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

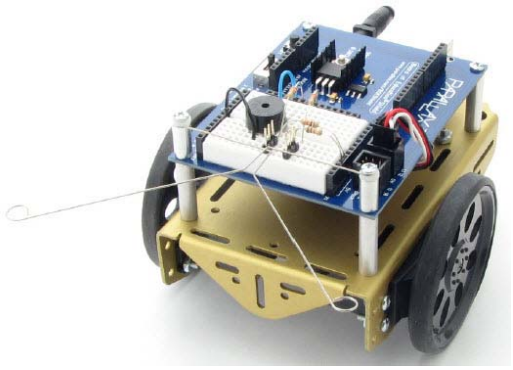
제5장 더듬이를 이용한 촉각센서 주행

촉각 스위치들을 범퍼 스위치 혹은 터치 스위치라고 부른다. 그리고 이것들은 로봇에서 많이 이용된다. 물체를 들어 올리고 다른 컨베이어 벨트에 옮기기 위해 프로그램된 로봇은 물체를 감지하기 위해 촉각 스위치에 의존한다. 자동화된 공장 라인은 물체를 카운터하기 위해 그리고 제조 공정에서 임의의 스텝에 대한 부품들을 일렬로 세우기 위해 촉각 스위치를 사용할 것이다. 각각의 경우, 스위치들은 프로그램된 출력의 어떤 형식을 트리거하여 입력 값으로 제공할 것이다. 입력 값들은 스위치가 눌러졌는지 혹은 그렇지 않은지에 따라 다른 동작을 하는 장비 공정에서 전기적으로 모니터링된다.

이 장에서, 여러분은 더듬이로 불리는 촉각 스위치를 ABOT에 장착하고 테스트한다. 여러분은 이러한 스위치들의 상태를 모니터하기 위해 그리고 이것이 장애물과 만날 때, 무엇을 할 것인가를 결정하기 위해 ABOT을 프로그램할 것이다. 최종 결과는 터치에 의한 자율 주행이 될 것이다.

촉각 주행

더듬이 스위치는 고양이의 많은 수염처럼, 이리저리 돌아다닐 때 터치를 통하여 로봇의 주변을 센싱하는 능력을 제공한다. 이 장의 활동들은 그들 스스로 더듬이를 이용하는 것이지만, 이 장의 후반부에서 배우게 될 다른 센서들과 결합될 수도 있다.



√ **다운로드 5장 아두이노 코드**(인터넷에서 다운로드)

√ 아래 링크를 따라서 출발합시다!!!

- 실습1: 더듬이 만들기 그리고 테스트
- 실습2: 더듬이 필드 테스트
- 실습3: 더듬이가 부착된 주행
- 실습4: 코너 탈출을 위한 인공 지능
- 실습5: 종합

실습1: 더듬이 만들기와 테스트

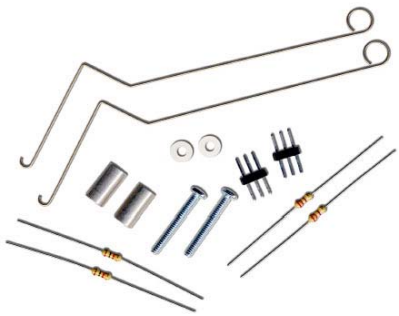
시스템 테스트를 기억하십니까? 먼저, 우리는 더듬이 회로를 만들 것이고 주행 스케치에서 이것을 이용하기 전에 입력 상태를 점검하기 위해 코드를 작성할 것이다.

더듬이 회로와 조립

√ 부품 목록에 있는 더듬이 하드웨어를 모은다.
√ 보드와 서보(servo)로부터 전원은 차단한다.

부품 리스트

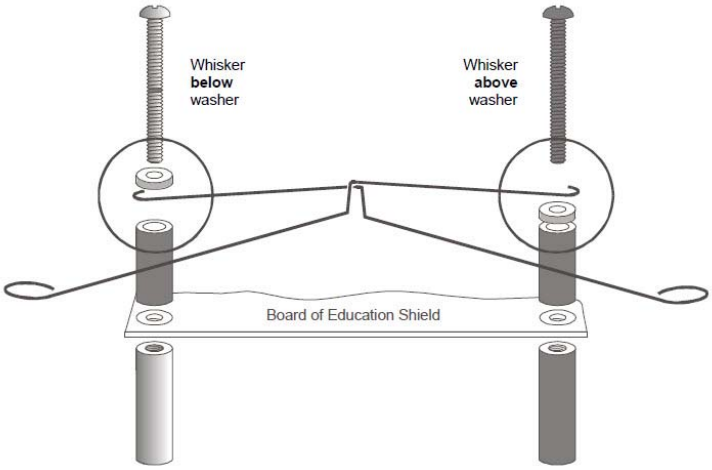
(2) 더듬이 와이어
(2) 7/8″ 팬 헤드 4-40 필립스 나사
(2) ½″ 둥근 스페이스
(2) 나일론 와셔, 사이즈 #4
(2) 3-pin m/m 헤드
(2) 저항 220 Ω (빨강-빨강-갈색)
(2) 저항 10 kΩ (갈색-흑색-오렌지색)



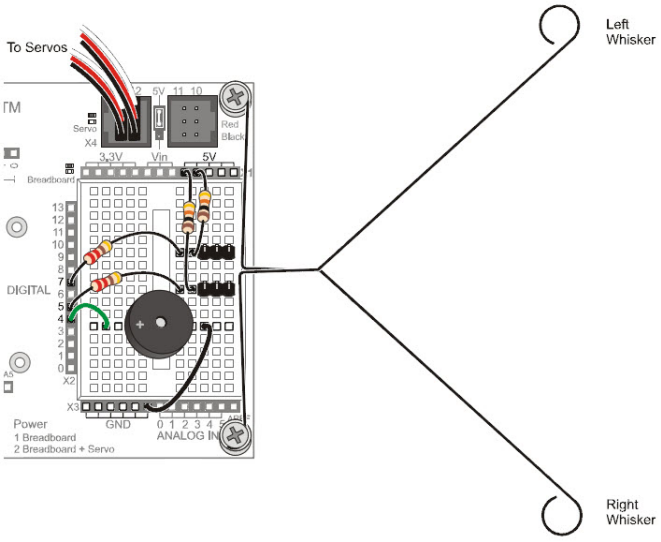
더듬이 만들기

√ 서보(servo) 주행을 테스트 하는 동안 신호를 모니터할 때 사용되었던 LED 회로를 제거하시오.
√ 앞쪽 받침대 위에 보드를 잡고 있는 두 개의 나사를 제거하시오.
√ 나일론 와셔와 각각에 ½″ 둥근 스페이스를 7/8″ 나사로 조립하시오.

√ 여러분의 보드에 있는 구멍과 아래 받침대 안쪽을 통하여 스크류를 부착합니다. 그러나 아직 너무 조이지는 마시오.
√ 더듬이 와이어의 끝에 있는 갈고리 모양을 둥근 나사에 밀어 넣는다. 이때 하나는 와셔 위에 그리고 다른 하나는 와셔 아래에 위치시키고, 서로 닿지 않도록 이것들을 교차시키시오.
√ 받침대 안에 있는 나사를 꽉 조이시오.



√ 3-핀 헤드에 해당하는 디지털 핀 5와 7에 연결하기위해 220 Ω저항을 사용하시오.
√ 각 3-핀 헤드에 5V 전원을 연결하기위해 10 kΩ 저항을 사용하시오.
√ 각 더듬이를 조정하고 브레드보드위에 3-핀 헤드와 닿지 않게 가깝게 합니다. 약 3mm거리 오른쪽입니다.



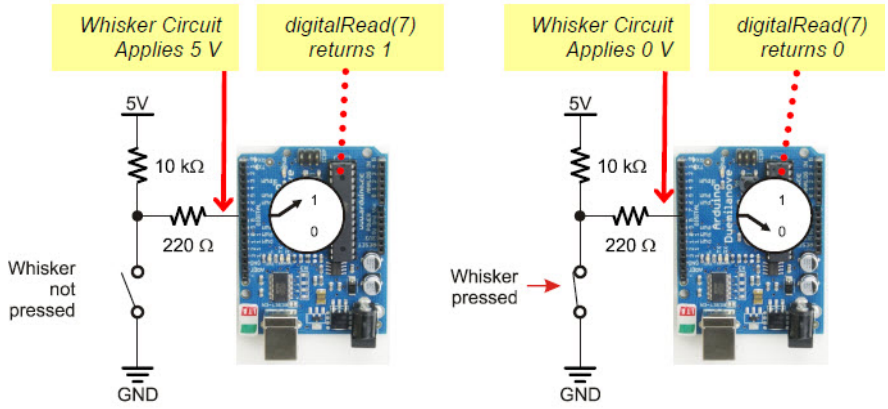
<어떻게 더듬이 스위치가 동작하는가?>

보드의 바깥쪽 모서리에 평평한 구멍이 Vss와 연결되어 있기 때문에 더듬이들이 접지(Vss)와 연결된다. 금속 받침대와 나사는 각각 더듬이로 전기적 연결 수단을 제공한다.

각 더듬이는 디지털 I/O에 연결되어 있으므로, 아두이노는 각 회로에 적용된 전압(0V 혹은 5V)을 감지하기 위해 프로그램될 수 있다. 먼저, pinMode(pin, mode)함수에서 각 핀을 입력 모드로 세팅한다. 그런 다음, digitalRead(pin)함수를 이용하여 핀의 HIGH 혹은 LOW 상태를 감지한다.

아래 그림을 살펴보자. 왼쪽의 r 회로는 더듬이가 눌러지지 않았을 때 5V를 적용한다. 그래서 digitalRead(7)은 1(HIGH) 값을 되돌려 준다. 오른쪽 회로는 더듬이가 눌러졌을 때 0V를 적용한다. 그래서digitalRead(7) 함수는 0(LOW)를 되돌려 준다.

가장 중요한 것은, 여러분 스케치는 변수 wLeft와 wRight가 가지고 있는 리턴 값을 저장할 수 있다. 그리고 이것은 행동을 감지하거나 결정을 내리기 위해 사용된다. 다음 예제 스케치는 이 방법을 증명할 것이다.

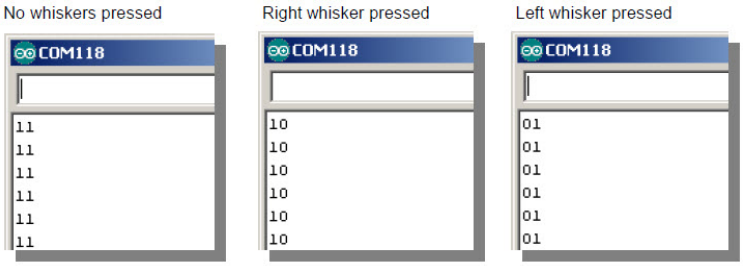


스위치 용어: 각 더듬이는 기계적인 확장이고 정상열림 상태(off until pressed) 순시(on only while pressed) 단극(one set of electrical contact points) 단역(only one position conducts) 스위치의 전기적 접지 연결이다.

<더듬이 테스트>

다음 스케치는 digitalRead(7)과 digitalRead(5)에 의해 반환되는 이진 값을 더듬이가 적당하게 함수화 하는지를 확실하게 만들기 위해 테스트한다. 이 방법은 여러분이 브레드보드 위에 3-핀 헤드에 각 더듬이를 누를 수 있다. 그리고 만약 아두이노의 디지털 핀이 전기적인 접촉을 센싱하는지를 점검하시오.

더듬이가 어느 쪽의 3-핀 헤드에 대해서도 눌러지지 않았을 때, 여러분은 각각의 더듬이에 대해 여러분 직렬모니터(Serial Monitor) 두 개의 열에 1이 나타나는 것을 기대할 수 있다. 만약 여러분이 오른쪽 더듬이를 누른다면 오른쪽 열이 0으로 나타날 것이다. 그리고 화면은 10를 나타낸다. 만약 여러분이 왼쪽 더듬이를 누르면, 왼쪽 열이 0을 나타낼 것이고 화면은 01를 나타낼 것이다. 물론 만약 여러분이 양쪽 더듬이를 모두를 누르면, 00을 나타낼 것이다.



Active-low 출력
더듬이 회로는 더듬이가 눌러졌을 때(작동) low 신호를 그리고 눌러지지 않았을 때 high신호를 전송하는 active-low 출력으로 연결된다. digitalRead가 low 신호에 대해 0를 그리고 high 신호에 대해 1를 리턴하므로, 0는 더듬이가 눌러졌을 때 여러분의 스케치는 무엇인가 일하는 것이고, 1은 더듬이가 눌러지지 않았다는 것을 말한다.

√ 여러분의 아두이노에 입력, 저장 그리고 업로드 TestWhiskers.
√ USB 케이블을 다시 연결 그리고 position 1에 3-position 스위치를 세트한다.
√ 스케치 업로딩을 끝나자마자 직렬모니터를 연다.
√ 연결된 USB 케이블을 분리한다. 그래서 아두이노는 시리얼 메시지를 직렬 모니터(Serial Monitor)로 보낸다.

예제 스케치 : DisplayWhiskerStates

```
/*
* Robotics with the BOE Shield - DisplayWhiskerStates
* Display left and right whisker states in Serial Monitor.
* 1 indicates no contact; 0 indicates contact.
*/
```

```
void setup()                          // Built-in initialization block
{
  tone(4, 3000, 1000);                // Play tone for 1 second
  delay(1000);                        // Delay to finish tone
  pinMode(7, INPUT);                  // Set right whisker pin to input
  pinMode(5, INPUT);                  // Set left whisker pin to input

  Serial.begin(9600);                 // Set data rate to 9600 bps
}

void loop()                           // Main loop auto-repeats
{
  byte wLeft = digitalRead(5);        // Copy left result to wLeft
```

```
  byte wRight = digitalRead(7);          // Copy right result to wRight

  Serial.print(wLeft);                   // Display left whisker state
  Serial.println(wRight);                // Display right whisker state

  delay(50);                             // Pause for 50 ms
}
```

직렬모니터에 나타난 숫자를 살펴보시오. 어떤 더듬이도 눌러지지 않으면, 모니터는 11을 표시해야 하고, 5V가 두 개 디지털 입력(5번과 7번)으로 공급됩니다. 오른 쪽 더듬이가 핀 세 개인 헤더(three-pin header)로 접촉시키고, 직렬모니터에 표시된 값을 기록하시오. 그 값은 10이어야 합니다.

√ 직렬모니터(Serial Monitor)에 나타나는 값들을 보시오. 더듬이가 눌러지지 않았을 때, 양쪽 디지털 입력(5와 7)이 5V가 적용된 11이 나타날 것입니다.
√ 3-핀 헤드 안쪽으로 오른쪽 더듬이를 누르고 직렬모니터에 나타난 값을 기록하시오. 이것은 아마 10 일 것입니다.
√ 오른쪽 더듬이를 누른 것을 놓고 3-pin 헤드 안쪽으로 왼쪽 더듬이를 누른 후, 직렬모니터에 나타난 값을 기록하시오. 이것은 아마 01 일 것입니다.
√ 3-pin 헤드에 대해 양쪽 더듬이를 누르면, 직렬모니터에 00이 나타날 것입니다.
√ 만약 더듬이가 이러한 테스트를 모두 통과 했다면, 여러분은 움직일 준비가 되었다는 것입니다. 만약 그렇지 않다면 여러분의 스케치와 회로에 대한 에러를 점검하시오.

이러한 과정은 중요합니다.
진지하게, 여러분은 다음 단계를 진행하기 전에 이러한 테스트를 통과하도록 여러분의 회로와 코드를 확실하게 만들어야 합니다. 이 장의 예제 나머지 부분은 더듬이가 올바르게 작동하는지에 달려 있습니다. 만약 여러분이 테스트를 하지 않았고 에러를 올바르게 고치지 않았다면, 예제의 나머지 부분은 올바르게 작동하지 않을 것입니다.

<DisplayWhiskerStates가 어떻게 일하는가? >

setup 함수안에서, pinMode(7, 입력)과 pinMode(5,입력)는 입력으로 디지털 핀 7과 5로 세트합니다. 그래서 그들은 더듬이 회로에 의해 적용된 전압을 모니터링할 수 있습니다.

```
pinMode(7, INPUT);          // Set right whisker pin to input
pinMode(5, INPUT);          // Set left whisker pin to input
```

loop함수 안에서, digitalRead에 대한 각 호출은 더듬이가 눌러졌다면 0를 리턴하고, 그렇지 않다면 1를 리턴합니다. wLeft와 wRight 변수로 값들이 저장되며, 더듬이-왼쪽과 더듬이-오른쪽에 대해 short이다.

```
byte wLeft = digitalRead(5);      // Copy left result to wLeft
byte wRight = digitalRead(7);     // Copy right result to wRight
```

다음, Serial.print(wLeft)함수는 직렬모니터(Serial Monitor)에 wLeft 의 값을 나타냅니다. 그리고 Serial.println(wRight)함수는 직렬모니터에 wRight 의 값을 나타냅니다.

```
Serial.print(wLeft);              // Display left whisker state
Serial.println(wRight);           // Display right whisker state
```

loop함수가 반복하기 전에, delay(50)을 넣는다. 이것은 직렬모니터가 매초 마다 받는 많은 메시지를 천천히 하기 위해서입니다. 아마 이것이 필요 없을지도 모르지만, 우리는 보다 오래된 하드웨어와 어떤 OS컴퓨터에 대해서 버퍼의 과동작(너무 많은 데이터를 저장) 가능성을 막기 위해서 입니다.

여러분 차례 - 함수 호출 nesting

여러분의 스케치는 digitalRead로부터 값을 저장하기 위해 변수를 사용할 필요가 없습니다. 대신에 digitalRead리턴하는 값 0 혹은 1이 Serial.print()함수 안에서 함수 호출하는 함수내의 함수에 의해 바로 사용되어진다. 직렬모니터에 리턴 값을 곧바로 보낸다. 이러한 경우, 여러분의 loop함수는 다음과 같을 것이다.:

```
void loop()                            // Main loop auto-repeats
{
  Serial.print(digitalRead(5));      // Display wLeft
  Serial.println(digitalRead(7));   // Display wRight

  delay(50);                           // Pause for 50 ms
}
```

√ loop함수를 대치하여, 스케치를 업로드 하시오. 그리고 더듬이가 같은 함수로 작동하는지를 확인하시오.

실습2: 더듬이 필드 테스트

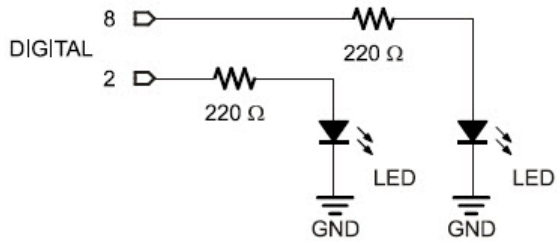
만약 여러분이 얼마의 시간이 지난 후 더듬이를 테스트해야 한다면 컴퓨터 없이 무엇을 할 것인가? 이러한 경우, 직렬모니터는 이용할 수 없으며, 여러분이 할 수 있는 것은 무엇입니까? 한 가지 방법은 더듬이의 상태를 나타내기 위해 LED 회로를 이용하는 것입니다. 이것은 더듬이가 눌려지거나 혹은 눌려지지 않았을 때 LED에 불을 켜는 간단한 스케치로 만들어집니다.

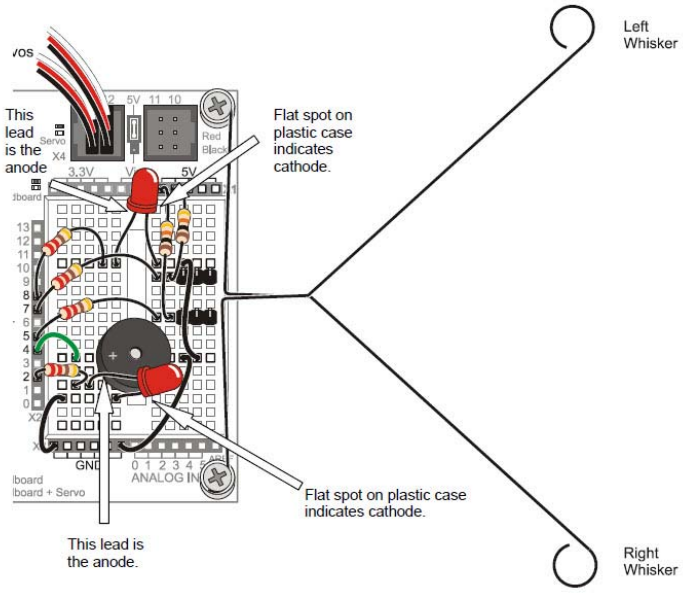
부품 목록:

(2) 220 Ω 저항
(2) 적색 LED

LED 더듬이 테스트 회로를 구성하라.

√ BOE Shield-Bot의 배터리 팩과 USB 케이블을 분리합니다.
√ 아래와 같이 회로를 구성하시오.





LED 더듬이 테스트 회로 프로그래밍

√ WhiskerStates를 TestWhiskersWithLEDs로 다시 저장하시오.
√ setup()함수에 pinMode() 호출을 추가하고, 디지털 핀 8과 2를 출력으로 세팅하시오.

```
pinMode(8, OUTPUT);            // Left LED indicator -> output
pinMode(2, OUTPUT);            // Right LED indicator -> output
```

√ 더듬이 입력 상태를 제어하는 LED를 만들기 위해, Serial.println(wRight)와 delay(50)사이에 두 개의 if...else문장을 삽입하시오.

```
if(wLeft == 0)                 // If left whisker contact
{
  digitalWrite(8, HIGH);       // Left LED on
}
else                           // if no left whisker contact
{
```

```
    digitalWrite(8, LOW);       // Left LED off
  }

  if(wRight == 0)               // If right whisker contact
  {
    digitalWrite(2, HIGH);      // Right LED on
  }
  else                          // If no right whisker contact
  {
    digitalWrite(2, LOW);       // Right LED off
  }
```

if...else문장은 조건을 기본으로 하는 코드 블록을 실행한다는 것을 상기하시오. 다음은 만약 wLeft가 0을 저장한다면, digitalWrite(8, HIGH)함수 호출을 실행할 것입니다. 만약 wLeft가 1을 저장한다면, digitalWrite(8, LOW)함수 호출을 실행할 것입니다. 결과는? 왼쪽 더듬이가 눌려졌을 때 혹은 왼쪽 LED가 불이 켜지고, 왼쪽 더듬이가 눌려지지 않았을 경우 LED불은 꺼진다. 두 번째 if…else 문장은 LED회로에 대해 wRight가 위와 같은 일을 합니다.

√ BOE 쉴드의 전원스위치를 1의 위치에 놓으시오.
√ 아두이노 프로그래밍 케이블을 다시 연결하시오.
√ TestWhiskersWithLeds를 저장하고, 여러분의 아두이노로 업로드하시오.
√ 브레드보드 안의 3-핀 헤드 기둥에 각각의 더듬이를 서서히 누르면서 스케치를 테스트하시오. 브레드보드의 한 쪽 위에 있는 빨강 LED는 더듬이가 눌려졌을 때 더듬이가 접촉이 되었다는 것을 나타내는 불이 켜집니다.
√ 두 개 LED가 불이 들어오고 어떤 상태에서도 유지된다면, 여러분의 전원스위치가 아마도 0의 위치에 있는 것입니다. 전원스위치를 1의 위치로 옮기고 다시 시도하시오.

```
/*
* Robotics with the BOE Shield - TestWhiskersWithLeds
* Display left and right whisker states in Serial Monitor.
* 1 indicates no contact; 0 indicates contact.
* Display whisker states with LEDs.  LED on indicates contact;
* off indicates none.
```

*/

```
void setup()                          // Built-in initialization block
{
  pinMode(7, INPUT);                  // Set right whisker pin to input
  pinMode(5, INPUT);                  // Set left whisker pin to input
  pinMode(8, OUTPUT);                 // Left LED indicator -> output
  pinMode(2, OUTPUT);                 // Right LED indicator -> output

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

  Serial.begin(9600);                 // Set serial data rate to 9600
}

void loop()                           // Main loop auto-repeats
{
  byte wLeft = digitalRead(5);        // Copy left result to wLeft
  byte wRight = digitalRead(7);       // Copy right result to wRight

  if(wLeft == 0)                      // If left whisker contact
  {
    digitalWrite(8, HIGH);            // Left LED on
  }
  else                                // if no left whisker contact
  {
    digitalWrite(8, LOW);             // Left LED off
  }

  if(wRight == 0)                     // If right whisker contact
  {
    digitalWrite(2, HIGH);            // Right LED on
  }
  else                                // If no right whisker contact
  {
```

```
    digitalWrite(2, LOW);              // Right LED off
  }

  Serial.print(wLeft);                // Display wLeft
  Serial.println(wRight);              // Display wRight
  delay(50);                          // Pause for 50 ms
}
```

실습3: 더듬이로 주행

이전에 우리들의 스케치는 ABOT이 여러분 혹은 프로그래머에 의해 사전 정의된 동작 리스트를 실행하는 것으로 만들어졌다. 이제 여러분은 더듬이 스위치를 모니터하고 응답으로 활동을 트리거하는 아두이노를 만들기 위해 스케치를 사용할 수 있다. ABOT을 운전하고 만약 어떤 것에 부딪친다면 스스로 자율 주행을 선택하도록 스케치를 사용할 수 있다. 이것이 자율 로봇의 예이다.

더듬이 주행

RoamingWithWhiskers 스케치는 ABOT이 하나 혹은 양쪽에 장애물을 만날 때까지 더듬이 입력을 지켜보면서 앞으로 움직이도록 만드는 것이다. 아두이노가 더듬이의 전기적인 접촉을 센싱하자 마자 무엇을 할 것인지를 결정하기 위해 if…else문을 사용할 것이다. 결정 코드는 더듬이가 접촉/비접촉의 여러 가지 조합의 경우를 점검할 것이다. 그리고 후진-회전 주행을 실행하기 위해 4장으로부터 주행 함수를 호출한다. 그런 다음, ABOT은 다른 장애물을 만날 때까지 전진 동작을 다시 할 것이다.

