 방향으로 해야 하는가? 그리고 오른쪽 바퀴는 무슨 방향으로 해야 하는가?

2. ABOT이 왼쪽으로 하나의 바퀴를 축으로 회전하고자 할 때, 어느 쪽 바퀴가 해당되는가? ABOT이 축회전 왼쪽을 만들기 위해 필요한 코드는 무엇인가?

3. 만약 ABOT이 똑바로 앞쪽으로 진행하도록 만들기 위해 스케치를 실행 할 때 한쪽으로 서서히 방향을 바꾼다면, 이것을 바로잡기 위해 어떻게 할 것인가? 조정을 하기 위해 필요한 명령어는 무엇인가? 그리고 무슨 종류의 조정을 여러분이 만들 수 있는가?

4. 만약 ABOT이 11in/s로 움직인다면, 36인치를 주행하는데 얼마나 시간이 걸리겠는가?

5. 왜 servo가 램프하는 for 루프에 delay(20)이 필요한가?

6. 목록에 있는 여러 가지 값들을 저장하기위해 무슨 종류의 변수를 사용하는가?

7. 목록으로부터 검색 값들을 사용하는 것은 무슨 loop가 적당한가?

8. 특별한 변수를 선택하기위해 사용할 수 있는 문장은 무엇인가? 그리고 그때 그때마다 바이어스를 평가하기위해 사용할 수 있는 것은 무엇인가? 각 case문에 대해 다른 코드 블록을 실행하기위해 사용할 수 있는 것은 무엇인가?

9. do-loop에 확장할 수 있는 조건은 무엇인가?

연습문제

1. 2.5초 동안 전속력으로 ABOT이 뒤로 가도록 만들기 위한 루틴을 적어라.

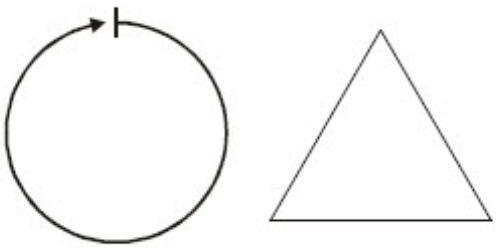
2. 여러분의 servo들을 테스트하고 오른쪽-회전을 180도 만들기 위해 1.2초간 소요되는지를 확인하라. 이러한 정보를 이용하여 ABOT이 30도, 45도 그리고 60도 회전을 하도록 루틴을 적으라.

3. ABOT이 똑바로 앞쪽으로, 그런 다음 안쪽과 바깥쪽으로 축회전하는 램프, 그런 다음 똑바로 계속 전진하는 루틴을 적어라.

프로젝트

1. 표2-2의 3열을 완성할 때이다. 이것을 하기위해 1열로부터 각 쌍의 값들을 이용하는 ForwardThreeSeconds 스케치 안에 있는 writeMicroseconds함수 내의 us 인수를 수정하라. 3열안에 있는 각 쌍에 대한 여러분의 ABOT의 결과 행동을 기록하라. 일단 완성되었다면, 이 표는 여러분 자신의 사용자 ABOT 주행들을 디자인할 때 지침표로서 사용될 것이다.

2. 아래 그림은 두 개의 간단한 코스를 나타낸다. 각 그림을 따라 여러분의 ABOT이 주행할 수 있도록 프로그램을 스케치하라. 직선거리(삼각형의 변과 원의 지름)는 10야드 혹은 1미터라 가정하자.



제4장 해답

질문 해

1. 왼쪽 바퀴는 반시계방향, 오른쪽 바퀴는 시계방향

2. 오른쪽 바퀴는 시계방향으로 회전하면서(전진), 그리고 왼쪽 바퀴는 움직이지 않는다.

```
servoLeft.writeMicroseconds(1500);
servoRight.writeMicroseconds(1300);
```

3. 왼쪽으로 방향 전환을 하기위해 오른쪽 바퀴를 천천히 움직인다. 그리고 오른쪽으로 방향 전환을 하기위해 왼쪽 바퀴를 천천히 움직인다. servo의 writeMicroseconds에 있는 us 파라미터를 1500에 가깝게 변화시킴으로써 바퀴의 속도를 늦춘다. 선형적인 속도 제어 범위 1400-1600의 끝에서 적당히 시작하라. 10씩 증가하여 1500까지 변하면서 점진적으로 움직일 것이다. 그리고 너무 지나치면 작은 단위 증가로 돌아가라.

4. 아래 데이터가 주어져 있다. 36 인치를 주행하는데 3727ms가 소요된다.
ABOT speed = 11 in/s

ABOT distance = 36 in/s

time = (ABOT distance / ABOT speed) * (1000 ms / s)

= (36 / 11) * (1000)

= 3.727272…s * 1000 ms/s

≈ 3727 ms

5. 루프의 각 반복 사이에 있는 20ms 지연을 없애면, ABOT이 즉시 전속력이 되는 것처럼 0부터 100까지 빠르게 램프할 것이다.

6. 배열

7. for 루프와 do-while루프가 이장에서 사용된 예제들이다.