예제 스케치: RoamingWithWhiskers

스케치를 먼저 시도하자. 그리고 이것이 어떻게 움지이는가를 좀 더 가까이서 지켜보자.

√ 3-위치 스위치를 위치 1로 놓으시오.
√ ABOT의 배터리팩을 아두이노에 다시 연결하시오.
√ 입력, 저장 그리고 RoamingWithWhiskers를 업로드하시오.
√ 프로그래밍 케이블을 ABOT에서 분리하고 전원 스위치를 2로 놓는다.
√ 바닥에 ABOT을 놓고, 이것이 움직이도록 해보라. 더듬이 스위치를 장착한 로봇이 자신의 경로에서 장애물을 만났을 때, 로봇은 후진, 회전 그리고 나서 새로운 방향으로 돌아다닐 것이다.

```
// Robotics with the BOE Shield - RoamingWithWhiskers
// Go forward.  Back up and turn if whiskers indicate BOE Shield bot bumped
```

// into something.

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

void setup()                          // Built-in initialization block
{
  pinMode(7, INPUT);                  // Set right whisker pin to input
  pinMode(5, INPUT);                  // Set left whisker pin to input

  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
}

void loop()                           // Main loop auto-repeats
{
  byte wLeft = digitalRead(5);        // Copy left result to wLeft
  byte wRight = digitalRead(7);       // Copy right result to wRight

  if((wLeft == 0) && (wRight == 0))   // If both whiskers contact
  {
    backward(1000);                   // Back up 1 second
    turnLeft(800);                    // Turn left about 120 degrees
  }
  else if(wLeft == 0)                 // If only left whisker contact
  {
    backward(1000);                   // Back up 1 second
    turnRight(400);                   // Turn right about 60 degrees
  }
  else if(wRight == 0)                // If only right whisker contact
  {
```

```
    backward(1000);                 // Back up 1 second
    turnLeft(400);                  // Turn left about 60 degrees
  }
  else                              // Otherwise, no whisker contact
  {
    forward(20);                    // Forward 1/50 of a second
  }
}

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
}
```

<RoamingWithWhiskers가 어떻게 일을 하는가?>

loop함수 안에 있는 if...else if...else문장은 더듬이가 주의를 요구하는 어떤 상태를 점검합니다. 문장의 시작은 if((wLeft == 0) && (wRight == 0))입니다. 번역하면, “wLeft변수와 wRight변수가 모두 0인 조건을 만족한다면”을 의미하는 것이다. 두 변수가 0이라면 if 문장의 코드 블록은backward(1000)와 turnLeft(800)을 실행합니다.

```
 if((wLeft == 0) && (wRight == 0)) // If both whiskers contact
{
  backward(1000);                  // Back up 1 second
  turnLeft(800);                   // Turn left about 120 degrees
}
```

if…else if…else문장 안에서, 스케치는 조건이 만족되지 않는 경우 코드 블록을 건너뛰게 됩니다. 그리고 조건을 만족할 때까지 점검을 계속 할 것이다. 스케치가 참 값을 찾았을 때, 코드 블록 안에 있는 무엇인가를 실행한다. 그런 다음 또 다른 점검 없이 if…else if…else문장의 끝으로 건너뛴다. 그리고 스케치 안에 있는 그 밖의 다른 명령에 따라 움직인다.

그래서 만약 양쪽 더듬이가 눌려지지 않았다면, 첫 번째 if문은 참이 아니고 코드 블록은 건너뛴다. 스케치는 첫 번째 else if문을 점검할 것이다. 그래서, 왼쪽 더듬이가 눌려지고 이 문장의 코드 블록 안에 있는 함수를 실행할 것이다. 1초 동안 뒤로 움직이고 난 후, 0.4초 동안 왼쪽으로 회전할 것이다. 스케치는 조건의 나머지 부분은 건너뛰고 마지막 else문 뒤에 오는 어떤 동작을 수행할 것이다.

```
 else if(wLeft == 0)               // If only left whisker contact
{
  backward(1000);                  // Back up 1 second
  turnRight(400);                  // Turn right about 60 degrees
}
```

만약 오른쪽 더듬이가 장애물을 감지했다면, 첫 번째 두 코드 블록은 건너뛰고 if(wRight == 0) 블록을 실행할 것이다.

```
  else if(wRight == 0)                // If only right whisker contact
{
  backward(1000);                     // Back up 1 second
  turnLeft(400);                      // Turn left about 60 degrees
}
```

이전의 모든 문장 조건이 참을 나타내지 않을 경우 else 함수가 실행된다. 이것은 필요하지 않지만, 더듬이가 눌려지지 않을 경우 유용하다. 만약 그러한 경우, ABOT은 20ms동안 앞쪽 방향으로 진행한다. loop를 반복하기 전에 왜 이런 작은 시간을 사용하는가? 재점검하기 전에 작은 전진 시간은 ABOT이 앞으로 움직일 때 더듬이 센서로부터 빠르게 변화를 감지하기 위한 것이다.

```
  else                                // Otherwise, no whisker contact
{
  forward(20);                        // Forward 1/50 of a second
}
```

전진, 후진, 좌회전 그리고 우회전 함수들은 4장 활동 #5에서 그리고 MovementsWithSimpleFunctions 스케치에서 소개될 것이다. 확실히 이 함수들은 코드를 단순화할 것이다.

여러분 차례

ABOT이 달리는 주행상태를 LED를 통해 나타나도록 만들기 위해 스케치의 if...else if...else 문장을 수정할 수 있다. 표시기 LED회로에서 HIGH와 LOW 신호를 보내는 digitalWrite함수 호출을 추가한다.

예는 다음과 같다.

```
  if((wLeft == 0) && (wRight == 0)) // If both whiskers contact
```

```
{
  digitalWrite(8, HIGH);          // Left LED on
  digitalWrite(2, HIGH);          // Right LED on
  backward(1000);                   // Back up 1 second
  turnLeft(800);                   // Turn left about 120 degrees
}
else if(wLeft == 0)                // If only left whisker contact
{
  digitalWrite(8, HIGH);          // Left LED on
  digitalWrite(2, LOW);           // Right LED off
  backward(1000);                   // Back up 1 second
  turnRight(400);                  // Turn right about 60 degrees
}
else if(wRight == 0)               // If only right whisker contact
{
  digitalWrite(8, LOW);           // Left LED off
  digitalWrite(2, HIGH);          // Right LED on
  backward(1000);                   // Back up 1 second
  turnLeft(400);                   // Turn left about 60 degrees
}
else                                // Otherwise, no whisker contact
{
  digitalWrite(8, LOW);           // Left LED off
  digitalWrite(2, LOW);           // Right LED off
  forward(20);                     // Forward 1/50 of a second
}
```

√ LED표시기를 이용한 주행상태를 알려주는 ABOT을 만들기 위해 RoamingWithWhiskers 안에 있는 if...else if...else문장을 수정하시오.

√ setup함수 안에서 디지털 핀을 출력으로 놓는다는 것을 기억하시오. 그렇게 함으로써 LED에 실제적인 전류가 공급될 수 있습니다.

```
  pinMode(8, OUTPUT);           // Left LED indicator -> output
  pinMode(2, OUTPUT);           // Right LED indicator -> output
```

실습4: 모서리 탈출을 위한 인공지능

마지막 스케치를 주목하십시오. ABOT은 코너에서 곤경에 빠질 수도 있습니다. 코너로 들어갔을 때, 왼쪽 더듬이는 왼쪽 벽을 접촉하고 그래서 뒤로 물러나고 오른쪽으로 회전한다. ABOT이 앞쪽으로 다시 움직일 때, 오른쪽 더듬이가 오른쪽 벽을 감지할 것이다. 그래서 뒤로 그리고 왼쪽으로 회전을 할 것이다. 그런 다음 로봇은 왼쪽 벽을 다시 접촉할 것이고 그런 다음 오른쪽 벽을 다시 접촉하는 것을 누군가가 이러한 곤경으로부터 꺼내줄 때까지 계속 할 것입니다.

코너 탈출 프로그래밍

RoamingWithWhiskers는 이러한 문제와 행동을 감지하기 위해 확장될 수 있습니다. 방법은 양쪽 더듬이가 물체를 번갈아가며 접촉하는 횟수를 세는 것입니다. 이것을 하기위해 스케치는 각각의 더듬이가 이전의 접촉 동안 무엇을 했는지를 기억해야 합니다. 그런 다음, 그러한 상태와 현재 더듬이 접촉 상태를 비교해야 합니다. 만약 그들이 반대로 있다면, 카운터(counter)에 1를 더합니다. 만약 카운터가 여러분이 정해 놓은 한계치를 넘는 경우, 로봇은 U턴을 할 것이고 코너를 탈출합니다. 그리고 카운터를 리셋(reset)합니다.

다음 스케치는 if문장을 다른 한쪽에 새롭게 만드는 것입니다. 스케치는 하나의 조건을 점검하고, 이 조건이 사실이라면, 첫 번째 if 문장의 코드 블록 내에 다른 조건을 점검합니다. 우리는 다음 스케치에서 교차 더듬이 접촉을 연속으로 점검하는 이 방법을 사용할 것입니다.

예제 스케치: EscapingCorners

이 스케치는 여러분의 ABOT이 어느 쪽 더듬이를 먼저 눌렀는지에 따라 다르지만 더듬이가 네 번째 혹은 다섯 번째 교차해서 눌러서 코너를 탈출하기 위해 역으로 그리고 U턴을 실행하도록 할 것입니다.

√ 전원 스위치를 위치 1로 놓고, EscapingCorners를 입력하고 저장하고 업로드하시오.
√ ABOT이 돌아 다니면서 더듬이를 교차로 누르는 이 스케치를 테스트한다.

더듬이가 교차로 네 번 혹은 다섯 번 접촉한 후에 역으로 그리고 U턴을 실행해야 합니다.

```
/*
* Robotics with the BOE Shield - EscapingCorners
* Count number of alternate whisker contacts, and if it exceeds 4, get out
* of the corner.
*/
```

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

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

byte wLeftOld;                      // Previous loop whisker values
byte wRightOld;
byte counter;                       // For counting alternate corners

void setup()                         // Built-in initialization block
{
  pinMode(7, INPUT);                // Set right whisker pin to input
  pinMode(5, INPUT);                // Set left whisker pin to input
  pinMode(8, OUTPUT);               // Left LED indicator -> output
  pinMode(2, OUTPUT);               // Right LED indicator -> output

  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

  wLeftOld = 0;                     // Init. previous whisker states
  wRightOld = 1;
  counter = 0;                      // Initialize counter to 0
```

```
}

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

  // Corner Escape

  byte wLeft = digitalRead(5);       // Copy right result to wLeft
  byte wRight = digitalRead(7);      // Copy left result to wRight

  if(wLeft != wRight)                // One whisker pressed?
  {                                  // Alternate from last time?
    if ((wLeft != wLeftOld) && (wRight != wRightOld))
    {
      counter++;                     // Increase count by one
      wLeftOld = wLeft;              // Record current for next rep
      wRightOld = wRight;
      if(counter == 4)               // Stuck in a corner?
      {
        wLeft = 0;                    // Set up for U-turn
        wRight = 0;
        counter = 0;                  // Clear alternate corner count
      }
    }
    else                             // Not alternate from last time
    {
      counter = 0;                    // Clear alternate corner count
    }
  }

  // Whisker Navigation
  if((wLeft == 0) && (wRight == 0))    // If both whiskers contact
  {
    backward(1000);                    // Back up 1 second
```

```
    turnLeft(800);                  // Turn left about 120 degrees
  }
  else if(wLeft == 0)               // If only left whisker contact
  {
    backward(1000);                 // Back up 1 second
    turnRight(400);                 // Turn right about 60 degrees
  }
  else if(wRight == 0)              // If only right whisker contact
  {
    backward(1000);                 // Back up 1 second
    turnLeft(400);                  // Turn left about 60 degrees
  }
  else                              // Otherwise, no whisker contact
  {
    forward(20);                    // Forward 1/50 of a second
  }
}

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
```

<어떻게 코너 탈출 작업을 하는가?>

이 스케치는 RoamingWithWhiskers의 수정된 버전이다. 그래서 코너 감지와 탈출을 위한 새로운 코드를 살펴 볼 것이다.

먼저 3개의 전역 바이트 변수가 추가된다: wLeftOld, wRightOld 그리고 counter. 변수 wLeftOld와 wRightOld는 이전의 더듬이 접촉 상태를 저장한다. 그래서 현재의 접촉 상태와 비교할 수 있다. 그런 다음 counter변수는 연속적인 교차 접촉의 횟수를 추적하는데 사용된다.

```
byte wLeftOld;                // Previous loop whisker values
byte wRightOld;
byte counter;                 // For counting alternate corners
```

이러한 변수들은 setup 함수에서 초기화한다. counter변수는 초기값이 0이다. 그러나 old변수 중의 하나는 초기값을 1로 설정해야 한다. 코너를 감지하는 루틴은 항상 교차 패턴을 살펴야 하고 이전의 교차 패턴과 비교를 해야 하므로 초기 교차 패턴이 있어야 한다. 그래서 loop함수가 그것들의 값을 점검하고 수정을 시작하기 전에 setup함수 내에 변수 wLeftOld와 wRightOld에 초기값을 할당한다.

```
  wLeftOld = 0;               // Initialize previous whisker
  wRightOld = 1;              // states
  counter = 0;                // Initialize counter to 0
```

“ // Corner Escape(코너 탈출)” 아래에 있는 첫 번째 일은 하나 혹은 다른 더듬이가 눌려졌는지를 점검하는 것이다. 이것을 하기 위한 가장 간단한 방법은 if 문장에 !=(같지 않다)연산자를 사용하는 것이다. “if(wLeft != wRight)”의 한글 의미는 “만약 wLeft 변수가 wRight 변수와 같지 않다면...”) 이다.

```
// Corner Escape

if(wLeft != wRight)          // One whisker pressed?
```

만약 하나의 더듬이가 눌려져서 같지 않다면, 이전 더듬이 접촉에서 반대쪽 패턴이 접촉했는지를 점검해야 한다. 그것을 하기 위해 현재 wLeft 값이 이전 상태의 것과 다르면 그리고 현재의 wRight값이 이전 상태의 것과 다른지를 if문을 이용하여 작성합니다. 그것은 if((wLeft != wLeftOld) && (wRight != wRightOld)) 이다. 만약 양쪽 조건들이 참이라면, 교차 더듬이 접촉을 추적하는 counter변수에 1를 더한다. 이것 역시 wLeftOld가 현재의 wLef를 저장하고 wRightOld가 현재의 wRight값을 저장하여 현재의 더듬이 패턴을 기억하는 것이다.

```
 if((wLeft != wLeftOld) && (wRight != wRightOld))
{
  counter++;                 // Increase count by one
  wLeftOld = wLeft;          // Record current for next rep
  wRightOld = wRight;
```

만약 이것이 4번째 연속적인 교차 더듬이 접촉이라면 counter변수는 0으로 리셋하고 U턴을 실행한다. if(counter == 4) 문장이 참이라면 이 코드 블록은 양쪽 더듬이가 눌려져 있는 더듬이 주행 루틴 방법을 실행할 것이다. 그것이 어떻게 하는가? wLeft과 wRight 모두 0으로 리셋 한다. 이것은 더듬이 주행 루틴이 양쪽 더듬이가 눌려졌을 것으로 생각할 것이고 U턴을 실시 할 것이다.

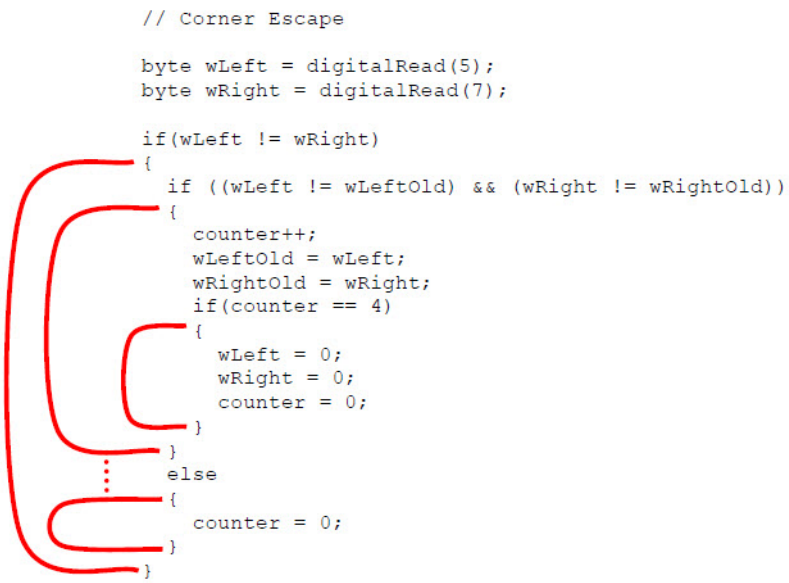
```
if(counter == 4)       // Stuck in a corner?
{
    wLeft = 0;           // Set up whisker states for U-turn
```

```
        wRight = 0;
        counter = 0;          // Clear alternate corner count
      }
  }
```

그러나 if((wLeft != wLeftOld) && (wRight != wRightOld)) 안에 있는 조건이 참이 아니라면, 더 이상의 교차 더듬이 접촉은 연속해서 발생하지 않는 것이라는 것을 의미한다. 그래서 ABOT은 코너에 갇히지 말아야 한다. 그런 경우, count변수는 0으로 리셋하며, 로봇이 코너를 찾았을 때, 다시 카운트를 시작할 것이다.

```
   else                        // Not alternate from last time
    {
      counter = 0;              // Clear alternate corner count
    }
  }
```

새롭게 만든 if 문장에 대하여 솜씨를 발휘해야하는 한 가지는 각 문장의 코드 블록에 대해 열고 닫는 중괄호가 추적을 유지하는 것이다. 아래 그림은 마지막 스케치로부터 새롭게 만든 if 문장을 나타낸다. 아두이노 편집기에서 여러분은 코드 블록을 강조하기 위해 중괄호를 더블-클릭할 수 있다. 그러나 때때로, 코드를 프린터하고 블록을 보기 쉽도록 모든 중괄호 열기와 닫기를 연결하는 간단한 선을 그릴 것이다. 이것은 아주 새롭게 만들어진 코드에서 버그를 찾는데 유용할 것이다.



```
// Corner Escape

byte wLeft = digitalRead(5);
byte wRight = digitalRead(7);

if(wLeft != wRight)
{
  if ((wLeft != wLeftOld) && (wRight != wRightOld))
  {
    counter++;
    wLeftOld = wLeft;
    wRightOld = wRight;
    if(counter == 4)
    {
      wLeft = 0;
      wRight = 0;
      counter = 0;
    }
  }
  else
  {
    counter = 0;
  }
}
```

이 그림에서 if(wLeft != wRight) 문장의 코드 블록은 결정 코드 부분의 나머지 모든 부분을 포함한다. 만약 wLeft와 wRight가 "같다"로 판정 난다면, 아두이노는 마지막 중괄호 닫기 '}'는 무슨 코드가 오더라도 건너뛸 것이다. 두 번째 레벨의 if문은 이전 값과 새로운 값의 wLeft와 wRight 값을 비교할 것이다. 형식은 if ((wLeft != wLeftOld) && (wRight != wRightOld))이다. 이 코드 블록의 끝 괄호는 if(counter==4) 코드 블록 아래에 있다. if ((wLeft != wLeftOld) && (wRight != wRightOld)) 문장은 만약 더듬이 값이 이전의 값과 반대가 아니라면 그 밖의 조건에서 counter를 0으로 리셋 하는 else조건을 가진다.

√ 그림 안에 있는 코드를 주의 깊게 살펴보시오.
√ wLeft=0, wRight=0 그리고 counter=3이라는 것을 기억하고, 이 문장이 무엇을 하는지를 생각하시오.
√ wLeft=1, wRight=0, wLeftOld=0, wRightOld=1 그리고 counter=3이라는 것을 기억하시오. 한 줄씩 다시 코드를 살펴보고 각 스텝에서 각 변수에 무슨 일 일어나는지를 설명하시오.

여러분 차례

EscapingCorners안에 있는 if문장 중의 하나는 카운터(counter)가 4에 도달하는

지를 점검합니다.

√ 값을 5와 6으로 증가시켜 그 효과를 테스트 해보시오. 교차 더듬이 접촉의 수를 카운팅하거나 여러분이 출발하는 벽면이 하나 이상인 것을 명심하시오.

제5장 종합

이 장은 ABOT에 대한 최초의 센서 시스템을 소개하였고 접촉에 의해 로봇 자신이 이리저리 돌아다니고 주행하는 것을 보였습니다. 이 장의 후미에 있는 프로젝트는 요구 기술로 만들어졌습니다. 그리고 새로운 방법의 다양성을 적용하였습니다.

전기적 부분

• 정상상태에서 open, momentary, single-pole, single-throw, 촉각 스위치 회로 구성

프로그래밍

• 입력에 대한 디니털 I/O를 설정하고 핀의 입력 상태를 모니터링 하기위해 아두이노 언어의 pinMode와 digitalRead 함수를 이용.
• 전역 변수의 선언과 초기화.
• 함수 호출을 새로 작성.
• 센서의 입력 상태를 기초로 프로그램 흐름을 관리하기 위해 if…else문을 이용.
• 새롭게 작성된 if 문장의 3단계 레벨 이용.

로봇 기술

• 센서 입력을 기초로 한 자율 로봇 주행
• 인공 지능의 간단한 예를 이용하여 자율적으로 굴러다니는 BOE Shield-Bot이 코너에서 움직일 수 없다면 빠져나오도록 할 수 있다.

엔지니어링 기술

• 부 시스템 다시 테스트 - 좋은 습관을 만들어라
• 시스템 입력의 상태를 시각적으로 나타내기 위해 LED표시기 이용
• 프로그램 흐름을 시각적으로 나타내기 위해 LED 표시기 이용

제5장 과제

질문

1. 더듬이는 전기적으로 무엇과 연결되어 있는가?

2. 더듬이가 눌려졌을 때 이것을 모니터링하는 I/O핀에 몇 볼트의 전압 발생하는가? digitalRead함수가 되돌려 주는 이진 값은 무엇인가? 만약 디지털 핀 8이 더듬이 회로를 모니터하기위해 사용되어진다면, 더듬이가 눌려졌을 때 digitalRead가 되돌려 주는 값은 얼마인가? 그리고 더듬이가 눌려지지 않았을 때 되돌려 주는 값은 얼마인가?

3. 만약 digitalRead(7)== 1이라면, 무엇을 의미하는가? digitalRead(7)== 0이라면 무엇을 의미하는가? digitalRead(5)== 1 그리고 digitalRead(5)== 0는 어떠한가?
4. 더듬이의 상태를 기초로 다른 주행 함수를 호출하기위해 이 장에서 사용하고 있는 문장은 무엇인가?

5. 이중의 if 문장을 가지는 목적은 무엇인가?

연습문제

1. 더듬이 접촉을 추적하기위해 싱글 변수 whiskers를 이용하는 루틴을 적어라. 이것은 더듬이가 접촉되지 않으면 3을 저장할 것이고, 오른쪽 더듬이가 접촉되면 2를 저장, 왼쪽 더듬이가 저장되면 1를 저장 혹은 양쪽 더듬이가 접촉되면 0를 저장한다. 힌트 : 결과에 2를 곱한다.
2. RoamingWithWhiskers안에 있는 loop함수를 수정하라 그래서 ABOT이 정지하도록 만들고 양쪽의 더듬이가 동시에 접촉되었을 때 재출발을 하지 않는다.
3. pause함수를 만들어 RoamingWithWhiskers에 추가한다. 이것은 ABOT이 어떤 시간까지 멈추는 일을 한다.
4. loop함수를 수정해서 ABOT이 뒤로 그리고 좌회전 전에 0.5초 동안 정지하도록 만들어라.

프로젝트

1. RoamingWithWhiskers를 수정해서 ABOT이 정지하고 일반적인 불분명한 주행을 실행하기 전에 4kHz 비프를 100ms 동안 지속하도록 만들어라. 양쪽 더듬이 접촉이 같은 샘플 동안 감지되었다면 비프 소리를 두 번 내도록 만들어라. 힌트: 연습 문제에서 사용된 pause함수를 이용하라. servo 움직임으로부터 tone호출 후 0.2초 정지는 0.1초 톤을 분리하거나, 혹은 여러분은 1초 톤을 들을 수 있다.

2. RoamingWithWhiskers를 수정하여 ABOT이 1야드(혹은 1미터)지름 원 안에서 굴러다닌다. 여러분이 한 개의 더듬이에 터치 할 때 ABOT은 보다 작은 원 안에서 움직일 것이다. 다른 더듬이를 터치할 때 ABOT은 좀더 큰 지름을 가지는 원 안에서 움직일 것이다.

제5장 해답

질문 해

1. 인반적으로 개방, 모멘트, 싱글-폴, 싱글-쓰루, 촉각 스위치
2. 0 볼트, digitalRead에의해 이진 수 0 결과가 리턴된다.
digitalRead(8) == 0, 더듬이가 눌려 졌을 때
digitalRead(8) == 1, 더듬이가 눌려지지 않았을 때

3. digitalRead(7)== 1 는 오른쪽 더듬이가 눌려지지 않았을 때를 의미한다.
digitalRead(7)== 0 는 오른쪽 더듬이가 눌려졌을 때를 의미한다.
digitalRead(5)== 1 는 왼쪽 더듬이가 눌려지지 않았을 때를 의미한다.
digitalRead(5)== 0 는 왼쪽 더듬이가 눌려졌을 때를 의미한다.

4. 이 장에서는 더듬이 조건들과 주행함수 호출을 평가하기위해 if, if…else 그리고 if…else if…else 문을 사용하였다.

5. 만약 하나의 조건이 참으로 나타날 경우, 코드는 이중으로 된 if 문과 함께 다른 조건을 평가할 필요가 있을 것이다.

연습문제 해

1. digitalRead는 1혹은 0을 리턴하므로, 여러분의 코드는 digitalRead(5)에 2를 곱할 수 있다. 그리고 whiskers 변수에 결과를 저장한다. whiskers변수에 대해 digitalRead(7)의 결과를 추가한다. 그리고 결과는 더듬이 없는 것에 대해 3일 것이다.

```
// Robotics with the BOE Shield Chapter 5, Exercise 1
// Value from 0 to 3 indicates whisker states:
// 0 = both, 1 = left, 2 = right, 3 = neither.

void setup()                          // Built-in initialization block
{
```

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

  pinMode(7, INPUT);                  // Set right whisker pin to input
  pinMode(5, INPUT);                  // Set left whisker pin to input

  Serial.begin(9600);                 // Set data rate to 9600 bps
}

void loop()                           // Main loop auto-repeats
{
  byte whiskers = 2 * digitalRead(5);
  whiskers += digitalRead(7);

  Serial.println(whiskers);           // Display wLeft

  delay(50);                          // Pause for 50 ms
}
```

2. if((wLeft == 0) && (wRight == 0)) 블록 안에서, backward와 turnLeft함수를 제거하고 servoLeft.detach and servoRight.detach로 대체한다.

```
 void loop()                          // Main loop auto-repeats
{
  byte wLeft = digitalRead(5);        // Copy right result to wLeft
  byte wRight = digitalRead(7);       // Copy left result to wRight

  if((wLeft == 0) && (wRight == 0))   // If both whiskers contact
  {
    pause(500);                       // Pause motion for 0.5 seconds
    backward(1000);                   // Back up 1 second
    turnLeft(800);                    // Turn left about 120 degrees
  }
  else if(wLeft == 0)                 // If only left whisker contact
```

```
  {
    pause(500);                    // Pause motion for 0.5 seconds
    backward(1000);                // Back up 1 second
    turnRight(400);                // Turn right about 60 degrees
  }
  else if(wRight == 0)             // If only right whisker contact
  {
    pause(500);                    // Pause motion for 0.5 seconds
    backward(1000);                // Back up 1 second
    turnLeft(400);                 // Turn left about 60 degrees
  }
  else                             // Otherwise, no whisker contact
  {
    forward(20);                   // Forward 1/50 of a second
  }
}
```

3. 해:

```
 void pause(int time)                // Pause drive wheels
{
  servoLeft.writeMicroseconds(1500);    // Left wheel stay still
  servoRight.writeMicroseconds(1500);   // Right wheel stay still
  delay(time);                          // Maneuver for time ms
}
```

4. forward함수는 더듬이가 다시 점검을 하기 전에 20m동안 앞으로 전진하도록 가정하였기 때문에 else 조건 안에 있는 pause를 호출은 확실하게 만들지 않는다.

```
 void loop()                        // Main loop auto-repeats
{
  byte wLeft = digitalRead(5);      // Copy right result to wLeft
  byte wRight = digitalRead(7);     // Copy left result to wRight
```

```
  if((wLeft == 0) && (wRight == 0)) // If both whiskers contact
  {
    pause(500);                     // Pause motion for 0.5 seconds
    backward(1000);                 // Back up 1 second
    turnLeft(800);                  // Turn left about 120 degrees
  }
  else if(wLeft == 0)               // If only left whisker contact
  {
    pause(500);                     // Pause motion for 0.5 seconds
    backward(1000);                 // Back up 1 second
    turnRight(400);                 // Turn right about 60 degrees
  }
  else if(wRight == 0)              // If only right whisker contact
  {
    pause(500);                     // Pause motion for 0.5 seconds
    backward(1000);                 // Back up 1 second
    turnLeft(400);                  // Turn left about 60 degrees
  }
  else                              // Otherwise, no whisker contact
  {
    forward(20);                    // Forward 1/50 of a second
  }
}
```

프로젝트 해

1. 이 문제를 풀기위한 해는 요구되는 파라미터와 함께 비프를 만드는 문장을 쓰는 것이다. 비프가 시작하자마자, ABOT이 비프소리를 내는 동안까지 유지하기위해 pause함수를 호출한다. else문장의 코드 블록에 pause 호출을 더한다. 20ms 동안 어떤 멈춤도 없이 앞쪽 방향으로 반복적으로 가는 것이 필요하다.

```
 // RoamingWithWhiskers  Chapter 5 Project 1
// Go forward.  Back up and turn if whiskers indicate ABOT
// bumped into something.
```

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

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

void setup()                       // Built-in initialization block
{
  pinMode(7, INPUT);               // Set right whisker pin to input
  pinMode(5, INPUT);               // Set left whisker pin to input

  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
}

void loop()                        // Main loop auto-repeats
{
  byte wLeft = digitalRead(5); // Copy right result to wLeft
  byte wRight = digitalRead(7);  // Copy left result to wRight

  if((wLeft == 0) && (wRight == 0))// If both whiskers contact
  {
    tone(4, 4000, 100);            // Play a 0.1 ms tone
    pause(200);                    // Stop for 0.2 seconds
    tone(4, 4000, 100);            // Play a 0.1 ms tone
    pause(200);                    // Stop for 0.2 seconds
    backward(1000);                // Back up 1 second
    turnLeft(800);                 // Turn left about 120 degrees
  }
  else if(wLeft == 0)              // If only left whisker contact
  {
    tone(4, 4000, 100);            // Play a 0.1 ms tone
```

```
    pause(200);                 // Stop for 0.2 seconds
    backward(1000);             // Back up 1 second
    turnRight(400);             // Turn right about 60 degrees
  }
  else if(wRight == 0)          // If only right whisker contact
  {
    tone(4, 4000, 100);         // Play a 0.1 ms tone
    pause(200);                 // Stop for 0.2 seconds
    backward(1000);             // Back up 1 second
    turnLeft(400);              // Turn left about 60 degrees
  }
  else                          // Otherwise, no whisker contact
  {
    forward(20);                // Forward 1/50 of a second
  }
}

void pause(int time)            // Backward function
{
  servoLeft.writeMicroseconds(1500); // Left wheel clockwise

servoRight.writeMicroseconds(1500);//Rightwheelcounterclockwise
  delay(time);                      // Maneuver for time ms
}

void forward(int time)          // Forward function
{
  servoLeft.writeMicroseconds(1700);//Leftwheel 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);//Leftwheel counterclockwise

servoRight.writeMicroseconds(1700);//Rightwheelcounterclockwise
  delay(time);                       // Maneuver for time ms
}

void backward(int time)              // Backward function
{
  servoLeft.writeMicroseconds(1300);  // Left wheel clockwise

servoRight.writeMicroseconds(1700);//Rightwheelcounterclockwise
  delay(time);                       // Maneuver for time ms
}
```

2. 4장의 해로부터 원 스케치를 출발하라. loop 함수에 원 코드를 옮기고 지연을 50ms로 감소한다. 그래서 더듬이가 1초에 20번 접촉하는지를 감시할 수 있다. 그런 다음, 오른쪽 더듬이가 눌러졌을 때 오른쪽 바퀴가 천천히 움직이는, 혹은 왼쪽 더듬이가 눌러 졌을 때 오른쪽 바퀴 스피드가 증가하는 변수가 줄고 증가하는 if 문장과 함께 더듬이 모니터링 코드를 추가한다.

```
 // Robotics with the BOE Shield - Chapter 5, project 2 - WhiskerCircle
// ABOT navigates a circle of 1 yard diameter.
// Tightens turn if right whisker pressed, or reduces turn if left whisker
// is pressed.
```

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

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

int turn;

void setup()                         // Built-in initialization block
{
  pinMode(7, INPUT);                 // Set right whisker pin to input
  pinMode(5, INPUT);                 // Set left whisker pin to input

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

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

  turn = 0;

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

void loop()                          // Main loop auto-repeats
{                                    // Nothing needs repeating
  int wLeft = digitalRead(5);
  int wRight = digitalRead(7);

  if(wLeft == 0)
  {
    turn -= 10;
  }
  else if(wRight == 0)
```

```
  {
    turn += 10;
  }

  // Arc to the right
  servoLeft.writeMicroseconds(1600);  // Left wheel counterclockwise
  servoRight.writeMicroseconds(1438 + turn);//Rightwheelclockwise slower
  delay(50);                          // ...for 25.5 seconds
}
```

제6장 포토트랜지스터 빛감지 조정하기

광센서는 로봇과 산업용 제어에 많이 응용된다. 직물공장의 천 끝부분을 찾는 것, 일년중 다른 시간에 가로등 동작시간을 결정하는 것, 사진 찍을 때 그리고 농작물에 물주는 것 등에 응용된다.

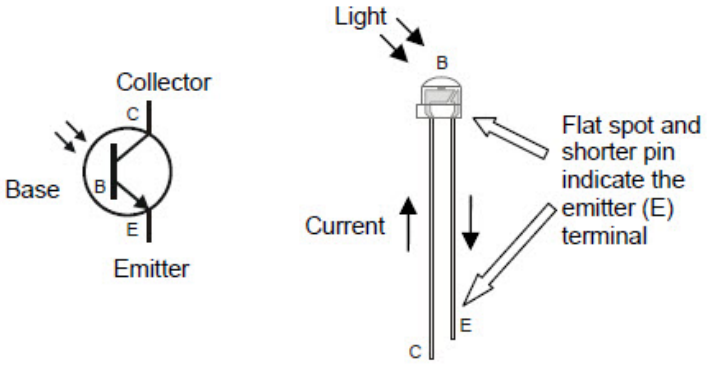
ABOT 키트에서 광센서는 가시광선에 반응하고, 또한 적외선(*infrared*)이라고 하는 비가시광선에도 반응한다. 이 센서들은 아두이노가 빛의 여러 가지 수준을 감지하도록 모니터할 수 있는 다른 회로에서 사용될 수 있다. 여러분의 스케치는 ABOT이 빛에 의해 조종되도록 확장될 수 있다. 마치 손전등 빔을 따라 움직이거나 어두운 방에서 열린 문으로 들어오는 광선을 따라 움직이는 것처럼 말이다.

- 6장 아두이노 코드 다운로드
- 다음 링크들을 따라 시작하세요.

포토트랜지스터(Phototransistor)란?

트랜지스터(*transistor*)는 3개의 단자중 2개를 통해 흐르는 전류의 양을 조절하는 밸브와 같다. 세 번째 단자는 얼마나 많은 전류가 두 개 단자 사이를 통과하는지를 조절한다. 트랜지스터 종류에 따라서 전류의 흐름이 전압에 의해 조절될 수 있고, 포토트랜지스터(phototransistor)의 경우처럼 빛에 의해 조절될 수 있다.

아래 그림은 여러분의 쉴드 키트에 있는 포토트랜지스터의 개략도와 부품 모양을 보여준다. 포토트랜지스터의 베이스(B)에 비춰지는 빛의 세기는 얼마나 많은 전류가 컬렉터(C)단자와 에미터(E)단자로 흐르게 할 것인가를 결정한다. 더 밝은 빛은 더 많은 전류를 흐르게 한다. 그리고 더 적은 빛은 전류를 덜 흐르게 한다.

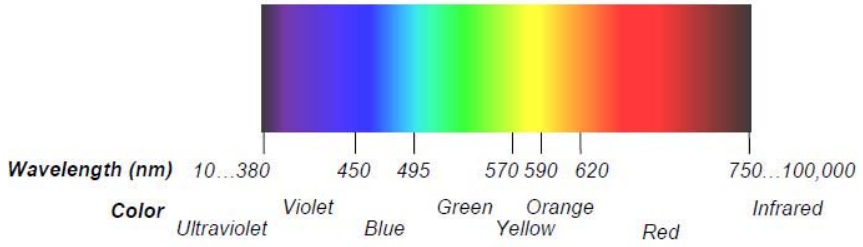


포토트랜지스터는 LED와 약간 닮은 점이 있다. 두 개의 소자는 유사성이 있는데, 첫째는 여러분이 회로에서 포토트랜지스터를 거꾸로 연결한다면, 그것은 동작하지 않는 점이고, 두 번째는 서로 다른 핀 길이를 가지고 있으며 단자를 구분하기 위하여 플라스틱 케이스의 평평한 부분을 가지고 있는 점이다. 두 개중 길이가 긴 것이 포토트랜지스터의 컬렉터 단자이다. 짧은 길이의 핀이 에미터를 가르키고, 그것이 포토트랜지스터의 플라스틱 케이스의 평평한 면과 더 가깝게 연결되어 있다.

광선

바다에서 여러분은 두 개의 인접한 파고점 사이 거리를 피트나 미터로 측정할

수 있다. 빛에 있어서도, 광선은 파동운동을 하고, 최고점 사이의 거리는 10억분의 1 미터인 나노미터(*nanometers* : nm)로 측정된다. 아래 그림은 우리가 친숙한 색깔과 자외선과 적외선처럼 우리 눈으로 식별할 수 없는 파장을 표시하고 있다.



로봇 쉴드 키트의 포토트랜지스터는 850 nm 파장에 가장 민감하고, 이것은 적외선 영역에 해당한다. 적외선은 눈으로 식별할 수 없지만, 할로겐등과 백열등 그리고 특히 태양광을 포함하여 많은 광원들이 상당한 양의 적외선을 방출한다. 또한 포토트랜지스터는 가시광선에도 덜 민감하게 반응하고, 특히 450nm 아래 파장에서 잘 반응한다.

이장의 포토트랜지스터 회로는 실내에서 형광등과 백열등에 잘 동작하도록 설계되었다. 태양 직사광 또는 할로겐 직사광을 피하도록 하시오. 이것은 포토트랜지스터에 너무 많은 적외선 광으로 흘러넘치지 않도록 하기 위한 것이다.

- 태양직사광이 실내로 들어않도록 창문을 막고, 어떤 할로겐 등의 빛도 천장에서 반사되도록 위쪽 방향으로 설치하시오.

과제 1: 단순 광을 전압 센서로

ABOT이 끝에 불빛이 있는 코스를 찾아가고 있다고 상상해보시오. 이번 코스에서 여러분 로봇의 마지막 임무는 불빛이 비추는 바로 아래에서 멈추는 것이다. 아두이노가 이진수 1을 별도의 광선을 감지하는 것으로 인식하고 또는 이진수 0을 분위기 광선을 감지하는 것으로 인식하는 간단한 포토트랜지스터 회로가 있다. 책상에 백열등과 손전등은 최상의 광선 조합을 만든다. 형광등과 LED광원 조합으로 과제의 회로를 사용하여 각각의 광원을 식별하는 것은 쉽지 않다.

분위기(Ambient)는 "모든 곳에 현재 존재하는"의 의미이다. 방안의 광선 수준이 분위기 광선에 해당한다고 생각할 수 있다.

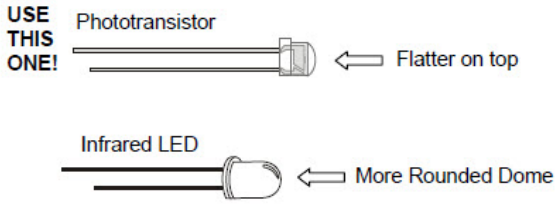
부품 목록

(1) 포토트랜지스터(phototransistor)
(2) 점프선(jumper wires)
(1) 저항, 2kΩ (빨강-흑색-빨강)
(1) 백열등 또는 형광등 또는 책상등

여러분의 로봇 실험구역 내에 광선 상태에 따라, 여러분은 2kΩ저항을 다음 저항들 중 하나로 교체할 수 있다.

(1) 저항, 220Ω (빨강-빨강-갈색)
(1) 저항, 470Ω (노랑-보라-갈색)
(1) 저항, 1kΩ (갈색-흑색-빨강)
(1) 저항, 4.7kΩ (노랑-보라-빨강)
(1) 저항, 10kΩ (갈색-흑색-주황)

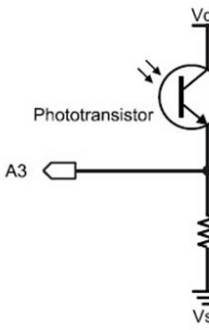
아래 그림은 여러분에게 포토트랜지스터와 적외선 LED가 비슷하게 보이기 때문에 서로 분리할 있도록 도움을 준다.

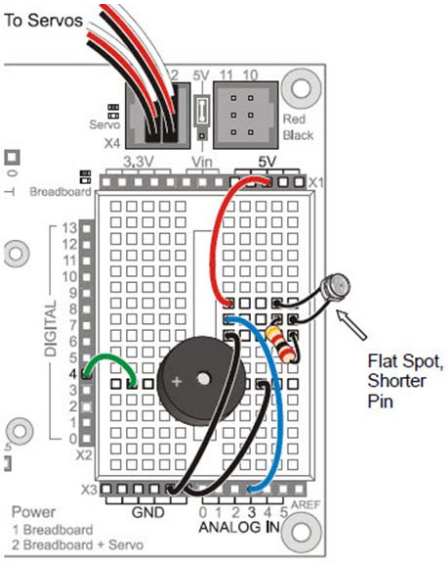


밝은 빛 감지기 설치

아래 개략도와 배선도는 밤에 자동으로 켜지는 가로등과 유사한 회로의 개략도와 배선도이다. 회로의 출력전압은 포토트랜지스터에 얼마나 많은 빛이 비춰지는가에 따라 변화한다. 아두이노는 한 개의 아날로그 입력 핀으로 전압 수준을 관찰할 것이다.

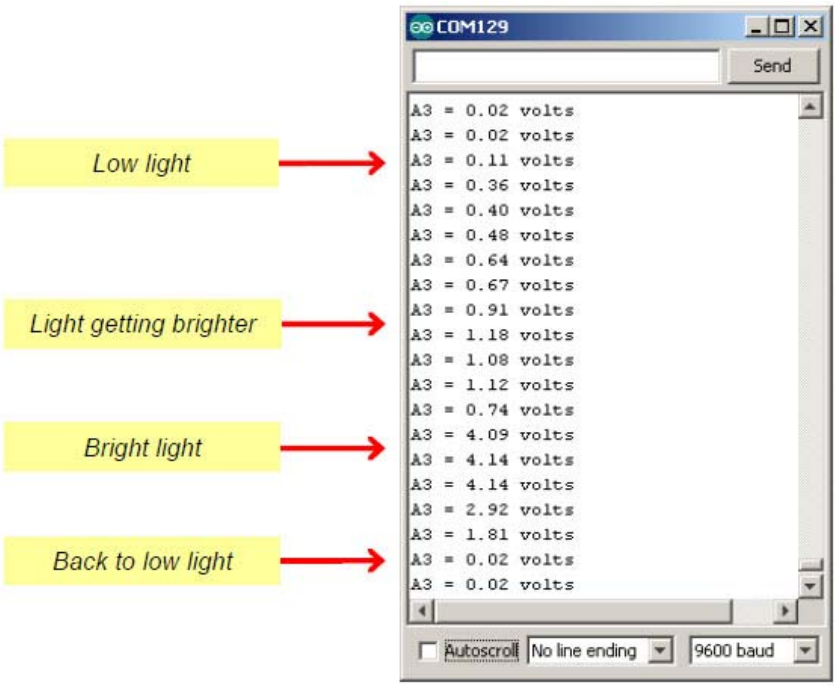
- 아두이노로부터 배터리 팩과 프로그램 케이블을 분리하고 BOE 쉴드의 스위치를 0으로 놓으시오.
- 더듬이 회로를 제거하고, 회로의 피에조스피커는 남겨두시오.
- 2 kΩ저항을 사용하여 회로를 구성하시오.
- 포토트랜지스터의 에미터 단자가 저항으로 연결되고 컬렉터가 5V에 연결되었는지 반복하여 점검하시오.
- 또한 포토트랜지스터의 단자들은 서로 닿지 않았는지 점검하시오.





예제 스케치: PhototransistorVoltage

PhototransistorVoltage 스케치는 쉴드로 접근할 수 있는 어두이노의 5개 아날로그 입력 채널들 중 하나와 A3사이에서 측정된 전압을 직렬 모니터가 나타낼 수 있도록 한다. 여러분이 구성한 회로에서, 전선은 A3에 연결되고 포토트랜지스터의 에미터와 저항이 서로 만나는 행으로 구성된다. 회로의 이 부분 전압은 포토트랜지스터의 변화에 의해 감지되는 광선 수준에 따라 변화하게 된다. 아래 직렬 모니터 캡쳐 화면은 측정전압의 몇 가지 예를 보여준다.



- 프로그램 케이블과 배터리 팩을 보드에서 분리하시오.
- 쉴드의 전원 스위치를 1의 위치에 놓으시오.
- PhototransistorVoltage 스케치를 입력하고 저장하고 아두이노로 업로드 하시오.
- 포토트랜지스 위로 손전등이나 램프를 천천히 움직이고, 직렬모니터에서 A3의 값을 관찰하시오. 더 밝은 빛은 전압 값을 더 크게 만들어야 하고 더 어두운 빛은 전압을 작게 나타내야 합니다.
- 분위기 광선이 형광등보다 밝다면 여러분은 2kΩ저항보다 더 작은 저항 값으로 교체할 필요가 있습니다. 1kΩ, 470Ω, 또는 220Ω까지 시도해 보세요.

```
/*
 * Robotics with the BOE Shield - PhototransistorVoltage
 * Display voltage of phototransistor circuit output connected to A3 in
 * the serial monitor.
 */

void setup()                      // Built-in initialization block
{
  Serial.begin(9600);             // Set data rate to 9600 bps
}

void loop()                       // Main loop auto-repeats
{
  Serial.print("A3 = ");          // Display "A3 = "
  Serial.print(volts(A3));        // Display measured A3 volts
  Serial.println(" volts");       // Display " volts" & newline
  delay(1000);                    // Delay for 1 second
}

float volts(int adPin)            // Measures volts at adPin
{                                 // Returns floating point voltage
 return float(analogRead(adPin)) * 5.0 / 1024.0;
}
```

더 밝은 빛에서 중단하기

HaltUnderBrightLight 스케치는 A3로 공급되는 전압이 3.5V를 초과할 만큼 충분히 밝은 빛을 포토트랜지스터가 감지할 때까지 ABOT이 전진하도록 할 것이다. 여러분은 3.5V를 마지막 스케치로부터 기록한 HIGH와 LOW전압 사이의 중간지점까지 변경할 수 있다.

- 마지막 스케치로부터 기록한 분위기 광선과 밝은 광선사이의 중간지점을 계산하시오.
- HaltUnderBrightLight 스케치에서, if(volts(A3) >3.5) 문장중 3.5 대신 중간값으로 대체하시오.
- HaltUnderBrightLight 의 수정본을 아두이노로 업로드 하시오.
- 마루 바닥에 손전등이나 램프를 두고, ABOT을 두어 걸음 떨어진 바닥에 놓고, 빛을 따라 똑바로 갈수 있도록 방향을 잡으시오.
- ABOT이 전진하도록 전원 스위치를 2의 위치로 옮기시오. 불빛 아래 얼마나 가까이에서 멈추었나요?
- ABOT이 밝은 불빛 바로 아래에서 멈추도록, if(volts(A3) >…)문장에서 여러분이 조작하는 문턱(threshold)값을 맞춰보시오.
-

```
/*
 * Robotics with the BOE Shield - HaltUnderBrightLight
 * Display voltage of phototransistor circuit output connected to A3 in
 * the serial monitor.
 */
```

```
#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

  servoLeft.writeMicroseconds(1700);    // Full speed forward
  servoRight.writeMicroseconds(1300);
}

void loop()                             // Main loop auto-repeats
{
  if(volts(A3) > 3.5)                   // If A3 voltage greater than 2
  {
    servoLeft.detach();                 // Stop servo signals
    servoRight.detach();
  }
}

float volts(int adPin)                  // Measures volts at adPin
{                                       // Returns floating point voltage
 return float(analogRead(adPin)) * 5.0 / 1024.0;
}
```

전압 함수가 어떻게 동작하는가?

아두이노의 A0, A1…A5 소켓이 아날로그를 디지털로 변환하는 아트멜(Atmel) 마이크로컨트롤러 핀에 연결되어 있다. 그것은 마이크로컨트롤러가 전압을 측정하는 방법이다. 전압을 많은 숫자로 나누고, 각각의 숫자가 전압을 대표하게 된다. 각각의 아두이노 아날로그 입력은 10비트 해상도를 가지고 있고, 10개의 2진수 자리는 측정전압을 설명하는데 사용한다. 10개의 2진수 비트는 1부터 1023까지를 헤아릴 수 있게 한다. 즉 0을 포함한다면 전압수준이 총 1024개가 된다.

기본적으로 아두이노의 analogRead 함수는 전압 측정값이 5V 이내인 상태에서

0-1023사이의 값을 사용하도록 구성된다. 5V 를 1024의 다른 단계로 나누면 각각의 전압 단계는 5/1024 전압(4.89 mV)만큼 나누어진다.

예제:

analogRead 함수가 645를 표시; 실제 전압 값은 얼마인가?

답:

$$V = 645 \times \frac{5V}{1024}$$
$$= 3.1494140625\ V$$
$$\approx 3.15\ V$$

스케치는 volts(A3)를 전압함수라고 한다. 이때 A3는 adPin으로 통용되고, 함수 내에서 analogRead(adPin)는 analogRead(A3)가 된다. 그리고 A3에 공급되는 전압을 나타내는 0-1023사이의 값을 반환한다. analogRead 함수는 정수형 값을 반환하지만, float(analogRead(adPin)로 표현된다면 정수형 값이 실수형 값으로 변환된다.