8. switch/case

9. do{...}루프 while(조건)

연습 해

1.Solution:

```
servoLeft.writeMicroseconds(1300);
servoRight.writeMicroseconds(1700);
delay(2500);
```

2.Solution:

```
// 30/180 = 1/6, so use 1200/6 = 200
servoLeft.writeMicroseconds(1700);
servoRight.writeMicroseconds(1700);
delay(200);

// alternate approach
servoLeft.writeMicroseconds(1700);
servoRight.writeMicroseconds(1700);
delay(1200 * 30 / 180);

// 45/180 = 1/4, so use 1200/4 = 300
servoLeft.writeMicroseconds(1700);
servoRight.writeMicroseconds(1700);
delay(300);

// 60/180 = 1/3, so use 1200/3 = 400
```

```
servoLeft.writeMicroseconds(1700);
servoRight.writeMicroseconds(1700);
delay(400);
```

3.Solution:

```
 // forward 1 second
servoLeft.writeMicroseconds(1700);
servoRight.writeMicroseconds(1700);
delay(1000);

// ramp into pivot
for(int speed = 0; speed <= 100; speed+=2)
{
  servoLeft.writeMicroseconds(1500);
  servoRight.writeMicroseconds(1500+speed);
  delay(20);
};

// ramp out of pivot
for(int speed = 100; speed >= 0; speed-=2)
{
  servoLeft.writeMicroseconds(1500);
  servoRight.writeMicroseconds(1500+speed);
  delay(20);
}

// forward again
servoLeft.writeMicroseconds(1700);
servoRight.writeMicroseconds(1700);
delay(1000);
```

프로젝트 해

1. 해 (표가 여러분이 출력한 것 보다 약간 다르게 보일것이지만)

Servo ports connected to:		Description	Behavior
Pin 13	Pin 12		
1700	1300	Full Speed: pin 13 CCW, Pin 12 CW	Forward
1300	1700	Full Speed: Pin 13 CW, Pin 12 CCW	Backward
1700	1700	Full Speed: Pin 13 CCW, Pin 12 CCW	Right rotate
1300	1300	Full Speed: Pin 13 CW, Pin 12 CW	Left rotate
1500	1700	Pin 13 Stopped, Pin 12 CCW Full speed	Pivot back left
1300	750	Pin 13 CW Full Speed, P12 Stopped	Pivot back right
1500	1500	Pin 13 Stopped, Pin 12 Stopped	Stopped
1520	1480	Pin 13 CCW Slow, Pin 12 CW Slow	Forward slow
1540	1460	Pin 13 CCW Med, Pin 12 CW Med	Forward medium
1700	1450	Pin 13 CCW Full Speed, Pin 12 CW Medium	Veer right
1550	1300	Pin 13 CCW Medium, Pin 12 CW Full Speed	Veer left

2. 원은 오른쪽으로 계속 방향을 바꾸는 방법으로 실행될 수 있다. 시도와 실수, 그리고 야드와 미터 단위는 여러분이 writeMicroseconds(us)에 대한 올바른 us파라미터와 delay(ms)위한 올바른 ms 파라미터는 사용하도록 도울 것이다. 아래는 servo 과 배터리의 쌍의 특별한 관계 일을 나타내는 해 이다. 여러분의 값들이 Circle 스케치 안에 있는 어떤 무엇가로부터 매우 변화할 것이다.

삼각형에 대해, 첫 번째, 질문 4에서 처럼, 1미터 1야드 직선 라인에 대해 ms로 요구된 주행 계산, 여러분의 ABOT과 특별한 면에 대하여 미세 조정을 한다. ABOT은 1미터 혹은 1야드 앞으로 움직여야 한다. 그리고 120도 회전을 한다. 삼각형의 세면을 따라 세 번 위의 동작을 반복한다. 여러분은 정밀한 120도회전을 얻기 위해 프로그램내의 “Turn left 120”루틴에서 delay 호출함수를 조정해야한다.

원 스케치:

```
// Robotics with the BOE Shield – Chapter 4, project 2 – Circle
```

```
// ABOT navigates a circle of  1 yard diameter.

#include <Servo.h>                      // Include servo library

Servo servoLeft;                        // Declare left and right servos
Servo servoRight;

void setup()                            // Built-in initialization block
{
  tone(4, 3000, 1000);                  // Play tone for 1 second
  delay(1000);                          // Delay to finish tone

  servoLeft.attach(13);                 // Attach left signal to pin 13
  servoRight.attach(12);                // Attach right signal to pin 12

  // Arc to the right
  servoLeft.writeMicroseconds(1600);  // Left wheel counterclockwise
  servoRight.writeMicroseconds(1438); // Right wheel clockwise slower
  delay(25500);                         // ...for 25.5 seconds

  servoLeft.detach();                   // Stop sending servo signals
  servoRight.detach();
}

void loop()                             // Main loop auto-repeats
{                                       // Nothing needs repeating
}
```

삼각형 스케치

```
// Robotics with the BOE Shield - Chapter 4, project 2 - Triangle
// ABOT navigates a triangle with 1 yard sides and 120
// degree angles.  Go straight 1 yard, turn 120 degrees, repeat 3 times
```

```
#include <Servo.h>                      // Include servo library

Servo servoLeft;                        // Declare left and right servos
Servo servoRight;

void setup()                            // Built-in initialization block
{
  tone(4, 3000, 1000);                  // Play tone for 1 second
  delay(1000);                          // Delay to finish tone

  servoLeft.attach(13);                 // Attach left signal to pin 13
  servoRight.attach(12);                // Attach right signal to pin 12

  for(int index = 1; index <= 3; index++)
  {
    // Full speed forward
    servoLeft.writeMicroseconds(1700);  // Left wheel counterclockwise
    servoRight.writeMicroseconds(1300); // Right wheel clockwise slower
    delay(5500);                        // ...for 5.5 seconds

    // Turn left 120 degrees
    servoLeft.writeMicroseconds(1300);  // Left wheel counterclockwise
    servoRight.writeMicroseconds(1300); // Right wheel clockwise slower
    delay(700);
  }
  servoLeft.detach();                   // Stop sending servo signals
  servoRight.detach();
}

void loop()                             // Main loop auto-repeats
{                                       // Nothing needs repeating
}
```