```
float volts(int adPin)                          // Measures volts at adPin
{                                                   // Returns floating point
voltage
 return float(analogRead(adPin)) * 5.0 / 1024.0;
}
```

전압 함수 블록 계산 왼쪽이 반환되므로, 결과 값이 함수 호출로 반환된다.

PhototransistorVoltage 스케치는 Serial.print(vsolts(A3))의 전압함수에 의해 반환되는 값을 나타낸다.

HaltUnderBrightLight는 if(volts(A3) >3.5) 표현 내부의 값을 광선 아래에서 멈추기 위해 ABOT으로 가져가는데 사용한다.

이진수와 아날로그 그리고 디지털

이진수 센서는 일반적으로 어떤 것의 존재와 비존재를 의미하는 두 개의 다른 상태를 전송할 수 있다. 예를 들면 더듬이가 압력을 받지 않으면 HIGH신호를 압력을 받으면 LOW신호를 보내는 것처럼 말이다.

아날로그 센서는 연속적인 범위의 측정값에 상응하는 연속적인 범위의 값을 보낸다. 이번 실험에서의 포토트랜지스터 회로는 아날로그 센서의 예이다. 그것들은 연속적인 범위의 빛 수준에 상응하는 연속적인 값들을 제공한다.

디지털 값은 digit로 표현된 숫자이다. 컴퓨터와 마이크로컨트롤러가 아날로그 측정값을 디지털로 저장한다. 아날로그 센서로 측정하고 디지털 값으로 측정치를 저장하는 과정을 A/D변환(*analog to digital conversion*)이라고 한다. 이 측정을 디지털화 측정(*digitized* measurement)이라 하고, 아날로그를 디지털로 변환하는 문서를 양자화 측정(*quantized* measurements)이라고도 한다.

아두이노 맵 함수

PhototransistorVoltage 스케치에서 우리는 0-1023사이의 측정값을 0.0-4.995범위의 전압값으로 변환한다. 다른 응용 즉, 모터제어나 다른 분석을 위하여 또 다른 함수에 여러분의 스케치가 사용되어 측정값을 다른 범위의 정수 범위 값으로 변환할 필요가 생길 수 있다.

이런 경우 아두이노 맵 함수가 편리하다. 하나의 정수범위를 또 다른 범위의 값으로 대응시킨 값을 “mapping”이라고 한다. 예를 들어 0-1024범위 사이의 범위를 1300-1700사이의 서보 제어 범위로 매핑하기를 원한다고 해보자. 여러분에게 맵 함수를 어떻게 사용하는지를 보여주는 예제가 다음에 있다.

```
int adcVal = analogRead(A3);
int newAdcVal = map(adcVal, 0, 1023, 1300, 1700);
```

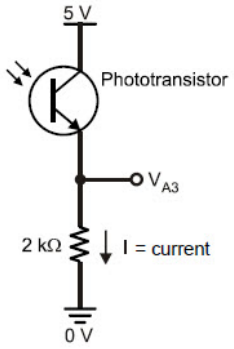
이번 예제에서, adcVal의 값이 512라면, newAdcVal 에 대한 맵함수의 결과는 1500이 된다. 그래서 0-1023범위의 중간지점으로부터 1300-1700 범위의 대응점으로 측정치가 대응된다.

포토트랜지스터 회로는 어떻게 동작 하는가

저항은 전류의 흐름을 방해하는 소자이다. 저항이 포함된 회로에서 전압은 수압과 연결하여 생각할 수 있다. 주어진 전류량에 대하여 적은 저항보다 큰 저항에서 전압강하가 크지만 저항을 통과하는 전류의 양은 동일하다. 만약 저항 값을 일정하게 유지하고 전류를 변화시킨다면, 여러분은 동일한 저항 양단에 더 많은 전류를 인가해서 더 큰 전압을 측정할 수 있고, 전류가 작아지면 전압도 작아지게 할 수 있다.

아두이노 아날로그 입력은 포토트랜지스터 회로에서 보이지 않는다. 그래서 A3는 영향을 받지 않는 회로를 모니터한다. 아래 회로를 살펴보자. 회로 위쪽의 5V와 아래쪽의 GND(0V)을 이용하여, ABOT의 배터리가 전자를 공급하여 5V 전압이 걸리도록 하기를 원한다.

A3 전압(V_{A3})이 빛에 따라 변하는 이유는 포토트랜지스터가 빛의 양이 많을 때 더 많은 전류가 흐르게 하고 빛의 양이 적을 때 더 적은 전류가 흐르게 하기 때문이다. 아래 회로에서 라벨I로 붙여진 전류가 저항을 통과하여 흐른다. 저항을 통하여 더 많은 전류가 흐르면, 저항 양단의 전압이 커질 것이다. 전류가 덜 흐르게 되면 전압은 낮아질 것이다. 저항의 한 쪽 끝이 Vss = 0V에 묶여져 있기 때문에, A3 전압은 전류가 증가하면 올라가고 전류가 작아지면 내려간다.



2kΩ 저항을 1kΩ 저항으로 교체한다면, V_{A3}는 동일한 전류에 대하여 더 작은 값이 될 것이다. 사실 동일한 전압 수준이 되기 위해서는 두 배의 전류가 되어야

할 것이다. 이것은 ABOT이 멈추도록 하는 HaltUnderBrightLight의 기준전압 3.5V 수준에 도달하기 위하여 빛의 밝기가 2배가 되어야 한다는 것을 의미한다.

그래서 포토트랜지스터와 직렬 연결되는 저항이 작아지면 빛에 덜 민감한 회로를 만드는 것이다. 만약 2kΩ 저항을 10kΩ 저항으로 교체한다면 V_{A3}는 5배 증가하고, 3.5V에 도달하기 위하여 1/5배 전류를 생성에 필요한 1/5배 빛이 필요할 것이다. 그래서 더 큰 저항은 회로가 빛에 더 민감하게 동작할 것이다.

직렬 연결 두개 또는 그 이상의 부품들이 끝과 끝으로 연결될 때, 직렬이라고 한다. 회로의 포토트랜지스터와 저항은 직렬로 연결되어 있다.

옴의 법칙

두 가지 양이 V_{A3}에 영향을 준다. 전류와 전압 그리고 옴의 법칙이 상호간의 동작원리를 설명한다. 옴의 법칙에 의하면 저항 양단의 전압이 저항을 통과하는 전류와 저항을 곱한 값과 같다. 그래서 여러분의 두 개의 값을 알고 있다면, 세 번째 값을 계산하기 위하여 옴의 법칙을 사용할 수 있다.

$$V = I \times R$$

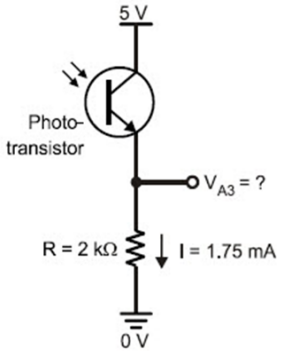
어떤 교재에서, 여러분은 E = I × R을 보게 될 것이다. E는 전압을 나타내고 "volts"의 또 다른 표현이다.

전압 V는 전압(volts)의 단위로 측정되고, 약자는 V이다. 전류 I는 암페어로 측정되고 약자는 A를 사용한다. 저항 R은 옴으로 측정되고 약자는 그리스 문자 오메가 (Ω)를 사용한다. 다음 회로에 흐르는 전류 수준은 밀리암페어(mA)이다. 소문자 m은 1/1000의 암페어 측정치를 나타내는 것이다. 비슷하게 kΩ에서의 소문자 k는 측정치가 옴의 1000배임을 나타낸다.

회로를 흐르는 두 개의 전류량에 대하여 포토트랜지스터에서의 V_{A3}를 옴의 법칙으로 계산해보자: 상당히 밝은 빛에 의한 결과로 일어나는 전류 1.75mA 그리고 덜 밝은 빛으로 일어나는 0.25mA

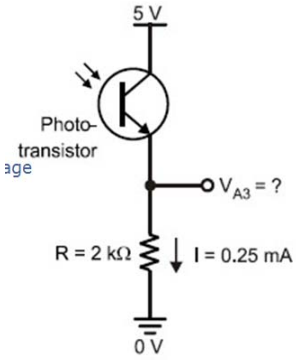
아래 예제는 해답을 보여준다. 여러분이 계산할 때, 밀리(m)는 천분의 1 그리고 킬로 (k)는 천이라는 것을 기억하고 옴의 법칙에 대입하시오.

예제 1: I = 1.75mA 그리고 R = 2kΩ



$$
\begin{aligned}
V_{A3} &= I \times R \\
&= 1.75\,mA \times 2\,k\Omega \\
&= \frac{1.75}{1000}\,A \times 2000\ \Omega \\
&= 1.75\,A \times 2\,\Omega \\
&= 3.5\,A\Omega \\
&= 3.5\,V
\end{aligned}
$$

예제 2: I = 0.25mA 그리고 R = 2kΩ



$$
\begin{aligned}
V_{A3} &= I \times R \\
&= 0.25\,mA \times 2\,k\Omega \\
&= \frac{0.25}{1000}\,A \times 2000\ \Omega \\
&= 0.25\,A \times 2\,\Omega \\
&= 0.5\,A\Omega \\
&= 0.5\,V
\end{aligned}
$$

옴의 법칙과 저항 조정

방안의 분위기 광선이 V_{A3} = 3.5 V 이고, 이 광선이 어두움에 대한 광선 0.5V를 만드는 불빛의 2배만큼 밝다고 말해보자. 더 높은 전류가 흐를 수 있는 또 다른 상황은 분위기 광선이 적외선보다 더 강한 경우이다. 어느 경우이든 포토트랜지스터는 회로를 통해 흐르는 전류의 두 배일 수 있고 측정의 어려움을 초래한다.

질문: 여러분은 회로의 전압응답을 밝은 빛의 3.5V로 그리고 어두운 빛의 0.5V로 낮추기 위하여 무엇을 할 수 있습니까?

대답: 저항 값을 반으로 낮추시오. 2kΩ저항을 1kΩ저항으로 바꾸시오.

- R = 1kΩ과 밝은 빛 전류 I = 3.5mA 와 어두운 빛 전류 I = 0.5mA을 사용하여 옴의 법칙 계산을 반복하시오. 두 배의 전류를 사용하여 밝은 빛에 대하여 3.5V 어두운 빛에서 0.5V를 나타낼 수 있습니까?

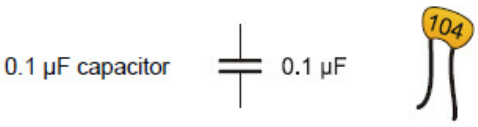
과제 2: 더 넓은 범위에서 빛의 레벨 측정하기

이전 과제에서 회로는 단지 제한된 빛의 범위에서 동작한다. 여러분은 실습1 회로가 잘 동작하는 하나의 방에서 계측을 하고 이어서 더 밝은 방으로 그것을 가져가면 모든 전압 측정값이 최대값을 나타내는 것을 발견할 것이다. 또는 더 어두운 방으로 가져가면 전압이 0.1V도 만들지 못할 것이다.

이 과제는 아두이노가 더 넓은 범위의 광선 수준을 측정하는데 사용할 수 있는 다른 포토트랜지스터 회로를 소개한다. 이 회로와 스케치는 0부터 75,000까지의 범위 값을 반환할 수 있다. 더 작은 값은 밝은 빛을 나타내고 더 큰 값은 어두운 빛을 나타낸다.

커패시터(Capacitor)란?

커패시터는 전하를 충전하는 소자이고, 많은 회로의 기초 구성요소이다. 또한 배터리는 전하는 축적하는 소자이고, 이번 과제를 통하여 충전되고 방전되고 다시 재충전되는 얇은 배터리와 동등하게 커패시터를 생각할 수 있다.



커패시터가 얼마나 많은 전하를 충전할 수 있는지가 패러데이(F)로 측정된다. 1 패러데이는 ABOT 회로에서 사용하는데 실용적이지 않을 만큼 매우 큰 값이다. 여러분 키트의 커패시터는 백만분의 1 비율로 저장한다. 백만분의 1 패러데이를 마이크로 패러데이(*microfarad*)로 부른다. 그리고 약자는 µF이다. 0.1 µF는 백만분의 1의 십분의 1을 저장한다.

커패시턴스 측정

microfarads: (millionths of a farad), abbreviated µF $1\ \mu F = 1\times10^{-6}$ F

nanofarads: (billionths of a farad), abbreviated nF $1\ nF = 1\times10^{-9}$ F

picofarads: (trillionths of a farad), abbreviated pF $1\ pF = 1\times10^{-12}$ F

0.1 μF커패시터의 겉표면에서 104는 피코패러데이(picofarads : pF) 측정치이다. 여기서 104는 10에 0을 4개 더하는 것을 나타내고, 그래서 커패시터는 100,000 pF 즉 0.1 μF이 된다.

$(100{,}000) \times (1 \times 10^{-12})$ F = $(100 \times 10^{3}) \times (1 \times 10^{-12})$ F

$= 100 \times 10^{-9}$ F = 0.1×10^{-6} F

= 0.1 μF.

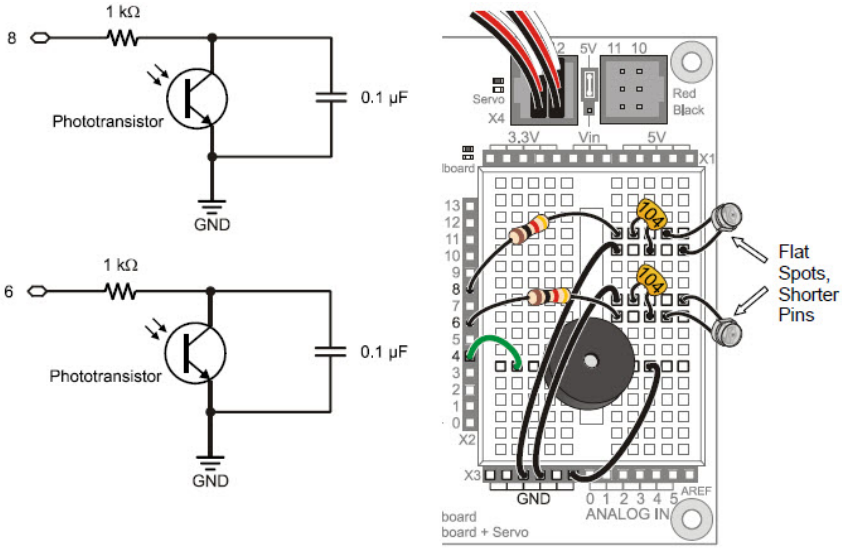
광 감지 눈 설치하기

다음 회로는 각각의 포토트랜지스터에 도달하는 광선 수준과 독립적으로 반응할 수 있다. 하나는 좌측전방 그리고 다른 하나는 우측전방으로 그것들이 약 45°위로 향하도록 설치될 것이다. 두 개 포토트랜지스터들의 값을 모니터하는 스케치는 ABOT의 어느 방향이 더 밝은 빛인지를 결정할 수 있다. 그래서 이 정보는 길 찾기 결정 수단으로 이용할 수 있다.

부품 목록

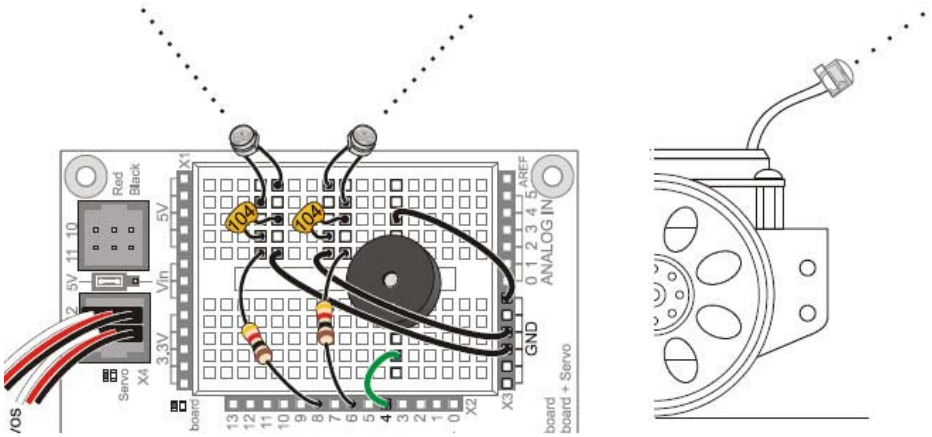
(2) 포토트랜지스터(phototransistors)
(2) 커패시터(capacitors), 0.1µF (104)
(2) 저항(resistors), 1kΩ (갈색-검정-빨강)
(2) 점퍼 와이어(jumper wires)

- 배터리와 프로그래밍 케이블을 여러분 보드와 분리하시오.
- 이전 포토트랜지스터 회로를 제거하고, 아래 회로를 설치하시오.
- 여러분의 포토트랜지스터가 반대로 연결되지 않고 단자가 서로 닿지 않도록 배선도에 따라 회로를 점검하시오.



이 장의 예제는 다른 방향에 대한 빛의 세기 차이를 감지하기 위하여 위쪽 방향과 바깥방향으로 향하는 포토트랜지스터에 의존할 것이다.

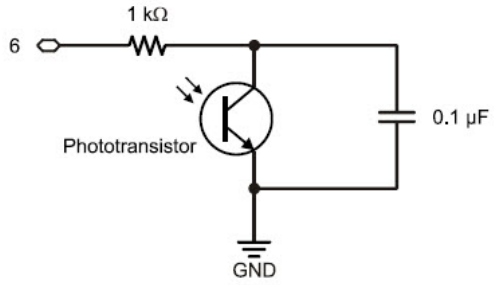
- 아래 그림처럼 포토트랜지스터를 브레드보드로부터 45°위쪽으로 향하도록 조정하고 약 90°바깥방향으로 조정하시오.



전하 이동과 포토트랜지스터 회로

다음 회로의 커패시터를 얇은 재충전식 배터리라고 생각하고, 포토트랜지스터를 빛으로 제어하는 전류 밸브라고 생각하시오. 각각의 커패시터는 5V로 충전될 수 있고, 그 다음 포토트랜지스터를 통하여 배출될 수 있다. 커패시터가 전하를 잃어버리는 비율은 얼마나 많은 전류가 포토트랜지스터를 통과하는지에 의존하고, 포토트랜지스터 베이스에 비추는 빛의 밝기에 의존한다. 다시 더 밝은 빛은 더 많은 전류를 흐르게 하고, 더 어두운 빛은 전류를 덜 흐르게 한다.

이러한 종류의 포토트랜지스터/커패시터를 전하이동(*charge transfer*) 회로라고 부른다. 아두이노는 커패시터 전압이 감소하는데 즉 어떤 전압 값 아래로 떨어지는데 얼마나 긴 시간이 걸리는지를 측정함으로써 각각의 커패시터와 연결된 포토트랜지스터를 통하여 전하 손실이 일어나는 비율을 결정할 것이다. 감쇠시간은 포토트랜지스터의 베이스에 도달하는 빛의 밝기에 따라 제어되는 전류 밸브를 얼마나 넓게 여는가에 부합한다. 더 많은 빛은 더 빠른 감쇠를 의미하고, 더 작은 빛은 천천히 감쇠하는 것을 의미한다.

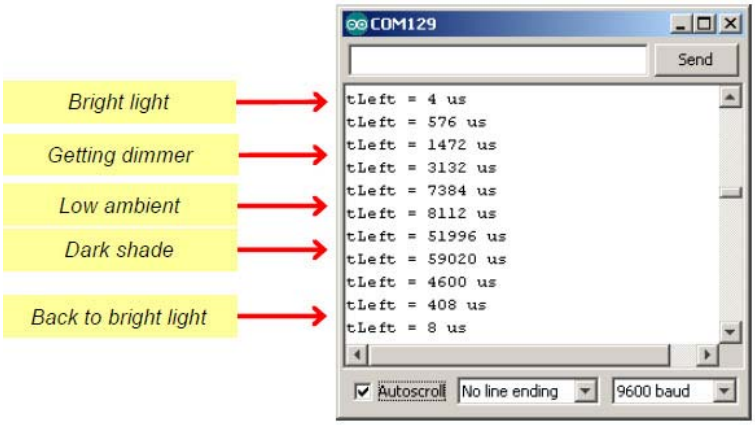


QT Circuit: 전하 이동이 약자는 QT이다. 문자 Q는 전기적 전하이고 T는 이동을 나타낸다.

병렬 연결: 그림 6-11에 나타난 포토트랜지스터와 커패시터는 병렬로 연결되어 있다. 각 단자의 하나는 공통 단자(또한 노드로 부른다)로 연결된다. 포토트랜지스터와 커패시터의 하나 단자는 GND에 연결된 하나의 단자를 가진다. 그리고 다른 하나의 단자는 동일한 1kΩ 저항단자에 연결된다.

포토트랜지스터 회로 테스터하기

LeftLightSensor 스케치는 핀 8번 QT 회로에서 커패시터를 충전하고, 전압 감쇠 시간을 측정하고, 직렬 모니터에서 그것을 보여준다. 다음 회로와 스케치에서, 더 낮은 숫자가 더 밝은 빛을 의미한다는 점을 기억하시오.



우리는 여러분이 다른 분위기 광선 수준을 위한 올바른 저항을 찾을 걱정없이 ABOT을 하나의 방에서 또 다른 방으로 옮길 수 있도록 이 장의 나머지 부분에서 광 감지기술을 사용할 것이다.

- 직사광선이 창문을 통해 안으로 비친다면 창문을 닫으시오.
- LeftLightSensor 입력하고 업로드하고 직렬모니터를 여시오.
- 직렬모니터에 디스플레이된 값을 기록하시오.
- 직렬모니터가 값을 디스플레이하지 못하거나 한 두개 이후에 멈춘다면, 회로에 에러가 있을 수 있다. 이러한 징후를 알게 되면 배선을 점검하고 다시 시도하시오.
- 포토트랜지스터 회로 8번 핀 위에서 그림자를 만들기 위해 여러분 손을 사용하거나 책을 사용하세요.
- 다시 직렬모니터에서 측정값을 점검하시오. 그 값은 처음 값보다 더 커져야 합니다. 또한 그것을 기록하시오.
- 직렬모니터에 출력이 없다면, 또는 빛의 수준에 무관하게 하나의 값에 멈춰있다면, 배선 에러일 수 있다. 회로를 점검하시오.
- 그림자를 더 어둡게 만들기 위하여 포토트랜지스터 위로 더 가깝게 어떤 물체를 움직이시오. 측정값을 기록하시오.
- 강하게 그림자를 만들거나 심지어 손으로 감싸쥐어서 그림자를 만들고 측정하시오.

```
/*
 * Robotics with the BOE Shield - LeftLightSensor
 * Measures and displays microsecond decay time for left light sensor.
 */

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
}
void loop()                 // Main loop auto-repeats
{
  long tLeft = rcTime(8);         // Left rcTime -> tLeft

  Serial.print("tLeft = ");       // Display tLeft label
  Serial.print(tLeft);            // Display tLeft value
  Serial.println(" us");          // Display tLeft units + newline

  delay(1000);                    // 1 second delay
}
                                  // rcTime function at pin
long rcTime(int pin)              // ..returns decay time
{
  pinMode(pin, OUTPUT);       // Charge capacitor
  digitalWrite(pin, HIGH);    // ..by setting pin ouput-high
  delay(1);                   // ..for 5 ms
  pinMode(pin, INPUT);        // Set pin to input
  digitalWrite(pin, LOW);     // ..with no pullup
  long time  = micros();      // Mark the time
  while(digitalRead(pin));    // Wait for voltage < threshold
  time = micros() - time;     // Calculate decay time
  return time;                // Return decay time
}
```

다른 포토트랜지스터 회로를 테스트하기

조종 동작을 하기 전에 여러분은 불빛에 대하여 오른쪽 핀 6번 광센서 회로에 대하여 동일한 테스트를 실행할 필요가 있을 것이다. 조종동작 이전에 두 개의 회로가 잘 동작해야한다.

- rcTime을 호출할 때 핀 변수를 8에서 6으로 변경하시오.
- tLeft의 모든 인스턴스를 tRight로 바꾸시오.
- 스케치를 동작시키고, 핀 6번 광센서가 동작하는지 점검하시오.

두 개의 포토트랜지스터 회로를 함께 테스트할 스케치를 별도로 갖는 것이 또한 좋을 것이다.

- BothLightSensors로 스케치를 다시 저장하고, 코멘트를 갱신하시오.
- loop 함수를 아래 것으로 바꾸시오.
- 한쪽이 방안 가장 밝은 광원 방향을 가르키고 다른 한 쪽이 방안의 밝은 광원을 벗어나는 방향을 가르킬 때까지 BOE 쉴드-봇을 회전시키시오. 직렬모니터의 tLeft 와 tRight 사이에서 여러분이 얻을 수 있는 가장 큰 차이는 무엇입니까?

```
void loop()                             // Main loop auto-repeats
{
  long tLeft = rcTime(8);               // Left rcTime -> tLeft
  Serial.print("tLeft = ");             // Display tLeft label
  Serial.print(tLeft);                  // Display tLeft value
  Serial.print(" us       ");           // Display tLeft units

  long tRight = rcTime(6);              // Left rcTime -> tRight
  Serial.print("tRight = ");            // Display tRight label
  Serial.print(tRight);                 // Display tRight value
  Serial.println(" us");                // Display tRight units + newline
  delay(1000);                          // 1 second delay
}
```

rcTime과 전압 감쇠

빛의 양이 낮을 때, rcTime함수는 정수 변수 심지어 워드 변수로 저장하는데 많은 시간이 걸린다. 다음 단계로 저장용량을 늘리면 -2,147,483,648 부터 2,147,483,647 까지 값을 저장할 수 있다. 그래서 long rcTime(int *pin*) 함수 정의는 함수가 실행되고 long 값을 반환한다. 또한 측정할 핀 번호가 어느 핀인지를 알 필요가 있다.

long rcTime(int *pin*)

전하 이동(charge transfer) 측정은 7단계이다. : (1) 커패시터를 충전하기 위하여 I/O 핀을 HIGH로 놓으시오. (2) 커패시터가 충분히 충전될 만큼 기다리시오. (3) I/O 핀을 입력으로 바꾸시오. (4) 시간을 체크하시오. (5) 아두이노의 2.1V 문턱 전압 아래로 전압이 내려가도록 기다리시오. (6) 다시 시간을 체크하시오. (7) 단계-6시간에서 단계-3시간을 빼시오. 그것이 감쇠 시간이다.

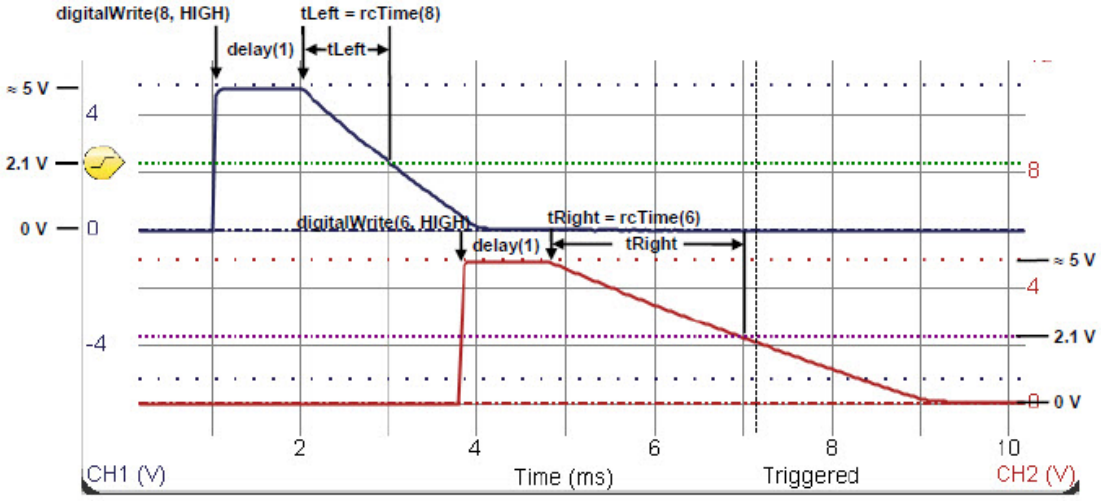
```
{
  pinMode(pin, OUTPUT);           // Step 1, part 1
  digitalWrite(pin, HIGH);       // Step 1, part 2
  delay(1);                        // Step 2
  pinMode(pin, INPUT);            // Step 3  part 1
  digitalWrite(pin, LOW);        // Step 3, part 2
  long time  = micros();          // Step 4
  while(digitalRead(pin));       // Step 5
  time = micros() - time;         // Step 6 & 7
  return time;
}
```

이번 스케치에서, 단계-1은 두 개의 하위 단계로 구성된다. 먼저 pinMode(pin, OUPUT)은 I/O핀을 출력으로 설정하고 다음에 digitalWrite(pin, HIGH)가 회로에 5V를 공급하도록 한다. 단계-3 또한 I/O핀이 HIGH신호를 전송하기 때문에 두 개의 하위 단계로 구성된다. 스케치가 I/O핀의 방향을 output-high 에서 input

으로 바꿀 때, 회로에서 10kΩ의 저항을 더하고, 이후 그 저항이 제거되어야 한다. pinMode(pin, INPUT)가 저항을 제거한 이후 digitalWrite(pin, LOW)를 더하는 것은 커패시터가 포토트랜지스터를 통하여 전하를 방출하도록 하기 위함이다.

고급 주제 선택: 전압 감쇠 그래프

아래 그래프는 BothLightSensors 스케치가 실행되는 동안 BOE 쉴드-봇의 좌측 우측 QT 회로 전압응답을 보여준다. 시간에 따른 응답전압을 측정하고 그래프해 주는 장치를 오실로스코프(*oscilloscope*)라고 부른다. 두 개 전압신호를 그래프하는 두 개의 측정선을 파형(trace)이라고 부른다. 위쪽 파형에 대한 전압 배율은 왼쪽을 따르고, 아래쪽 파형에 대한 전압 배율은 오른쪽을 따른다. 두 개 파형이 시간 배율은 아래쪽을 따른다. 각각 파형위에 BothLightSensors에서 명령이 실행될 때 마다 표시를 보여준다. 그래서 여러분은 전압신호가 어떻게 응답하는지를 알 수 있다.



위 그래프 파형은 핀 8번 QT회로의 커패시터 전압을 그린 것이다. 이것은 왼쪽 광센서이다. digitalWrite(8, HIGH)의 응답으로 전압은 0V부터 거의 5V까지 1ms 표시에서 빠르게 올라간다. 그 다음 2ms 표시에서 rcTime 호출이 감소가 시작되도록 한다. rcTime함수는 전압이 약 2.1V 부근까지 감쇠하는데 걸리는 시간을

측정하고, tLeft변수에 그것을 저장한다. 그래프에서는 감쇠가 약 1ms 정도 걸리는 것처럼 보이고, 그래서 tLeft 변수는 1000에 가까운 값을 저장해야 한다.

그래프에서 아래 파형은 핀 6번 QT 회로의 커패시터 전압(오른쪽 광센서)을 그린다. 측정은 왼쪽 센서 측정이 종료된 이후 시작한다. 전압은 감쇠시간이 약 2ms 걸리는 것만 제외하고 위쪽 파형과 유사한 형태로 변화한다. 우리는 하고, tRight가 2000가까운 값을 저장하는 것을 보게 될 것으로 기대한다. 이 값이 더 커지면 더 늦은 감쇠에 대응하고, 더불어 더 낮은 광 수준에 대응하는 것이다.

실습3: 로밍(Roaming)을 위한 빛 측정

우리는 이제 다양한 광원조건에서 동작할 수 있는 회로를 가지고 있다. 우리는 또한 선택할 수 있는 어떤 코드를 필요로 한다. 변화를 선택할 수 없는 스케치 코드의 예제가 다음과 같다:

```
if(tLeft > 2500)…              // Not good for navigation.
```

만약 문장이 방안의 그림자를 물리치고 잘 동작한다고 하더라도, 또 다른 더 밝은 빛을 만나면 그림자를 결코 감지하지 못할 것이다. 그렇지 않고 더 어두운 방으로 옮기더라도 오랜 시간동안 찾지 못할 것이다. 조종을 위하여 문제는 각각의 센서에 광 수준을 알려주는 실제 숫자가 아니라는 점이다. 문제는 두개 센서가 많은 빛을 감지하는 차이이고 따라서 로봇은 센서가 더 밝은 빛을 찾는 방향으로 회전할 수 있다. (혹은 여러분이 원하는 바에 따라 더 밝은 빛으로부터 멀어질 수 있다.)

해답은 간단하다. 오른쪽 센서 측정치를 두 개의 합으로부터 나누시오. 여러분의 결과는 항상 0과 1의 범위 내에 있게 될 것이다. 이 기술은 정규화 미분(*normalized differential*) 측정의 예제이다. 여기서 다음 식으로 표현할 수 있다.

$$normalized\ differential\ shade = \frac{tRight}{tRight + tLeft}$$

예를 들어 0.25의 정규화 미분측정은 오른쪽 센서의 밝기가 왼쪽 밝기의 1/2이라는 것을 의미한다. tRight 와 tLeft 의 실제 값은 밝은 방에서는 작아지고 어두운 방에서는 커질 것이지만, 왼쪽 빛이 오른쪽 센서에 대한 밝기의 1/2배이면 해답은 여전히 0.25가 될 것이다. 0.5의 측정값은 tRight 와 tLeft 의 값이 동일하다는 것을 의미한다. 둘 다 함께 커질 수 있고 작아질 수 있지만, 결과가 0.5라면 동일한 밝기의 광수준을 감지하는 것을 의미한다.

여기에 또 다른 시도가 있다. 정규화 미분 측정으로부터 0.5를 빼보시오. 그러면

결과 값이 0과 1사이 대신에 -0.5에서 +0.5 범위의 값으로 변화하고, 측정값 0은 서로 동일한 밝기를 의미한다. 이러한 결과는 영점 조정된 정규화 미분 측정치(*zero-justified* normalized differential shade measurement)이다.

$$zero\ justified\ normalized\ differential\ shade = \frac{tRight}{tRight + tLeft} - 0.5$$

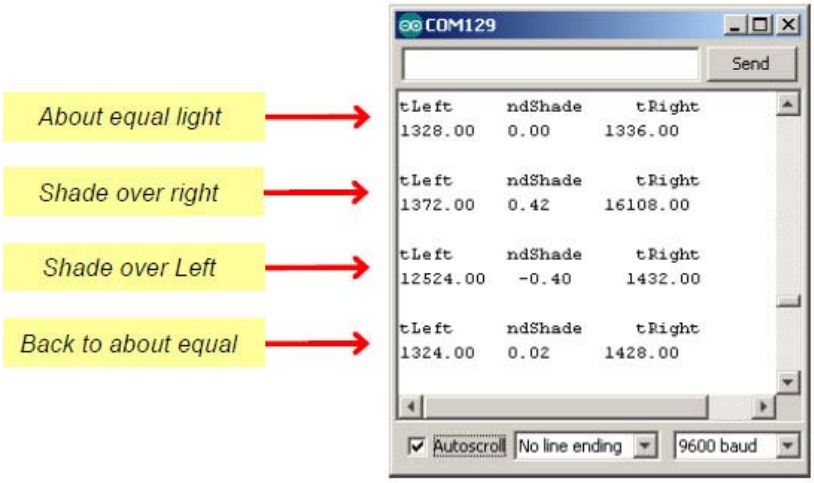
그러나 왜 그렇게 하는가? 양수와 음수가 바퀴 속도 조절에 사용될 수 있기 때문에 -0.5와 +0.5범위의 값이 조정을 위한 스케치를 위해 더 좋다. 여기서 영점 조정된 미분 측정치가 다음 스케치에 소개된다.

```
float ndShade;                          // Normalized differential shade
ndShade = tRight / (tLeft + tRight) - 0.5; // Calculate it and subtract 0.5
```

마지막 측정치는 ndShade로 이름 붙여진 유동 소수점 변수에 저장될 것이고, 그래서 먼저 선언되어진다. 다음 줄에서 영점 조정된 미분 측정치 연산이 실행된다. 결과값은 tLeft에 비하여 tRight가 감지하는 전체 그림자 비율을 나타내는 -0.5에서 0.5사이의 값이 될 것이다. ndShade가 0일 때, 그것은 tRight 와 tLeft가 동일한 값이라는 것을 의미하고 그래서 센서들이 동일한 밝기의 불빛을 감지한다는 것을 의미한다. ndShade가 -0.5에 가까워질수록 우측 센서가 더 어두운 그림자이고, ndShade가 0.5에 가까워지면 좌측 센서의 그림자가 더 어둡다는 것이다. 이것은 조종할 때 매우 유용할 것이다. 먼저 직렬 모니터로 그것을 테스트해보자.

예제 스케치: LightSensorValues

다음 화면캡쳐는 LightSensorValues 스케치가 실행되는 직렬모니터 예제를 나타낸다. 우측센서 위의 그림자 때문에 ndShade값이 약 0.4이고, 좌측 센서 위의 그림자로는 ndShade 값이 약 -0.4가 된다.



- 근처 창문에서 직사광선이 들어오지 않는가를 확인하시오. 실내 불빛이 좋고, 직사광선은 여전히 센서가 과동작하는 상태가 될 것이다.
- ABOT의 좌측 센서위에 그림자를 드리울 때 음수가 나오는지를 점검하고, 더 어둡게하면 더 큰 음수로 나타나는지를 점검하시오.
- ABOT의 우측 센서위에 그림자를 드리울 때 양수가 나오는지를 점검하고, 더 어둡게하면 더 큰 양수로 나타나는지를 점검하시오.
- 두 개의 센서가 동일한 수준의 불빛이나 그림자에 대하여 ndShade 값이 0에 가까운지를 점검하시오.
- 두 개 센서 위에 동일한 그림자를 만드시오. 전체 불빛의 수준이 내려가더라도 ndShade의 값은 여전히 0에 가깝게 머물러 있어야 합니다.

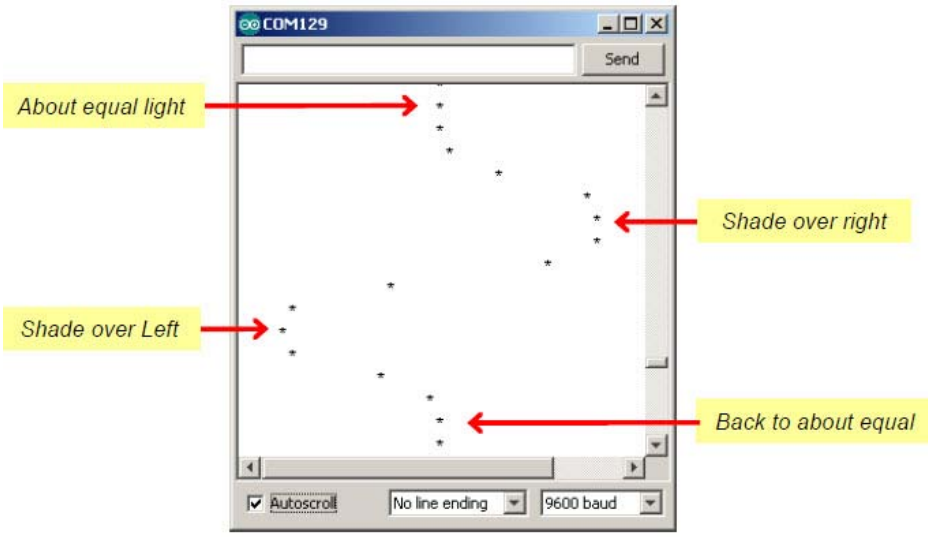
```
/*
 * Robotics with the BOE Shield - LightSensorValues
 * Displays tLeft, ndShade and tRight in the Serial Monitor.
 */
```

```
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
}
```

```
void loop()                    // Main loop auto-repeats
{
  float tLeft = float(rcTime(8));      // Get left light & make float
  float tRight = float(rcTime(6));     // Get right light & make float

  float ndShade;           // Normalized differential shade
  ndShade = tRight / (tLeft+tRight)-0.5;//Calculate it and subtract 0.5

  // Display heading
  Serial.println("tLeft       ndShade       tRight");

  Serial.print(tLeft);                // Display tLeft value
  Serial.print("    ");                 // Display spaces
  Serial.print(ndShade);              // Display ndShade value
  Serial.print("       ");              // Display more spaces
  Serial.println(tRight);             // Display tRight value
  Serial.println(' ');                // Add an extra newline

  delay(1000);                        // 1 second delay
}

long rcTime(int pin)               // rcTime measures decay at pin
{
  pinMode(pin, OUTPUT);             // Charge capacitor
  digitalWrite(pin, HIGH);          // ..by setting pin ouput-high
  delay(5);                         // ..for 5 ms
  pinMode(pin, INPUT);              // Set pin to input
  digitalWrite(pin, LOW);           // ..with no pullup
  long time  = micros();            // Mark the time
  while(digitalRead(pin));          // Wait for voltage < threshold
  time = micros() - time;           // Calculate decay time
  return time;                      // Returns decay time
}
```

측정 빛의 그래프 표시

아래 직렬모니터 캡쳐 화면은 ndShade 변수 값을 그림으로 나타낸 예제이다. 별표(asterisk)는 빛이나 그림자가 두 개 센서에 걸쳐서 같다면 -0.5와 0.5사이 중간점에 있을 것이다. ABOT의 우측센서 위의 그림자가 더 어둡다면 별표는 스케일에서 우측으로 자리할 것이다. 좌측센서 위의 그림자가 더 어둡다면 별표는 좌측으로 위치할 것이다. 그림자/불빛의 대조비가 더 커지는 것은 별표 위치를 중심으로부터 더 멀게 할 것이다.



- LightSensorDisplay 스케치를 아두이노로 로드하시오.
- 각각의 광센서 위로 서로 다른 그림자를 드리우도록 시도해 보고, 직렬모니터에서의 별표가 어떻게 응답하는지를 살펴보시오. 여러분이 두 개 센서위로 동일한 그림자를 드리우면 별표는 여전히 중간에 있어야 한다는 점을 명심하세요. 두 개 센서 사이에서 차이가 있다면 센서가 더 어두운 것을 인식한다는 것을 가르킨다.

```
/*
 * Robotics with the BOE Shield - LightSensorDisplay
 * Displays a scrolling graph of ndShade.  The asterisk positions ranges
```

```
* from 0 to 40 with 20 (middle of the display) indicating same light on
* both sides.
*/

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
}

void loop()                     // Main loop auto-repeats
{
  float tLeft = float(rcTime(8));     // Get left light & make float
  float tRight = float(rcTime(6));    // Get right light & make float

  float ndShade;                // Normalized differential shade
  ndShade = tRight / (tLeft+tRight) - 0.5; //Calculate it and subtract 0.5

  for(int i = 0; i<(ndShade * 40) + 20; i++) //Place asterisk in 0 to 40
  {
    Serial.print(' ');          // Pad (ndShade * 40) + 20 spaces
  }
  Serial.println('*');                  // Print asterisk and newline

  delay(100);                           // 0.1 second delay
}

long rcTime(int pin)                // rcTime measures decay at pin
{
  pinMode(pin, OUTPUT);               // Charge capacitor
  digitalWrite(pin, HIGH);          // ..by setting pin ouput-high
  delay(5);                           // ..for 5 ms
  pinMode(pin, INPUT);              // Set pin to input
```

```
  digitalWrite(pin, LOW);          // ..with no pullup
  long time  = micros();            // Mark the time
  while(digitalRead(pin));         // Wait for voltage < threshold
  time = micros() - time;           // Calculate decay time
  return time;                      // Returns decay time
}
```

LightSensorDisplay의 동작 방법

loop 함수는 좌측과 우측 광센서에 대한 rcTime 측정을 반복하고 그것들을 tLeft와 tRight에 저장한다.

```
void loop()                                 // Main loop auto-repeats
{
  float tLeft = float(rcTime(8));           // Get left light & make float
  float tRight = float(rcTime(6));          // Get right light & make float
```

ndShade가 유동 소수점 변수로 선언된 후, tLeft 와 tRight가 영점 조정된 정규화 미분 측정값을 얻기 위한 표현으로 사용된다. 그 결과는 -0.5와 +0.5 사이에 존재할 것이고 ndShade에 저장된다.

```
  float ndShade;                            // Normalized differential shade
  ndShade = tRight / (tLeft+tRight) - 0.5; // Calculate it and subtract 0.5
```

다음에, 별표를 프린트하기 위하여 loop 동안 우측 위치에 커서를 둔다. ndShade 값을 가져와서 40을 곱한다. 또한 ndShade가 -0.5라면 우리가 앞쪽 20칸을 남겨두고 프린트하기 위하여 ndShade 값에 20칸을 더해야 한다. 그래서 (-0.5 × 40) + 20 = 0. 이제 ndShade가 0이라면, 우리는 20칸을 남기기를 원한다. 그래서 (0 × 40) + 20 = 20. 만약 +0.5라면 우리는 40칸을 남기고 프린트하고 싶다. (0.5 × 40) + 20 = 40. 물론 0.25처럼 사이에 어떤 것이 있다면 우리는 (0.25 × 40) + 20 = 30이다. 그래서 중간과 맨 오른쪽 사이의 절반을 프린트할 것이다.

```
  for(int i = 0; i<(ndShade * 40) + 20; i++) // Place asterisk in 0 to 40
  {
    Serial.print(' ');                  // Pad (ndShade * 40) + 20 spaces
  }
```

빈 칸을 프린트한 후에 한 개 별표가 선위에 프린트된다. println이 프린트하고 반복을 통하여 별표가 다음 줄에 표시되도록 하기 위하여 새로운 줄을 더한다.

```
  Serial.println('*');                   // Print asterisk and newline

  delay(100);                            // 0.1 second delay
}
```

실습4: 광선 추적 루틴 점검

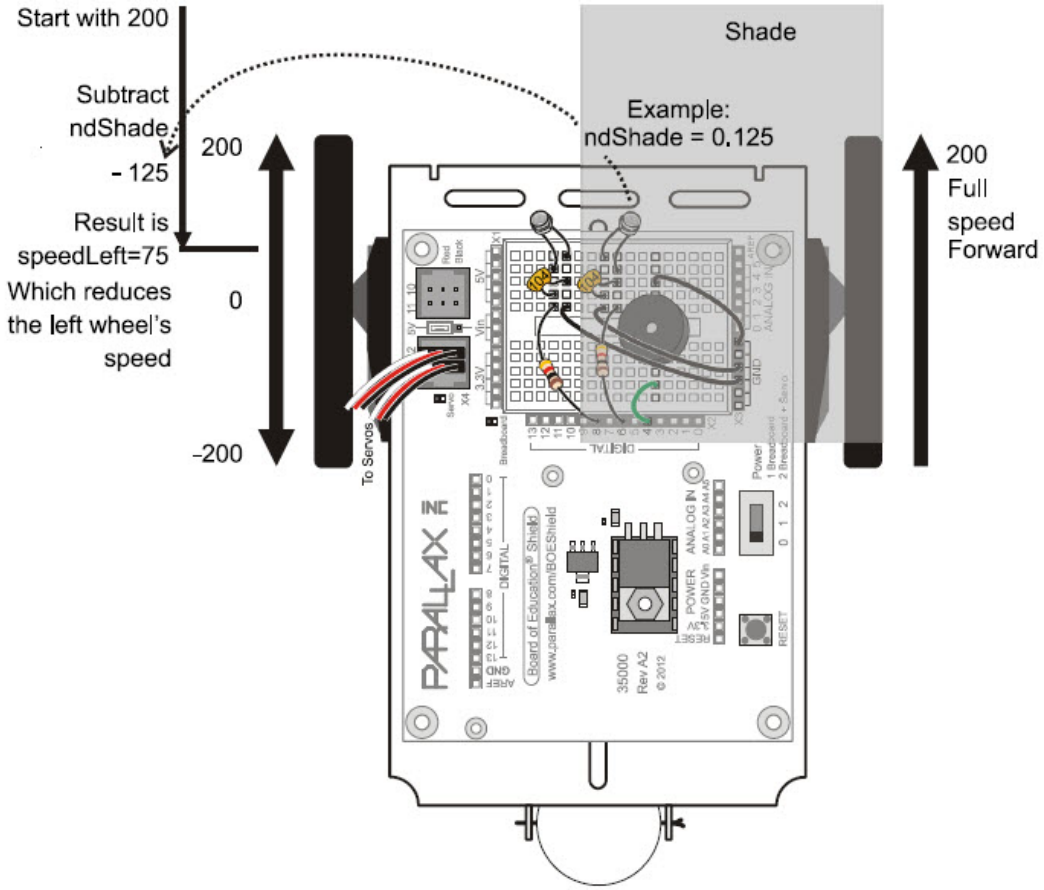
ABOT이 광선 방향으로 추적하도록 만드는 한 가지 접근은 그림자를 거부하도록 만드는 것이다. 여러분은 ABOT의 측면으로 감지되는 광선사이의 차이가 작거나 또는 많을 때 ABOT이 조금 또는 많이 회전하도록 만들기 위한 ndShade 변수를 사용할 수 있다.

그늘진 곳의 조종 결정

ABOT의 우측 그림자로부터 멀어지도록 회전을 잘하게 하는 If 문장이 여기에 있다. 프로그램은 두 개의 int 변수 speedLeft 와 speedRight를 선언하는 것으로부터 시작한다. 그것들은 loop 함수에서 다른 블록들이 그들의 값을 검사할 것이기 때문에 if…else블록 내에서는 선언되지 않는다. 다음에 if(ndShade >0.0)는 로봇의 우측 측면으로 그림자가 감지되면 ABOT이 어둠으로부터 멀어지게 회전하도록 왼쪽 바퀴를 천천히 속도를 낮추는 것을 실행할 코드블록을 가지고 있다. 이것을 하기 위해 ndShade * 1000.0는 200으로부터 차감된다. speedLeft로 그 결과를 할당하기 이전에, int(200.0-(ndShade×1000.0)는 유동 소수점 해답을 정수형으로 바꾼다. 정수형 값을 필요로 하는 4장 동작함수와 호환되는 값을 만들기 위해 우리는 다음과 같이 실시한다.

```
int speedLeft, speedRight;                    // Declare speed variables

if (ndShade > 0.0)                            // Shade on right?
{                                             // Slow down left wheel
  speedLeft = int(200.0 - (ndShade * 1000.0));
  speedLeft = constrain(speedLeft, -200, 200);
  speedRight = 200;                           // Full speed right wheel
}
```

다음 도면은 ndShade가 0.125일 때 어떻게 동작하는지를 예제로 보여준다. 왼쪽 바퀴는 200 - (0.125×1000) = 75이기 때문에 속도가 느려진다. 선형 속도 제어가 100에서 -100사이 범위에 있기 때문에 전속의 약 3/4으로 바퀴를 구동한다. 한편 speedRight는 전속 전진을 위하여 200으로 놓는다.



ndShade가 커지면 커질수록, 그것은 더 많이 200으로부터 차감한다. 다음 예제에서 문제는 없다. 그러나 ndShade가 0.45이라면, 그것은 speedLeft 변수에 -250을 저장하려고 할 것이다. 우리가 원하는 속도가 -200과 200까지의 범위내에 있도록 동작(maneuver)함수를 요구하기 때문에, 우리는 speedLeft가 제한구역을 벗어나지 않도록 아두이노의 constrain 함수를 사용할 것이다. speedLeft = constrain(speedLeft, -200, 200).

여기에 좌측의 그림자로부터 멀어지게 잘 회전하는 또 다른 문장이 있다. 그것은 우측 바퀴를 천천히 회전하게 하고 좌측 바퀴는 전속으로 전진하도록 한다. (ndShade*1000)에 200을 더한다. 이런 이유로 다음이 if(ndShade >0.0)을 위한

그 밖의 문장이고, 그래서 그것은 ndShade가 0과 같거나 더 작을 때 사용될 것이다. ndShade가 -0.125이면, speedRight = int(200.0 + (ndShade * 1000.0)) 는 200 + (-1.25 × 1000) = 200 - 125 = 75로 평가할 것이다. constrain함수는 speedRight를 제한하기 위하여 다시 사용된다.

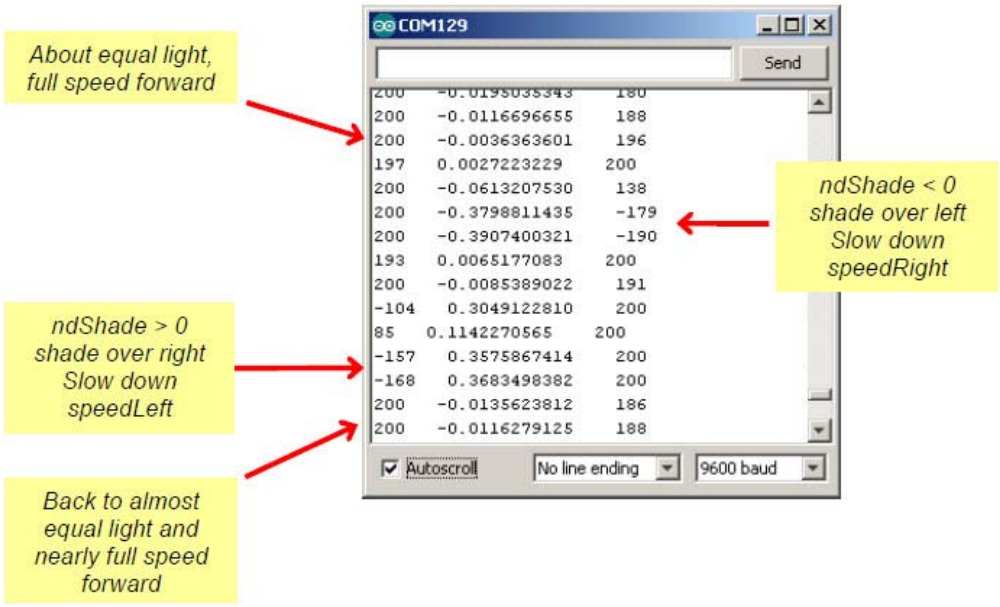
```
  else                                          // Shade on Left?
  {                                             // Slow down right wheel
    speedRight = int(200.0 + (ndShade * 1000.0));
    speedRight = constrain(speedRight, -200, 200);
    speedLeft = 200;                            // Full speed left wheel
  }
```

직렬 모니터로 조정결정 테스트

실제로 조정결정을 테스트하기 전에 직렬모니터로 변수 값을 살펴보는 것이 최선이다. 그래서 동작 함수를 호출하지 않고, 먼저 올바른 값을 갖는지를 알아보기 위하여 Serial.print 호출을 사용해보자.

```
  Serial.print(speedLeft, DEC);               // Display speedLeft
  Serial.print("    ");                       // Spaces
  Serial.print(ndShade, DEC);                 // Display ndShade
  Serial.print("     ");                      // More spaces
  Serial.println(speedRight, DEC);            // Display speedRight

  delay(2000);                                // 1 second delay
}
```

print와 println의 호출은 좌측 열에서 speedLeft의 값을 보여주는 화면으로 나타나고, 우측 열에서 speedRight, 그리고 두 개 사이에서 ndShade를 보여주도록 한다. 주의깊게 살피시오. 더 밝은 빛 측면은 항상 전속력 전진을 위해 200을 표시할 것이고, 다른 한 쪽은 200보다 더 작은 값으로 속도가 느려질 것이다. (그림자가 어두울수록 더 작은 값이 됨)



예제 스케치 - Light Seeking Display

- 전원 스위치를 1에 놓았는지 확인하시오.
- LightSeekingDisplay 스케치를 입력하고 저장하고 업로드 하시오.
- 직렬모니터를 여시오.
- ABOT의 우측 광센서 위에 서로 다른 그림자를 만들어 보시오. speedLeft 속도를 늦추고, 그림자의 양을 바꾸어 보시오.
- 좌측 바퀴 속도가 느려지는지를 검사하기 위하여 좌측 광센서로 동일하게 시도해 보시오.

두 개 센서에 더 많이 그림자를 만들어 보시오. 그림자가 두 개 센서에 동일하기 때문에 ndShade는 0에 가까워져야 한다. (speedLeft 와 SpeedRight는 바퀴 속도가 느려지기 전에 100까지 감소해야 한다.)

```
/*
 * Robotics with the BOE Shield - LightSeekingDisplay
 * Displays speedLeft, ndShade, and speedRight in Serial Monitor. Verifies
 * that wheel speeds respond correctly to left/right light/shade conditions.
 */
```

```
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
}

void loop()                     // Main loop auto-repeats
{
  float tLeft = float(rcTime(8));  // Get left light & make float
  float tRight = float(rcTime(6)); // Get right light & make float

  float ndShade;                // Normalized differential shade
  ndShade = tRight / (tLeft+tRight)-0.5; //Calculate it and subtract 0.5

  int speedLeft, speedRight;          // Declare speed variables

  if (ndShade > 0.0)                   // Shade on right?
  {                                    // Slow down left wheel
    speedLeft = int(200.0 - (ndShade * 1000.0));
    speedLeft = constrain(speedLeft, -200, 200);
    speedRight = 200;                  // Full speed right wheel
  }
  else                                 // Shade on Left?
  {                                   // Slow down right wheel
    speedRight = int(200.0 + (ndShade * 1000.0));
    speedRight = constrain(speedRight, -200, 200);
    speedLeft = 200;                 // Full speed left wheel
  }

  Serial.print(speedLeft, DEC);       // Display speedLeft
  Serial.print("    ");               // Spaces
  Serial.print(ndShade, DEC);         // Display ndShade
  Serial.print("     ");              // More spaces
  Serial.println(speedRight, DEC);    // Display speedRight
```

```
  delay(1000);                        // 1 second delay
}

long rcTime(int pin)                  // rcTime measures decay at pin
{
  pinMode(pin, OUTPUT);               // Charge capacitor
  digitalWrite(pin, HIGH);            // ..by setting pin ouput-high
  delay(5);                           // ..for 5 ms
  pinMode(pin, INPUT);                // Set pin to input
  digitalWrite(pin, LOW);             // ..with no pullup
  long time  = micros();              // Mark the time
  while(digitalRead(pin));            // Wait for voltage < threshold
  time = micros() - time;             // Calculate decay time
  return time;                        // Returns decay time
}
```

실습5: 불빛으로 쉴드-봇 조종하기

여기서 LightSeekingDisplay 스케치는 실제로 하려고 하는 것을 화면에 표시할 4가지를 필요로 한다.

1. Serial.print 호출을 제거하시오.
2. 서보 코드를 더하시오.
3. 동작함수(maneuver function.)를 더하시오.
4. 동작함수에 speedLeft 와 speedRightAdd를 통과할 반복함수로 호출

결과는 LightSeekingShieldBot 스케치이다.

- LightSeekingShieldBot를 아두이노에 입력하고 저장하고 업로드하시오.
- 배터리팩을 연결하고, ABOT을 마루에 놓고, 전원 스위치를 2로 놓으시오.
- ABOT이 떠돌도록 해보자. 그리고 좌측과 우측 광센서 위에 그림자를 만들어 보자.

```
/*
 * Robotics with the BOE Shield - LightSeekingShieldBot
 * Roams toward light and away from shade.
 */
```

```
#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
}
```

```
void loop()                // Main loop auto-repeats
{
  float tLeft = float(rcTime(8));    // Get left light & make float
  float tRight = float(rcTime(6));   // Get right light & make float

  float ndShade;                  // Normalized differential shade
  ndShade = tRight/(tLeft+tRight)-0.5;//Calculate it and subtract 0.5

  int speedLeft, speedRight;      // Declare speed variables

  if (ndShade > 0.0)              // Shade on right?
  {                                // Slow down left wheel
    speedLeft = int(200.0 - (ndShade * 1000.0));
    speedLeft = constrain(speedLeft, -200, 200);
    speedRight = 200;             // Full speed right wheel
  }
  else                             // Shade on Left?
  {                               // Slow down right wheel
    speedRight = int(200.0 + (ndShade * 1000.0));
    speedRight = constrain(speedRight, -200, 200);
    speedLeft = 200;               // Full speed left wheel
  }

  maneuver(speedLeft, speedRight, 20);    // Set wheel speeds
}

long rcTime(int pin)              // rcTime measures decay at pin
{
  pinMode(pin, OUTPUT);       // Charge capacitor
  digitalWrite(pin, HIGH);      // ..by setting pin ouput-high
  delay(5);                       // ..for 5 ms
  pinMode(pin, INPUT);          // Set pin to input
  digitalWrite(pin, LOW);       // ..with no pullup
```

```
  long time  = micros();        // Mark the time
  while(digitalRead(pin));     // Wait for voltage < threshold
  time = micros() - time;       // Calculate decay time
  return time;                  // Returns decay time
}

// maneuver function
void maneuver(int speedLeft, int speedRight, int msTime)
{
  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
}
```

광선/그림자 감도 조정

여러분이 빛에 더 민감하기를 원한다면, 1000을 다음 두 줄 명령에서 더 큰 값으로 변경하시오.

```
speedLeft = int(200.0 - (ndShade * 1000.0));
speedRight = int(200.0 + (ndShade * 1000.0));
```

덜 민감하기를 원합니까? 1000을 더 작은 값으로 변경하시오.

반복함수로 맞춰질 수 있는 ABOT을 위한 여러 가지 광감지 조정 아이디어들이 여기에 있다.

- ABOT이 광선 대신 그림자를 따르도록 하기 위하여 if…else 문장 앞에 “ndShade = -ndShade right” 를 놓으시오. 이것이 왜 어떻게 동작하는지 궁금합니까? 이장의 끝 부분에서 프로젝트 2를 점검하시오.
- 매우 밝은 조건과 어두운 조건을 감지함으로써 밝은 장소와 어두운 장소에서 돌아다니는 것을 마치시오. tLeft 와 tRight를 더하고, 결과를 높은 문턱전압(어두움) 혹은 낮은 문턱전압(밝음)과 비교하시오.
- 여러분의 움직이지 않는 상태이거나 밝은 광원 방향으로 회전하는 것에 의해 ABOT 함수가 광 나침반이 되도록 만드시오.
- 보봇이 물체 주위를 탐지하거나 항해할 수 있도록 빛을 향해 움직이는 것에 더듬이를 포함하시오.

제6장 요약

이장은 로봇 조종을 위해 밝은 빛과 어두움을 감지할 수 있는 한 쌍의 광센서 사용법에 초점이 맞춰져 있다. 많은 전자공학 개념과 프로그램 기법들이 소개된다.

전자공학

- 포토트랜지스터가 무엇인가 그리고 베이스, 에미터 그리고 컬렉터를 어떻게 찾는가?
- 자외선 가시광선 적외선의 파장대역은 무엇인가?
- 분위기 광선은 무엇을 의미 하는가?
- 디지털 센서와 아날로그 센서의 차이는 무엇인가.
- 옴의 법칙은 무엇이고, 포토트랜지스터 회로 응답전압을 맞추기 위하여 저항을 어떻게 사용하는가?
- 출력 전압회로에서 간단한 디지털 센서로 포토트랜지스터 사용하기
- 커패시터는 무엇이고, picofarads단위로 표시된 작은 커패시터는 얼마나 작은 양인가?
- 저항-커패시터 전하 이동 회로(QT회로로도 불리어짐)에서 아날로그 센서로 포토트랜지스터 사용하기.
- 부품이 직렬과 병렬로 연결된다는 것은 어떤 의미인가?
- 전압 감쇠가 무엇이고, 저항-커패시터 회로에서 어떻게 이용되는가?

프로그래밍

- 아날로그 센서 출력으로부터 전압 값을 얻기 위한 아두이노 analogRead 함수의 사용법
- 아날로그 센서로부터 측정할 저항-커패시터 회로에서 스케치는 전압감쇠를 어떻게 측정할 수 있는가?
- 변수 값을 위한 상향 한계와 하향 한계를 설정하기 위한 아두이노 constrain 함수를 어떻게 사용하는가?

로보틱스 수단

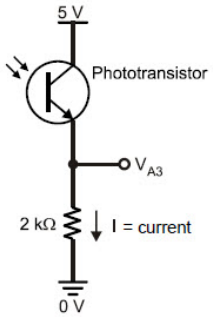
- 빛에 반응하는 자율 센서 주행을 위한 한 쌍의 포토트랜지스터 사용하기

기술 수단

- 회로와 코드 루틴의 서브시스템 점검하기
- 아나로그 입력으로부터 10비트 값을 아두이노에 제공하는 상황에서의 개념
- 영점 조정된 정규화 미분 측정을 위한 방정식 사용하기

제6장 도전

포토트랜지스터 출력 전압 회로



질문

1. 트랜지스터는 무엇을 조절하는가?
2. 어느 포토트랜지스터 끝이 단자를 가지고 있는가?
3. 단자를 구분하기 위하여 포토트랜지스터 플라스틱 케이스의 납작한 지점을 어떻게 사용할 수 있는가?
4. 포토트랜지스터는 어느 색에 더 민감한가: 빨강 또는 초록?
5. 빛이 더 밝아진다면 회로에서 V_{A3} 가 어떻게 반응하는가?
6. V_{A3} 가 증가하도록 또는 감소하도록 하는 위 회로에서 포토트랜지스터는 무엇을 하는가?
7. 빛에 더 민감하도록 위 회로를 어떻게 수정할 수 있는가?
8. 입력으로 설정되어야 할 I/O핀으로 공급되는 전압이 문턱전압 위 또는 아래일 때 무슨일이 일어나는가?
9. 커패시터가 축적한 전하량이 감소한다면, 단자 전압에 무슨일이 일어나

는가?

연습

1. 회로에서 I = 1 mA이라면 V_{A3}에 대하여 풀어보시오.
2. 회로에서 V_{A3}가 4.5V라면 저항을 통과하는 전류를 계산하시오.
3. 105로 적혀있는 커패시터의 값을 계산하시오.
4. 핀 7번으로 지연시간을 측정하는 rcTime문장을 작성하시오. 그리고 tDecay로 이름 붙여진 변수에 결과를 저장하시오.
5. 아두이노가 양쪽 끝에서 1001의 감쇠 값을 측정한다면 ndShade측정치가 무엇이 될지 계산하시오.
6. 직렬모니터에서 50개의 등호 기호를 표시하는 반복문을 작성하시오.

프로젝트

1. 활동1에서, HaltUnderBrightLight sketch 와 함께하는 회로는 과정의 끝에서 ABOT을 불빛 아래에서 멈추게 한다. 여러분이 경쟁이전에 제한된 시간만을 가지고 있다면, 그리고 더 나아가 광원조건을 모르고 있다면 어쩌겠는가? 여러분은 그 장소에서 ABOT을 보정할 필요가 있을 것이다. ABOT이 밝은 빛을 감지할 때 반복적으로 피에죠스피커가 비프음을 내는 스케치 그리고 분위기 불빛을 감지할 때 조용하게 유지하는 스케치가 이런 작업에 유용할 것이다. 실습1에서 이와 같이 동작하는 스케치를 작성하고 점검하시오.
2. ABOT을 이리저리 돌아다니게 만들고 불빛 대신에 어두움을 찾도록하는 응용을 개발하시오. 이와 같은 응용은 Building the Photosensitive Eyes로부터 전하 이동 회로를 이용해야 합니다.
3. ABOT이 유일한 광원이 형광등 불빛인 방안에서 책상 위의 밝은 백열등 방향으로 움직이도록 하는 응용을 개발하시오. ABOT은 백열등 방향으로 움직일 수 있어야 하고 백열등 아래에서 움직여야 합니다. 이 응용은 Building the Photosensitive Eyes로부터 전하 이동 회로를 사용해야 합니다.

제6장 해답

질문 해답

1. 전류의 양은 컬렉터로 흘러 들어가고 베이스를 통과하여 흘러나온다.
5. 포토트랜지스터의 컬렉터와 에미터 끝은 단자로 연결되어 있다.
6. 평평한 지점에 가까운 단자가 에미터이다. 평평한 지점과 멀리 떨어져 있는 단자는 컬렉터이다.
7. 빨강색 파장이 적외선 파장에 더 가깝다. 그래서 빨강에 더 민감해야 한다.
8. VA3는 빛의 양과 함께 증가한다.
9. 포토트랜지스터는 더 많은 전류와 더 작은 전류를 저항으로 공급한다.
10. 2 kΩ 저항을 더 큰값으로 변경하시오.
11. 공급전압이 문턱전압 이상이라면, 입력은 핀이 1의 값을 저장하는 비트를 예약한다. 만약 문턱전압 아래라면, 입력은 0을 저장하는 비트를 예약한다.
12. 전압이 감소한다.

연습 해답

1. V = I × R = 0.001 A × 2000 Ω = 2 V.
2. V = I × R → I = V ÷ R = 4.5 ÷ 2000 = 0.00225 A = 2.25 mA.
3. 105 → 10 with 5 zeros appended and multiplied by 1 pF. 1,000,000 × 1 pF = (1 × 106) × (1 × 10–12) F = 1 × 10–6 F = 1 µF.
4. It would be long tDecay = rcTime(7);
5. ndShade = tRight / (tLeft+tRight) – 0.5 = 1001 ÷ (1001 + 1001) – 0.5 = 0.5 – 0.5 = 0.
6. Solution:

```
for(int i = 1; i<=50; i++)  // Repeat 50 times
{
  Serial.print('=');          // one = char each time through
}
```

Project Solutions

1. 이것은 HaltUnderBrightLight의 수정 버전입니다.

```
/*
 * Robotics with the BOE Shield - Chapter 6, Project 1
 * Chirp when light is above threshold.  Will require updating value of
 * threshold & retesting under bright light to get to the right value.
 */
```

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

void loop()                               // Main loop auto-repeats
{
  if(volts(A3) > 3.5)                     // If A3 voltage greater than 3.5
  {
    tone(4, 4000, 50);                    // Start chirping
    delay(100);
  }
}

float volts(int adPin)                    // Measures volts at adPin
{                                         // Returns floating point voltage
 return float(analogRead(adPin)) * 5.0 / 1024.0;
}
```

2. 이것의 해답은 LightSeekingShieldBot를 복사하고, loop함수에 하나의 명령을 더 추가하시오:ndShade = -ndShade. if…else문장바로 전에 그것을 추가하시오. 그다음 회피할 그림자를 가르키는 것 대신에, 그것은 회피할 밝은 빛을 가르킵니다. 또 다른 접근은 tLeft / (tLeft + tRight)와 동등한 ndLight 계산을 사용하는 것입니다. 여러분은 스케치

에서 ndShade를 ndLight로 바꿔야 할 것입니다.

3. 문턱값을 결정하기 위하여 LightSensorValues를 사용하시오. tLeft + tRight를 취하는 것이 최선입니다. 여러분은 이것을 테스트하기 위하여 LightSensorValues를 사용할 수 있다. 밝은 빛 수준에 이르는 방법의 1/4 값으로 시작하시오. 그래서 밝은 빛에 대하여 tLeft + tRight = 4000과 400이라면, 400 + $\frac{1}{4}$ × (4000 − 400) = 400 + 900 = 1300 값을 사용하시오.

   다음에 HaltUnderBrightLight부터 LightSeekingShieldBot의 메인 loop까지 유사한 if 문장을 그것에 추가하시오. HaltUnderBrightLight는 전압 출력회로를 사용하기 때문에 보다 더 크게 뜻의 (>)연산자를 사용합니다. 보다 더 작은 값은 더 밝은 빛을 의미하기 때문에 QT회로를 위해 보다 더 작은 뜻의 (<)연산자를 필요로합니다. 우리는 또한 1300.0처럼 부동소수점 값으로 문턱 값을 표현합니다. 여기에 예제가 있습니다.

```
// Add this if condition to stop under the bright lamp.
if((tRight + tLeft) < 1300.0)    // tLeft+tRight < 1300?
{
  servoLeft.detach();              // Stop servo signals
  servoRight.detach();
}
```

여기 LightSeekingShieldBot의 수정된 버전에 트릭을 추가할 것입니다. 여러분은 여전히 광선 상태에 따라 그것을 계산해야 할 것입니다.

```
/*
 * Robotics with the BOE Shield - Chapter 6, Project 3
 * Roams toward light and away from shade.
 */
```

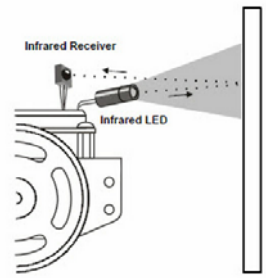
```
#include <Servo.h>                    // Include servo library
Servo servoLeft;                  // Declare left and right servos
Servo servoRight;
```

```
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
}

void loop()                       // Main loop auto-repeats
{
  float tLeft = float(rcTime(8));    // Get left light & make float
  float tRight = float(rcTime(6)); // Get right light & make float

  // Add this if condition to stop under the bright lamp.
  if((tRight + tLeft) < 1300.0)      // If A3 voltage greater than 2
  {
    servoLeft.detach();              // Stop servo signals
    servoRight.detach();
  }

  float ndShade;                    // Normalized differential shade
  ndShade = tRight / (tLeft+tRight) - 0.5;//Calculate it and subtract 0.5

  int speedLeft, speedRight;       // Declare speed variables

  if (ndShade > 0.0)                // Shade on right?
  {                                 // Slow down left wheel
    speedLeft = int(200.0 - (ndShade * 1000.0));
    speedLeft = constrain(speedLeft, -200, 200);
    speedRight = 200;               // Full speed right wheel
  }
  else                             // Shade on Left?
```

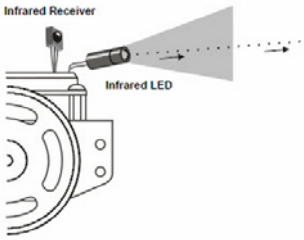
```
  {                              // Slow down right wheel
    speedRight = int(200.0 + (ndShade * 1000.0));
    speedRight = constrain(speedRight, -200, 200);
    speedLeft = 200;             // Full speed left wheel
  }
  maneuver(speedLeft, speedRight, 20); // Set wheel speeds
}
long rcTime(int pin)             // rcTime measures decay at pin
{
  pinMode(pin, OUTPUT);       // Charge capacitor
  digitalWrite(pin, HIGH);    // ..by setting pin ouput-high
  delay(5);                   // ..for 5 ms
  pinMode(pin, INPUT);        // Set pin to input
  digitalWrite(pin, LOW);     // ..with no pullup
  long time  = micros();      // Mark the time
  while(digitalRead(pin));    // Wait for voltage < threshold
  time = micros() - time;     // Calculate decay time
  return time;                // Returns decay time
}
// maneuver function
void maneuver(int speedLeft, int speedRight, int msTime)
{
  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
}
```

제7장 적외선 헤드라이트 자율 주행

ABOT은 이미 더듬이 센서를 사용 할 수 있다. 그러나 단지 물체에 닿았을 경우에만 장애물을 감지한다. 만약 ABOT은 물체를 "보는것" 만으로 물체에 대해서 무엇을 하는지 알아차릴 수 있다면 더 좋지 않을까? 자 이제 적외선(infrared hedadlights)를 가지고 아래에서 보는 것과 같이 사람의 눈과 비슷한 것을 만들려고 한다. 각각의 지시등은 적외선 LED가 손전등 같이 빛을 앞으로 곧장 보내도록 튜브 안에 들어있다. 각각의 눈은 적외선 수신부(infrared receiver)이고, 물체에서 반사된 적외선 LED 빛을 감지하였는지를 나타내는 아두이노 HIGH/LOW 신호를 보낸다.



적외선 반사됨,
물체 감지됨



적외선 반사되지 않음,
물체 감지되지 않음

TV 리모콘은 약 38kHz로 특정 길이의 시간동안 특정 주기 사이에서 빛을 깜박인다.

이것은 리모콘에 있는 각 키에 서로 다른 깜박임/꺼짐 패턴을 만든다.

적외선 빛 신호(Infrared Light Signals)

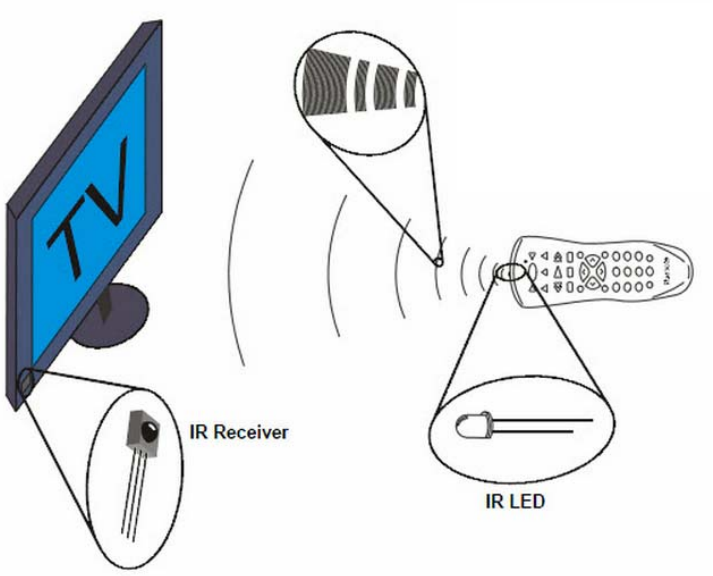
적외선(Infrared)은 일반적으로 IR 이라고 한다. 이 빛은 사람이 눈으로 볼 수 없는 빛의 범위에 있다. 이장에서는 IR LEDs가 적외선 빛을 내는 것을 소개하고 단지 우리가 가시광선을 내기위해 사용한 red LED와 같은 것이다.

이 장에서는 적외선 수신부(infrared receiver)가 적외선 빛을 감지한다. 지난 장에서 포토레지스터와 비슷하다. 그러나 조금 다른 것은 이들 적외선 수신부는 주변 환경의 빛을 감지하는 것이 아니고 적외선 빛을 매우 빨리 있는지 없는지 감지 할 수 있게 설계되어 있다.

작은 전조등으로써 사용되는 ABOT에 있는 적외선 LED는 실제로 TV 리모콘에

서 볼 수 있는 것과 같다. TV 리모콘은 TV에 메시지를 보내기 위해 IR LED 빛을 쏜다. TV에 있는 마이크로콘트롤러는 이들 메시지를 우리가 사용하려는 ABOT에 있는 것과 같은 적외선 수신부로 잡아낸다.

TV 리모콘은 약 38kHz(초당 약 38,000회) 대역에서 IR LED를 매우 빠르게 깜박임으로 메시지를 보낸다.



IR 수신부는 단지 이 대역에 맞게 깜박이는 적외선만 받는다. 이것은 리모콘이 메시지로 해석 할 수 있는 태양과 백열등 빛과 같은 적외선을 막는다. 그래서 IR 수신부는 신호를 감지 할 수 있고, 아두이노는 28kHz 깜박일 것이다.

주의! 어떤 형광등 빛은 IR 수신부에 의해 감지 될 수 있다. 이들 빛은 ABOT의 적외선 빛을 사용하는데 문제가 될 수 있다. 이들 중 하나는 이 장에서 개발할 “탐지기”(Sniffer)는 ABOT 코스에서 가까이 있는 형광등 빛을 테스트 하는데 사용될 수 있다.

디지털 카메라, 휴대전화, 웹 카메라 안에 있는 빛 센서는 모든 적외선 빛을 감지 할 수 있다. 디지털 카메라를 통해서 보면 적외선 LED가 켜져 있는지 꺼진지

우리가 볼 수 있다. 이들 사진들은 디지털 카메라와 TV 리모콘의 예이다. 우리가 리모콘의 버튼을 누르고 있고 디지털 카메라의 렌즈가 IR LED가 가리키고 있으면, 흰 빛이 깜박이면서 적외선 LED를 표시하는 것이 보인다.

With a button pressed and held, the IR LED doesn't look any different.

Through a digital camera display, the IR LED appears as a flashing, bright white light.

디지털 카메라 안에 들어있는 이 픽셀 센서는 빨강(Red), 초록(Greed), 파랑(blue) 빛의 강도를 감지한다. 픽셀센서가 빨강, 초록, 파랑을 감지하는 것에 관계없이 이것은 적외선을 감지한다. 모든 세 픽셀 컬러 센서는 또한 적외선을 감지한다. 디지털 카메라는 모든 색상을 함께 혼합하여 흰색으로 표시한다.

Infra는 아래라는 의미이고, 그래서 적외선은 빨간색(red) 아래라는 의미이다. 이 이름은 빨간색 빛의 파장의 주파수 보다 적다는 것을 의미한다. 우리의 IR LED 송신부는 파장의 길이가 980 나노미터(nanometer, nm으로 표시)로 전송하고, 우리의 IR 수신부도 파장이 같은 것을 감지한다. 이 파장은 적외선 범위 내에 있다. 원적외선 범위는 2000 ~ 10,000 nm이며, 범위의 특정 파장은 나이트 비전 고글 및 적외선 온도 감지를 위해 사용됩니다.

http://learn.parallax.com/node/301

실습1: 목표물 감지기 만들기와 테스트

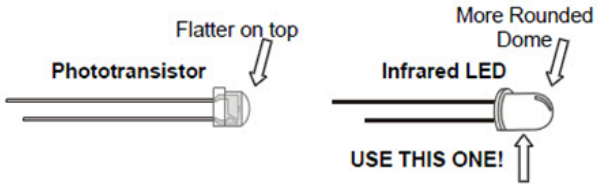
이 장에서는 ABOT을 위한 적외선 물체 감지기를 만들고 테스트한다.

부품 목록

(2개) IR 수신기(IR receivers)
(2개) IR LEDs 투명 케이스
(2개) IR LED 쉴드 막이(shield assemblies)
(2개) 저항, 220 Ω (빨강-빨강-갈색)
(2개) 저항, 2 kΩ (빨강-검정-빨강)

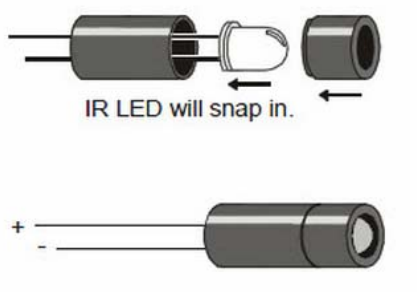
부품 목록에서 부품들을 모아서 적외선 수신부, LED와 쉴드 막이 부품을 사용하기 위해 아래의 그림을 이용하시오.

여러분이 선택한 것이 포토트랜지스터가 아니고 적외선 LED가 확실한지 아래 그림을 보고 확인하시오. 적외선 LED는 둥근 플라스틱의 튀어나온 부분이 좀 더 길고 둥글다. 그리고 오른쪽 그림에 있는 것이다.



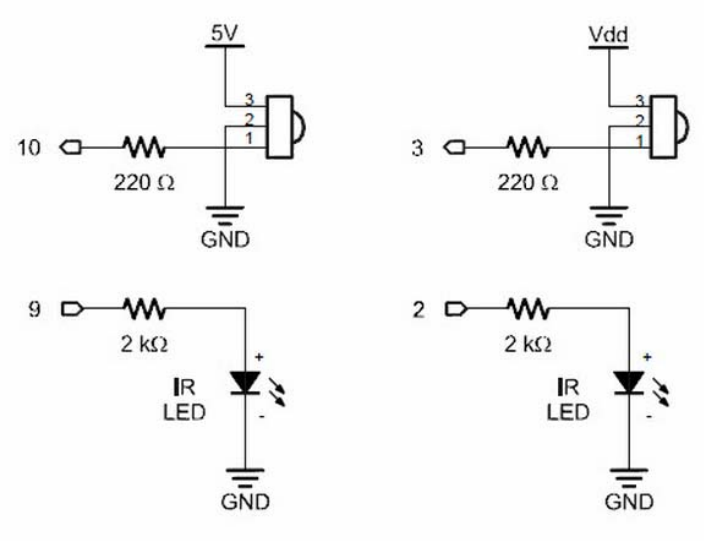
IR 전조등 조립하기

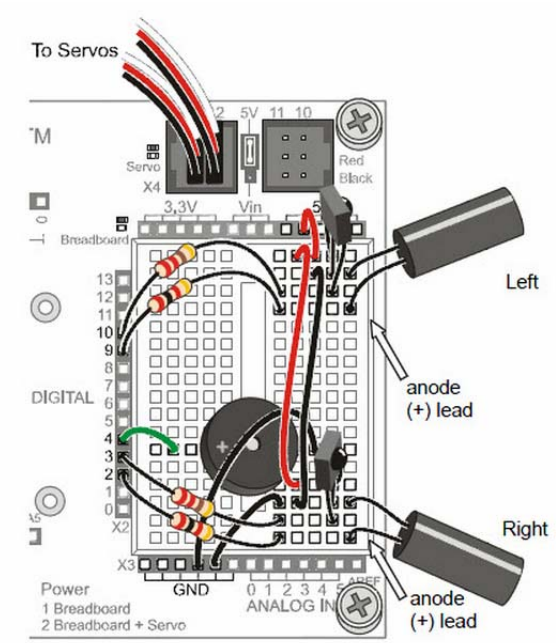
✓ IR LED를 LED 지지대(두 개 중 더 큰 것)에 아래 보이는 것처럼 끼운다.
✓ IR LED 는 LED 지지대에 딸깍 끼운다.
✓ IR LED의 투명 플라스틱 케이스위세 짧은 튜브를 끼운다. 튜브의 끝에 링은 LED 지지대에 꼭 끼게 맞춘다.
✓ 사용하는 동안 조립된 부품들이 분리되지 않게 투명 테이프의 작은 조각을 이용하라.



IR 물체 감지 회로 IR 목표물 감지 회로
다음 그림은 IR 물체 감지 회로도를 보여준다. 하나의 IR 물체 감지자(IR LED와 수시부는 한쌍이다)는 ABOT의 브레드보드 가장 앞의 한쪽 코너에 끼운다.

✓ 전원과 프로그래밍 케이블 연결을 빼시오.
✓ 부품을 배치하기 위한 참고로 배선도를 사용하여 아래 그림과 같이 회로를 완성하시오.
✓ IR LED의 양극을 각각 2 kΩ 저항에 연결하시오. 음극선은 브레드보드에서 IR 수신부의 중앙핀과 같은 선으로 연결하시오. 그리고 점퍼선은 GND에 연결하시오.


5V
10
220 Ω
GND
Vdd
3
220 Ω
GND
9
2 kΩ
IR
LED
GND
2
2 kΩ
IR
LED
GND


To Servos
Servo
X4
3,3V
Vin
Red
Black
Breadboard
Left
anode
(+) lead
DIGITAL
Right
anode
(+) lead
GND
ANALOG IN
Power
1 Breadboard
2 Breadboard + Servo

주의 !
IR LED의 양극과 음극을 확인하시오.
양극선은 IR LED에서 관습적으로 좀 더 길다. 음극선은 좀 더 짧고 플라스틱 케이스에서 편편한 쪽이다. 이들은 빨간색 LED에도 사용하는데 동일하게 습관적인 것이다.

물체 감지 테스트 코드

여러분의 ABOT의 적외선 수신부는 적외선 빛(980 나노미터 범위)을 38KHz 가까운 속도로 감지하도록 설계되어 있다. IR LED를 이 속도로 on/off 하게 만들기 위해, 우리는 각 스케치를 시작할 때 스피커 삑 소리를 내는 것과 비슷한 음정을 사용할 수 있다.

적외선 감지는 다음 세 단계를 따른다.
1. IR LED on/off 를 38 kHz 속도로 비춘다.
2. IR 물체에 반사된 38 kHz IR 빛을 보내기위한 반응으로 낮은(low) 신호를 보내기 위해 IR 수신부는 시간을 1/1000초 또는 조금 더 지연시킨다.
3. 각각의 높은 신호(IR 감지가 아니) 또는 낮은 신호(IR 감지) 둘 다를 위해 IR 수신부의 상태를 확인한다.

여기 왼쪽 IR LED(핀 9번)와 IR 수신부(핀 10) 세단계의 예제가 있다.

```
tone(9, 38000, 8);                // IRLED가 38 kHz 로 최소 1 ms 지속
delay(1);                         // Wait 1 ms
int ir = digitalRead(10);         // IR receiver -> ir variable
```

참고 : **음정(tone)은 약 1.1ms 동안 지속된다.**
8초간 38kHz 신호를 발생시키기 위해 tone(9, 38000, 8) 을 기대 하더라도 신호는 실제로 아두이노의 소프트웨어 버전 v1.0 에서는 1.1 ms 동안만 지속된다. tone 이라는 이름을 부여하고 함수는 가청 범위로 테스트되어 설계되어 있을 수 있다. 만약 스피커가 실행된다면 38kHz는 들을 수 없을 것이다. 이것은 ultrasonic(초음파)의 범위이고 이것은 사람이 들을 수 없는 너무 높은 소리라는 것을 의미한다. 사람이 들을 수 있는 범위는 20 Hz 에서 20kHz 이다.

tone 함수는 초당 38000 번 high/low를 반복하는 파장을 발생시킨다. 아두이노 소프트웨어 v1.0으로서, 실험을 통하여 8ms 동안 도착했고, 이것은 38kHz 사각파를 1.1ms 동안 지속적으로 일으킨다.
tone 함수는 배경으로 처리되는 것이라는 것을 명심하시오.
IR 수신부는 38 KHz 신호에 반응하기 위한 천분의 몇십초 단위가 필요하다. delay(1)은 ir = digitalRead(10)가 IR 수신부의 출력이 준비될 때 까지 복사되어 호출되는 것을 막는다.

다음 스케치는 세 개의 매개변수로 irDetect 라고 하는 함수를 정의 한다. : 하나는 IR LED 핀 이고, 다른 하나는 IR 수신부를 정의하고 나머지는 IR LED를 깜박이는 주기를 설정하기 위한 것이다.

```
int irDetect(int irLedPin, int irReceiverPin, long frequency)
```

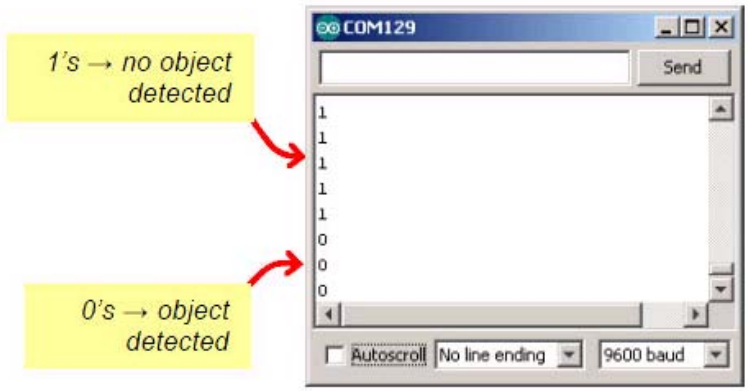
loop 함수에서 irDetect를 호출할 것이다.

```
int irLeft = irDetect(9, 10, 38000);     // 실제 적용 예
```

이 호출은 irLedPin 매개변수에 9를 할당하고, irReceiverPin에 10을 할당하고 frequency 에 38000을 할당하는 것이다. 이 함수는 적외선을 감지하는 것과 물체를 감지하지 않았을 때 1을 리턴하고 물체를 감지하였을 때 0을 리턴하는 세 단계를 수행한다. 이 리턴 값은 irLeft에 저장된다.

스케치 예제 : TestLeft

이 스케치는 ABOT의 왼쪽 IR 감지기만 테스트 하는 것이다. 이것은 여러분이 두개의 회로 중 하나만 집중 할 수 있기 때문에 간단히 문제해결을 도와준다. 이것은 아직도 보조 실험의 예이다. 이 보조 실험을 확인 한 후 우리는 시스템을 통합한다. 첫 번째 테스트와 전선 또는 코드가 정확한지를 확인해야 한다.



✓ 건전지를 아두이노에서 빼내시오.
✓ 3지점 스위치를 1위치에 놓는다.
✓ 코드를 입력하고 저장 한 다음 아두이노에 TestLeftIr을 업로드 하시오.

```
/*
 * ABot - TestLeftIR
 * IR 감지기가 물체를 감지하지 않으면 1을 표시하고
 * 물체를 감지하면 0을 표시한다.
 */
```

```
void setup()                    // Built-in initialization block
{
  tone(4, 3000, 1000);          // Play tone for 1 second
  delay(1000);                  // Delay to finish tone
  pinMode(10, INPUT); pinMode(9, OUTPUT);// Left IR LED & Receiver
  Serial.begin(9600);           // Set data rate to 9600 bps
}
```

```
void loop()                          // Main loop auto-repeats
{
  int irLeft = irDetect(9, 10, 38000);  // Check for object
  Serial.println(irLeft);         // Display 1/0 no detect/detect
  delay(100);                          // 0.1 second delay}

// IR Object Detection Function
int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);    // IRLED 38 kHz for at least 1 ms
  delay(1);                           // Wait 1 ms
  int ir = digitalRead(irReceiverPin);  // IR receiver -> ir variable
  delay(1);                            // Down time before recheck
  return ir;                          // Return 1 no detect, 0 detect
}
```

- ✓ ABOT 프로그래밍 케이블을 연결하고, 스케치를 업로드 시킨 후 시리얼 모니터를 여시오.
- ✓ 손이나 종이와 같은 물체를 IR 물체 감지기에서 1인치(2-3cm) 정도 앞에 그림 7-1과 같이 두시오.
- ✓ 시리얼 모니터에 예상한대로 장애물을 감지하지 않았을 때 1을, 장애물을 감지하였을 때 0이 표시된다면 다음 장으로 넘어가세요.
- ✓ 시리얼 모니터에 예상 값이 표시된다면 오른쪽 IR 물체 감지기(Object Detector)로 넘어가시오. 만약 예상 값이 표시되지 않으면 도움말을 참고하세요.

실습 - Test the Right IR Object Detector

오른쪽 IR 물체 감지기를 테스트하기 위해 스케치를 수정하려면 irLeft를 irRight로 고치고, 회로에 테스트하기 위해 핀 번호를 매개 변수에 정확히 대입한다. 다음은 변경하기 위한 목록 순서이다.

- ✓ TestLeftIr를 TestRightIr로 변경하여 저장한다.
- ✓ pinMode(10, INPUT); pinMode(9, OUTPUT)를 pinMode(3, INPUT);

pinMode(2, OUTPUT)으로 변경한다.

- ✓ int irLeft = irDetect(9, 10, 38000)를 int irRight = irDetect(2, 3, 38000)로 변경한다.
- ✓ Serial.println(irLeft)를 Serial.println(irRight)로 변경한다.
- ✓ ABOT의 오른쪽 IR 물체 감지기를 위해 테스트 단계를 반복한다.
- ✓ 회로나 코드의 에러를 해결하기 위하여 도움말을 찾으시오.

실습2: 현장 테스트

이번 실습에서는 시리얼 모니터의 도움 없이 물체를 감지하였는지를 알려주는 지시자 LED를 만들고 테스트 한다. 근처에 PC나 노트북이 없어도 조작 할 수 있어 편리하고, IR 감지기 회로에 문제가 있을 때 필요하다.

여러분은 또한 형광등 빛에서 생기는 적외선 간섭 "탐지" 하기위해 스케치를 할 것이다. 어떤 형광등 빛은 적외선 LED에서 보내는 신호와 비슷한 신호를 보낸다. 형광등은 "안정기(ballast)" 라고 하는 전압 조정 장치가 내부에 있다. 어떤 안정기는 38.5 kHz 적외선 신호로 발산하기 때문에 IR 감지기와 같은 파장으로 작동한다. 방향 조정을 위한 IR 물체 감지기를 사용할 때 이 간섭 대문에 ABOT이 이상한 행동을 할 수도 있다.

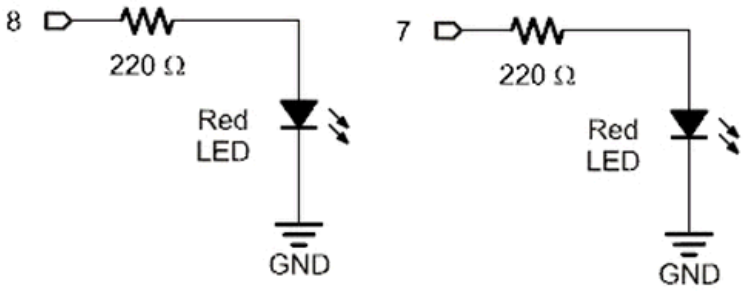
LED 지시 회로 추가하기

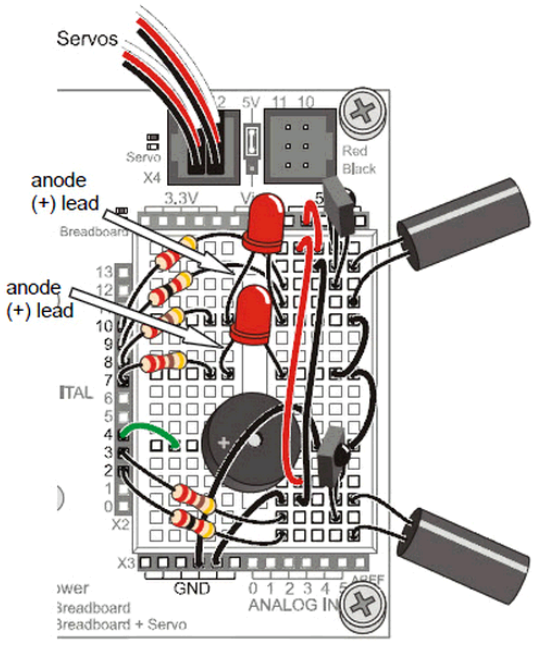
LED 지시회로는 더듬이에서 사용한 것과 비슷하다. 선을 연결 할 때 양극과 음극을 주의해야 한다.

필요한 부품들

(2개) 빨강색 LEDs
(2개) 저항, 220 Ω (red-red-brown)

✓ 프로그래밍 케이블과 전원 공급선을 뺀다.
✓ ABOT에 부품의 위치를 맞추기 위해 배선도를 사용하여 배선도의 빨간색 LED를 회로도에서 보는 것처럼 추가하시오.





시스템 테스트하기

이 시스템에는 꽤 많은 구성 요소들이 있어서 배선 에러가 날 가능성이 높아진다. 그래서 적외선 감지기가 작동하는 것을 확인하는 테스트를 하는 것이 중요하다. ABOT에서 프로그래밍 케이블을 분리하기 전에 이 스케치를 이용하여 회로를 확인한다.

예제 스케치 – TestBothIrAndIndicators

- ✓ 프로그래밍 케이블과 전원 케이블을 연결하고 전원스위치를 1에 놓는다.
- ✓ TestBothIrAndIndicators 코드를 입력하고 저장한 다음 아두이노에 업로드 하시오.
- ✓ 업로드 한 다음에 스피커에서 들을 수 있는 소리가 맑게 나는지 확인하시오.
- ✓ 물체가 앞에 있을 때 각 IR 감지기로 부터 아두이노가 '0'을 받고 있는지 확인하기 위해 시리얼 모니터를 사용하시오.
- ✓ 왼쪽 IR 감지기 앞에 물체가 있을 때 왼쪽(8번 핀)의 빨간색 LED가 빛을 내는지 확인하시오.
- ✓ 오른쪽 IR 감지기와 7번 핀 LED 에 반복해서 실험 하시오.

```
/*
 * Robotics with the BOE Shield - TestBothIrAndInciators
 * Test both IR detection circuits with the Serial Monitor.  Displays 1 if
 * the left IR detector does not detect an object, or 0 if it does.
 * Also displays IR detector states with indicator LEDs.
 */
```

```
void setup()                    // Built-in initialization block
{
  tone(4, 3000, 1000);       // Play tone for 1 second
  delay(1000);                // Delay to finish tone
  pinMode(10, INPUT);  pinMode(9, OUTPUT);//Left IR LED & Receiver
  pinMode(3, INPUT);  pinMode(2, OUTPUT);//Right IR LED & Receiver
  pinMode(8, OUTPUT); pinMode(7, OUTPUT);// Indicator LEDs

  Serial.begin(9600);          // Set data rate to 9600 bps
}
 void loop()                    // Main loop auto-repeats
{
  int irLeft = irDetect(9, 10, 38000);// Check for object on left
  int irRight = irDetect(2, 3, 38000);// Check for object on right
  digitalWrite(8, !irLeft);         // LED states opposite of IR
  digitalWrite(7, !irRight);
  Serial.print(irLeft);              // Display 1/0 no detect/detect
  Serial.print("   ");              // Display 1/0 no detect/detect
  Serial.println(irRight);          // Display 1/0 no detect/detect
  delay(100);                       // 0.1 second delay
}
// IR Object Detection Function
int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);  // IRLED 38 kHz for at least 1 ms
  delay(1);                             // Wait 1 ms
  int ir = digitalRead(irReceiverPin);// IR receiver -> ir variable
```

```
  delay(1);                    // Down time before recheck
  return ir;                   // Return 1 no detect, 0 detect
}
```

TestBothIrAndIndicators 코드 설명

irDetect 함수는 두 개, 즉 irDetect 와 irRight를 호출한 후 물체가 감지되지 않으면 1을 저장하거나 물체가 감지되면 0을 저장한다. 스케치는 지시자의 불빛이 irLeft 와 irRight의 값에 따라 켜짐/꺼짐을 반복하기 위해 if…else를 사용할 수 있다. 하지만 다른 방법도 있다.

이 스케치는 iRight의 값을 직접적으로 사용한다. irLeft 가 0을 저장할 때 단지 하나만 받고 빨간색 지시 LED를 켠다. 그리고 1이 저장되면 LED를 끈다. 1은 LED를 켜는 지시자이고 0을 끄는 지시자이기 때문에 digitalWrite(8, irLeft) 를 사용하면 우리가 원하는 반대 값을 받을 수 있다. 그래서 이 스케치는 irLeft 저장값을 반전하기 위해 not 연산자(!)를 사용한다. 자 이제 digitalWrite(8, !irLeft) 는 IrLeft 값을 반전한다. 그래서 0을 저장할 때 LED 는 켜지고, 1을 저장 할 때 LED가 꺼진다. 같은 방법으로 오른쪽 IR 감지기와 LED 지시자에 적용할 수 있다.

참고

Not(!) 연산자 는 변수에서 이진 값을 반전시킨다.
좀 더 나가면 **digitalWrite(8, !irLeft)**를 볼 것이다. 아마도 **HIGH (상수 1)는** 불이 켜지고 **LOW (상수 0)**은 불이 꺼지는 **digitalWrite** 함수 값의 매개변수를 적용할 것이다. 변수값을 사용할 때 **digitalWrite** 함수는 변수의 가장 오른쪽 이진수 자리를 사용한다. 즉 1은 불이 켜지고 0을 불이 꺼진다. 그래서 **irLeft** 가 0 (물체 감지)일 때는 **!irLeft** 는 이진수 00000000 을 11111111. 으로 변경하게 된다. 가장 오른쪽 자리수가 1이기 때문에 불이 켜진다. **irLeft는** 1일 때 실제 이진 수는 00000001 이다. !irLeft 구문의 결과는 11111110이 된다. 가장 오른쪽 자리수가 0이므로 불이 켜진다.

실습하기 – 거리 테스트와 원격 테스트(Remote Testing and Range Testing)

LED가 작동함으로 우리는 ABOT에서 프로그래밍 케이블을 빼 낼 수 있고, 다양한 물체를 감지하는 테스트를 할 수 있다. 어떤 물체는 적외선을 더 잘 반사하기 때문에 IR 물체 감지기가 더 멀리 있는 물체를 감지 할 수 있을 것이고 어떤 것은 매우 가까이 있어야 한다.

- ✓ ABOT에서 프로그래밍 케이블을 분리 하고, 여러분이 방향을 지시하는 범위에서 물체를 감지하는 것을 다시 해 보시오.
- ✓ 다양한 색상이 있는 물체를 감지하는지 거리를 실험 해 보시오. 가장 가까운 거리에서 감지되는 색상과 표면은 무엇인가요? 가장 멀리서 감지되는 색상과 표면은 무엇인가요?

IR 간섭 탐지하기 (Sniffing for IR Interference)

아무것도 없는 곳에서도 ABOT이 어떤 물체를 탐지했다고 말하는 것을 볼 수도 있다. 이것은 아마도 주변에서 38.5 kHz 와 비슷한 주파수로 IR 빛과 비슷한 것을 발생하고 있다는 의미이다. 이것은 직접 태양빛이 창문을 통과하여 감지가 안되는 원인이 될 수도 있다. 만약 이런 빛에 있는 근처에서 로봇 대회 또는 시연을 해야 한다면 적외선 시스템은 매우 나쁜 수행 결과가 될 것이다. 그래서 사람들 앞에서 시연하기 전에 정확한 시연을 하기 전에 미리 “sniffer" 스케치로 IR 간섭영역을 확인해야 한다.

이 스케치의 원리는 간단하다. IR LED를 통해서 어떤 IR 도 전송되고 있지 않을 때 IR 이 어떤 것이라도 감지한다는 것을 관찰하기 위해, 만약 감지된다면 피에조스피커를 사용하여 경고음을 발생시키는 것이다.

주의

여러분은 IR 간섭을 발생시키는 기계 중 하나인 리모콘을 사용할 수 있다.
TV, VCR, CD/DVD 재생기 리모콘은 IR LED와 같은 종류를 사용하는 모든 기기를 조작할 수 있다. 이들은 ABOT이 방금 감지한 IR 감지기와 같은 종류를 사용하고 있다. 그래서 ABOT이 TV, VCR, CD/DVD 재생기등에 메시지를 전달할 수 있다. 모든 IR 간섭은 ABOT을 지시하고 리모콘 버튼의 누름/해제를 반복한다.

예제 – IrInterferenceSniffer

이 스케치에서는 ABOT가 tone을 실행 해야한다. 지시자 LED를 켜고, 시리얼 모니터 적외선을 감지했다는 경고를 표시한다. 다시 아무 IR도 전송되지 않기 때문에 38kHz 적외선은 외부 소스로 부터 와야 한다는 의미이다.

- ✓ 코드를 입력하고 아두이노에 IrInterferenceSniffer 를 업로드 하시오.
- ✓ IR 간섭이 감지될 때 ABOT이 경고 소리가 나는지 테스트 하시오. 만약 교실이라면 각각의 ABOT을 TestBothIrAndIndicators로 작동 시킬 수 있다. IR LED는 IrInterferenceSniffer bot의 IR 수신기에 IR LED만 지시한다. ABOT이 없다면 TV, VCR, CD/DVD 재생기를 위한 리모콘을 사용할 수 있다. ABOT에서 버튼중 하나를 반복해서 누르고 떼고 하여 간단한 지시를 한다. ABOT이 없다면 TV, VCR, CD/DVD 재생기를 위한 리모콘을 사용할 수 있다. ABOT이 경고음으로 반응하면 여러분은 IR 간섭 탐지기가 작동하는 것을 알 수 있다.

```
/*
 * ABOT 로봇을 위한- IrInterferencesSniffer
 * 적외선 간섭의 외부 원인을 테스트 하기 위함 만약 IR 간섭이 감지된다면
 * 시리얼 모니터가 경고를 표시하고, 피에조 스피커가 경고음을 발생시킨다.
 * 지시등이 깜박인다.
 */
```

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

```
  pinMode(10, INPUT);               // Left IR Receiver
  pinMode(3, INPUT);                // Right IR Receiver
  pinMode(8, OUTPUT);               // Left indicator LED
  pinMode(7, OUTPUT);               // Right indicator LED

  Serial.begin(9600);               // Set data rate to 9600 bps
}

void loop()                         // Main loop auto-repeats
{
  int irLeft = digitalRead(10);     // Check for IR on left
  int irRight = digitalRead(3);     // Check for IR on right

  if((irLeft == 0) || (irRight == 0)) // If left OR right detects
  {
    Serial.println("IR interference!!!"); // Display warning
    for(int i = 0; i < 5; i++)          // Repeat 5 times
    {
      digitalWrite(7, HIGH);        // Turn indicator LEDs on
      digitalWrite(8, HIGH);
      tone(4, 4000, 10);            // Sound alarm tone
      delay(20);                    // 10 ms tone, 10 between tones
      digitalWrite(7, LOW);         // Turn indicator LEDs off
      digitalWrite(8, LOW);
    }
  }
}
```

실습하기 - 형광등의 간섭 테스트하기

✓ ABOT에서 프로그래밍 케이블을 분리하고, 형광등이 있는 불빛 아래에서 테스트 하시오.

✓ 특히 자주 경보가 발생된다면 IR이 물체를 탐지하기 전에 형광등을 끄시오.

✓ 창문을 통해 빛이 들어온다면 블라인드를 닫고 다시 테스트 하거나 창문으로

부터 멀리 떨어진 장소로 로봇을 이동하시오.

✓ ABOT은 적외선 간섭으로 부터 벗어나기 위해 IrInterferenceSniffer를 항상 사용하시오.

실습3: 감지 범위 맞추기

자동차의 전조등(또는 더 밝은 플래시)이 밝을수록 어두울 때 더 멀리 볼 수 있다는 사실을 알고 있다. ABOT의 적외선 LED 전조등이 더 밝으면 또한 감지 범위가 증가 한다. 더 작은 저항은 LED에게 더 많은 전류를 흐르게 한다. LED에 더 많은 전류가 흐르므로 더 밝아지는 원인이 된다. 이 실습에서는 LED와 적외선 LED에 서로 다른 저항 값의 영향을 실험할 것이다.

필요한 부품들:
이 실험에서 추가로 필요한 부품들.

(2개) 470 Ω 저항 (노랑-보라-갈색)
(2개) 220 Ω 저항 (빨강-빨강-갈색)
(2개) 1 kΩ 저항 (갈색-검정-빨강)
(2개) 4.7 kΩ 저항 (노랑-보라-빨강)

저항 시리즈와 LED 밝기

먼저, LED를 하나 사용하여 저항이 LED를 어떻게 밝기를 다르게 만드는지 보자. 모두 LED가 필요하고 스케치는 LED에 high 신호를 전달한다.

예제 스케치 - P1LedHigh
✓ 코드를 입력하고 저장하여 LeftLedOn를 업로드 시킨다.
✓ 스케치를 실행시키고 회로에서 P8에 연결된 LED가 빛을 내는지 확인한다.

```
// ABOT 로봇에 - LeftLedOn
// 밝기 실험을 위한 왼쪽 LED를 켜는 실험

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

  pinMode(8, OUTPUT);          // Left indicator LED
  digitalWrite(8, HIGH);
}
```

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

실습4: 물체 감지와 피하기

IR 감지기에 대해서 흥미로운 사실 한 가지는 더듬이(whiskers)와 비슷한 결과를 낸다는 것이다. 물체를 감지하지 않았을 때 결과값은 high(1)이고, 물체를 감지하였을 때 결과값은 low(0)이다. 이 실습에서 RoamingWithWhiskers 스케치는 수정하여 IR 감지기를 작동시킨다. 수정하는 것은 간단하다. 단계는

1. RoamingWithWhiskers를 RoamingWithIr로 저장하시오.
2. irDetect 함수를 추가한다.

```
int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);
  delay(1);
  int ir = digitalRead(irReceiverPin);
  delay(1);
  return ir;
}
```

3. digitalRead 호출을 수정한다.

```
byte wLeft = digitalRead(5);
byte wRight = digitalRead(7);

/* irDetect 를 호출

int irLeft = irDetect(9, 10, 38000);
int irRight = irDetect(2, 3, 38000);
```

4. wLeft 인스턴스를 irLeft 로 바꾸고 irLeft를 irRight로 바꾸시오.
5. 주석 /*…*/ 와 // 을 추가하시오,

```
Replace all instances of wLeft with irLeft and wRight with irRight.
Update the /*…*/ and // comments.
```

스케치 실습 - RoamingWithIr

✓ RoamingWithWhiskers 를 연다.
✓ RoamingWithIr으로 저장한다.
✓ 아래 스케치와 같이 수정한다.
✓ ABOT에서 프로그래밍 케이블을 분리시킨다.
✓ 건전지 팩을 다시 연결하고 3-point 스위치를 2에 둔다.
✓ ABOT을 운행시키거나 장애물을 피할 수 있는 곳에 둔다.
✓ RoamingWithWhiskers (별도로 접촉이 필요하지 않다)처럼 행동하는지 관찰한다.

```
/*
 * Robotics with the BOE Shield - RoamingWithIr
 * Adaptation of RoamingWithWhiskers with IR object detection instead of
 * contact switches.
 */
```

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

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

void setup()                        // Built-in initialization block
{
  pinMode(10, INPUT); pinMode(9, OUTPUT);//Left IR LED & Receiver
  pinMode(3, INPUT); pinMode(2, OUTPUT);//Right IR LED & Receiver

  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
}
```

```
void loop()                      // Main loop auto-repeats
{
  int irLeft = irDetect(9, 10, 38000);  // Check for object on left
  int irRight = irDetect(2, 3, 38000);// Check for object on right

  if((irLeft == 0) && (irRight == 0))  // If both sides detect
  {
    backward(1000);                 // Back up 1 second
    turnLeft(800);                  // Turn left about 120 degrees
  }
  else if(irLeft == 0)              // If only left side detects
  {
    backward(1000);                 // Back up 1 second
    turnRight(400);                 // Turn right about 60 degrees
  }
  else if(irRight == 0)             // If only right side detects
  {
    backward(1000);                 // Back up 1 second
    turnLeft(400);                  // Turn left about 60 degrees
  }
  else                              // Otherwise, no whisker contact
  {
    forward(20);                    // Forward 1/50 of a second
  }
}

int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);     // IRLED 38 kHz for at least 1 ms
  delay(1);                         // Wait 1 ms
  int ir = digitalRead(irReceiverPin); // IR receiver -> ir variable
  delay(1);                         // Down time before recheck
  return ir;                        // Return 1 no detect, 0 detect
```

```
}

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
}
```

실습5: 고성능 IR 방향 조정하기

이전 실습에서 프로그램에서 했던 방향 조정 양식은 더듬이에서는 괜찮았다. 그러나 IR 감지기를 사용할 때는 불필요하게 느리다. 더듬이를 사용할 때는 ABOT이 물체를 접촉해야 한다. 그런 다음 장애물 주변으로 뒤로 간다. 적외선으로는 ABOT은 대부분 장애물 앞에서 피할 것이다. 그리고 장애물 주변을 피할 수 있다.

충돌을 피하기 위해 샘플링 속도 증가시키기

여러분은 모든 주행에서 20 ms을 줄일 수 있다. 이것은 스케치를 초당 거의 50번씩 확인하고 또 확인한다는 의미이다. (이것은 실제로 조금 더 느리다. 초당 40번 정도 된다. 실행 코드 때문에 IR 감지에 시간이 너무 많이 걸린다.) ABOT이 방향 조정으로 장애물에 도달하기 전에 충돌을 피하기 위해 작은 회전을 실행할 것이다. 이런 방법으로 필요 이상으로 멀리 가지 않고 장애물 주변을 완전히 피할 수 있다. 그래서 복잡한 코스도 성공적으로 방향조정을 할 수 있다. 다음 스케치를 실험 한 후에 ABOT이 주행하는 꽤 좋은 방법이라는 것에 동의 할 것이다.

예제 스케치- FastIrRoaming

✓ 코드를 입력하고 저장한 다음 FastIrRoaming을 업로드 하시오. 그 다음 앞의 실습과 같은 장애물로 테스트 하시오.

```
/*
 * Robotics with the BOE Shield - FastIrRoaming
 * Adaptation of RoamingWithWhiskers with IR object detection instead of
 * contact switches
 */
```

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

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

```
void setup()                    // Built-in initialization block
{
  pinMode(10, INPUT); pinMode(9, OUTPUT);
                                          //Left IR LED & Receiver
  pinMode(3, INPUT);   pinMode(2, OUTPUT);
                                          // Right IR LED & Receiver

  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
}

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

  int irLeft = irDetect(9, 10, 38000); // Check for object on left
  int irRight = irDetect(2, 3, 38000);// Check for object on right

  if((irLeft == 0) && (irRight == 0)) // If both sides detect
  {
    maneuver(-200, -200, 20);    // Backward 20 milliseconds
  }
  else if(irLeft == 0)             // If only left side detects
  {
    maneuver(200, -200, 20);     // Right for 20 ms
  }
  else if(irRight == 0)            // If only right side detects
  {
    maneuver(-200, 200, 20);       // Left for 20 ms
  }
  else                               // Otherwise, no IR detects
  {
```

```
    maneuver(200, 200, 20);        // Forward 20 ms
  }
}

int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);  // IRLED 38 kHz for at least 1 ms
  delay(1);                              // Wait 1 ms
  int ir = digitalRead(irReceiverPin);// IR receiver -> ir variable
  delay(1);                        // Down time before recheck
  return ir;                       // Return 1 no detect, 0 detect
}

void maneuver(int speedLeft, int speedRight, int msTime)
{
  // speedLeft, speedRight ranges: Backward  Linear
  // Stop  Linear  Forward
  //      -200       -100......0......100        200
  // Set Left servo speed
  servoLeft.writeMicroseconds(1500 + speedLeft);
  // Set right servo speed
  servoRight.writeMicroseconds(1500 - speedRight);
  if(msTime==-1)                  // if msTime = -1
  {
    servoLeft.detach();           // Stop servo signals
    servoRight.detach();
  }
  delay(msTime);                  // Delay for msTime
}
```

FastIrRoaming 동작 설명

이 스케치는 TestManeuverFunction 스케치로부터 작동 함수를 사용한다. 이 maneuver 함수는 세 개의 매개변수 speedLeft, speedRight, 와 msTime 가 있

다. 속도 매개변수를 재 호출하는데 200을 사용하여 최대 속력으로 전진하고 -200은 뒤로 최대 속력으로 진행하고 -100에서 100사이의 값은 일정하게 속도를 조절한다. 또한 **msTime** 값은 20이고 각 maneuver는 loop 함수가 돌아오기 전에 20ms동안 실행한다.

```
void loop()                // Main loop auto-repeats
{
  int irLeft = irDetect(9, 10, 38000);  // Check for object on left
  int irRight = irDetect(2, 3, 38000);  // Check for object on right

  if((irLeft == 0) && (irRight == 0))    // If both sides detect
  {
    maneuver(-200, -200, 20);    // Backward 20 milliseconds
  }
  else if(irLeft == 0)                  // If only left side detects
  {
    maneuver(200, -200, 20);       // Right for 20 ms
  }
    else if(irRight == 0)              // If only right side detects
  {
    maneuver(-200, 200, 20);       // Left for 20 ms
  }
  else                               // Otherwise, no IR detects
  {
    maneuver(200, 200, 20);        // Backward 20 ms
  }
}
```

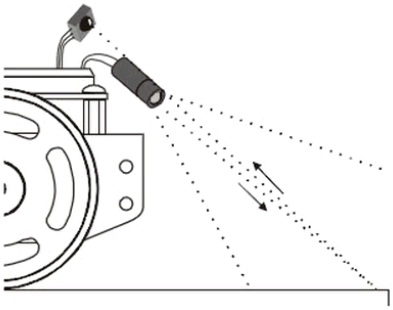
실습하기

- ✓ FastIrRoaming 을 FastIrRoamingYourTurn로 저장한다.
- ✓ ABOT이 물체를 감지한 것을 지시하는 LED를 추가하시오.
- ✓ **ABOT**의 속도를 반으로 줄이기 위해 **speedLeft**와 **speedRight**를 수정하시오. (200과 -200은 은 너무 큰 숫자이고 50은 앞으로 -50은 뒤로 속도가 반이 되는 것을 기억하시오).

활동6: 낭떠러지 감지

지금까지 로봇이 물체를 감지하였을 때 회피하는 방향 조정을 위해 프로그래밍 하였다. ABOT이 물체를 감지하지 않았을 때도 피하는 행동을 해야한다. 예를 들어 ABOT이 책상에서 주행한다면 이것의 IR 감지기는 책상의 표면을 감지하고 있다. 이 스케치는 IR 감지기가 책상의 표면을 "보면서" 계속 주행할 수 있어야 한다.

✓ 전원 케이블과 프로그래밍 케이블을 분리하시오.
✓ IR 물체 감지기가 아래쪽과 바깥쪽으로 보이도록 초점을 맞추시오.



추천 재료

(1개) 검은 비닐 전기테이프 ¾" (19 mm) 폭, 또는 검은 템페라 페인트와 브러시.
(1개) 흰 포스터 용지, 22 x 28 in (56 x 71 cm).

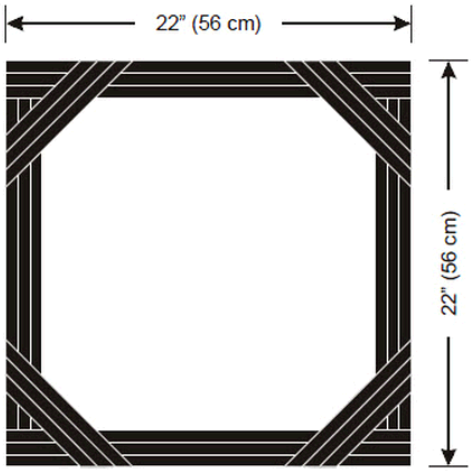
포스터 보드를 이용한 낭떠러지 시뮬레이션

흰 포스트 용지에 가장자리 부분을 전기 테이프 또는 템페라 페인트를 책상 모서리로 보이게 낭떠러지를 만들어서 시뮬레이션 한다. ABOT에 위험이 줄어든다.

✓ 전기 테이프를 이용하여 그림 7-12 에서 보이는 것처럼 만든다. 전기테이프이라면 작은 종이위에 시험용 패치를 만들어 적외선이 감지기가 그것으로 부터 반사되는 것을 볼 수 없는 것을 확인하는 것을 먼저 테스트한다. 반사된

것이 적외선이 아니라면 줄 사이에 곧은 선으로 세 줄을 만들어서 사용하는 데 그 사이에 흰 종이가 보이지 않게 한다.

- ✓ 근처에 형광빛이 ABOT의 IR 감지기에 감지되지 않도록 하기 위해 IrInterferenceSniffer 스케치를 실행시킨다.
- ✓ ABOT이 포스터 보드를 감지하는지를 확인하기 위해 TestBothIrAndIndicators를 사용한다. 그러나 전기 테이프 또는 페인트는 감지하지 않는다.



만약 ABOT이 여전히 전기 테이프를 너무 명확하게 본다면, 여기 몇 가지 해결 방법이 있다.

- ✓ IR 감지기와 LED 각도를 조금 아래로 조절 한다.
- ✓ 다른 회사의 전기 테이프를 사용해본다.
- ✓ ABOT이 좀 더 가까운 것을 보기 위해 2 kΩ를 4.7 kΩ (노랑 - 보라 빨강)으로 교체 한다.
- ✓ 서로 다른 주파수 매개변수로 tone 명령을 조정한다. 여기에 몇 가지 매개변수는 ABOT에 좀 더 가까이 보게 한다: 39000, 40000, 41000

만약 이전의 IR LED를 사용한다면 ABOT은 아마도 너무 **근시**가 되는 문제가 발생 할 수도 있다 여기에 ABOT이 물체에 정밀도를 증가시키도록 좀 더 멀리 볼 수 있게 하는 몇 가지 방법이 있다.

- ✓ 2kΩ (빨강-검정-빨강) 저항을 1kΩ (빨강-검정-갈색)으로 교체한다. 마찬가지로 470Ω(노랑-보라-갈색) 대신에 2kΩ의 LEDs로 교체한다.

✓ IR LEDs와 감지기를 각도를 아래로 좀 더 기울여서 표면의 오른쪽으로 보게 한다.

만약 테이프나 페인트로 칠한 코스로 성공한 후 책상위에서 시도해 본다면

✓ 당신은 전기 테이프 또는 페인트로 구분된 코스에서 ABOT을 실행시키기 전에 동일한 단계를 따르는 것을 기억하라!

ABOT이 탐지하는 것을 확인 한다. 여러분의 손은 로봇의 위에서 그것을 잡을 준비를 하는 것이 좋다. ABOT은 탁자로 운행 준비:

✓ 항상 탁자의 끝에 다다를 때 그 위에서 ABOT을 잡을 준비를 한다. 만약 ABOT이 가장자리를 운행하려고 한다면 빠지기 전에 그것을 받는다. 아니면 ABOT이 아닌 다른 로봇이 될 수 있다.

✓ 만약 여러분이 선 근처에 있을 경우 ABOT이 여러분을 감지할 것이다. 현재 스케치는 테이블 아래에서 여러분을 구별할 방법이 없기 때문에 로봇이 탁자 가장자리에서 계속 앞으로 나갈 수도 있다. 그래서 감지지가 보는 범위에서 떨어져 있어야 한다.

예제 스케치 : AvoidTableEdge

대부분 ABOT이 탁자 위에서 끝을 넘어가지 않게 하기 위한 프로그래밍에서는 FastIrRoaming 에서 if...else if...else 구문을수정하면 쉽게 해결 할 수 있다. 첫 번째로 뒤로가는 것 대신에 양쪽 감지기가 물체를 볼 때 20ms 전진이 필요하다. 또한 물체로부터 멀리 가는 것 대신에 물체를 향하여 돌아야 할 필요가 있고 낭떠러지를 보면 20ms 보다 더 멀리 회전해야 할 필요가 있다. 375ms 회전은 아주 좋으나 가장 좋은 수행 결과 값은 여러분이 맞추면 된다.

✓ FastIrNavigation 를 열고 AvoidTableEdge로 저장한다.
✓ 예제 스케치와 동일하게 스케치를 수정한다. maneuver 가 loop 내부에서 호출하는 것을 주의 깊게 본다. 20ms동안 전진했던 상태는 지금은 250ms 동안 뒤로 간다. 마찬가지로 뒤로 가는 상태는 20ms 동안 전진한다. 또한 20ms 동안 호출한 상태는 375ms 동안 오른쪽으로 회전하고 20ms동안 왼쪽

으로 회전하는 상태는 375ms 동안 오른쪽으로 회전한다.

✓ 전기 테이프나 페인트로 만든 코스 위에서 스케치를 테스트 한다.

✓ 만약 탁자 위에서 해본다면 이미 논의된 주의 사항들을 따르는 것을 잊지 마시오.

```
/*
 * Robotics with the BOE Shield - AvoidTableEdge
 * Adaptation of FastIrRoaming for table edge avoidance
 */

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

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

void setup()                    // Built-in initialization block
{
 // Left IR LED & Receiver
  pinMode(10, INPUT);  pinMode(9, OUTPUT);

// Right IR LED & Receiver
  pinMode(3, INPUT);  pinMode(2, OUTPUT);

  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
}

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

  int irLeft = irDetect(9, 10, 38000);// Check for object on left
```

```
  int irRight = irDetect(2, 3, 38000); // Check for object on right

  if((irLeft == 0) && (irRight == 0)) // Both sides see table surface
  {
    maneuver(200, 200, 20);        // Forward 20 milliseconds
  }
  else if(irLeft == 0)              // Left OK, drop-off on right
  {
    maneuver(-200, 200, 375);   // Left for 375 ms
  }
  else if(irRight == 0)            // Right OK, drop-off on left
  {
    maneuver(200, -200, 375);    // Right for 375 ms
  }
  else                                 // Drop-off straight ahead
  {
    maneuver(-200, -200, 250);     // Backward 250 ms before retry
  }
}

int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);   // IRLED 38 kHz for at least 1 ms
  delay(1);                            // Wait 1 ms
  int ir = digitalRead(irReceiverPin); // IR receiver -> ir variable
  delay(1);                           // Down time before recheck
  return ir;                         // Return 1 no detect, 0 detect
}

void maneuver(int speedLeft, int speedRight, int msTime)
{
  // speedLeft, speedRight ranges: Backward  Linear  Stop  Linear
  //Forward
  //       -200        -100......0......100          200
```

```
  //왼쪽 서보 속도 설정
  servoLeft.writeMicroseconds(1500 + speedLeft);

  //오른쪽 서보 속도 설정
  servoRight.writeMicroseconds(1500 - speedRight);
  if(msTime==-1)                  // if msTime = -1
  {
    servoLeft.detach();           // Stop servo signals
    servoRight.detach();
  }
  delay(msTime);                  // Delay for msTime
}
```

AvoidTableEdge 가 작동하는 방법

AvoidTableEdge 는 FastIrRoaming 에서 loop 함수에서 if…else if…else 구문을 수정한 것이기 때문에 두 개의 구분을 하나씩 보자.

```
// From FastIrRoaming

  if((irLeft == 0) && (irRight == 0))
  {
    maneuver(-200, -200, 20);
  }
  else if(irLeft == 0)
  {
    maneuver(200, -200, 20);
  }
  else if(irRight == 0)
  {
    maneuver(-200, 200, 20);
  }
  else
```

```
  {
    maneuver(200, 200, 20);
  }


//From AvoidTableEdge

  if((irLeft == 0) && (irRight == 0))
  {
    maneuver(200, 200, 20);
  }
  else if(irLeft == 0)
  {
    maneuver(-200, 200, 375);
  }
  else if(irRight == 0)
  {
    maneuver(200, -200, 375);
  }
  else
  {
    maneuver(-200, -200, 250);
  }
}
```

if((irLeft == 0) && (irRight == 0)) 반응은 IR 감지기가 둘 다 장애물을 감지했기 때문에 FastIrRoaming 는 maneuver(-200, -200, 20)으로 뒤로 간다. 반면에IR 감지기가 탁자를 감지했기 때문에 AvoidTableEdge 는 maneuver(200, 200, 20)로 앞으로 간다. 이것은 다른 20ms 동안 전진하는데 안전하다는 의미이다.

다른 if(irLeft == 0) 반응은 FastIrRoaming이 20ms 동안 오른쪽으로 회전하고 한다는 의미이다. maneuver(200, -200, 20)는 왼쪽 장애물을 피하기 위해 전

진하는 단계이다. AvoidTableEdge 탁자의 오른쪽인 낭떠러지로부터 멀리 회전한다. 만약 그렇다면 if…else if…else 구문이 else if 상태를 만들지 말아야 한다. 이것은 왼쪽 IR 감지기가 탁사를 본다는 의미이다. 그러나 오른쪽은 아니다. 그래서 이 코드는 쉴드를 maneuver(-200, 200, 375)으로 0.375 초 동안 왼쪽으로 회전한다. 이것은 오른쪽에서 감지되는 낭떠러지를 피하게 만들어야 한다.

여러분의 차례

375ms 는 탁자 모서리를 피하려고 하는 것인데 다른 응용에서는 수정할 수 있다. 예를 들면 만약 ABOT이 탁자의 끝을 꼭 껴안고 있다고 하면, 작은 회전이 좀 더 유용할 것이다. 반대로 ABOT이 영역 바깥쪽 물체를 말려고 하면 더 큰 회전(너무 많이는 아니고)이 좀 더 유용할 수 있다. 그래서 지그재그로 뒤돌아선 다음 탁자를 가로지르면서 앞으로 나간다.

maneuver 함수를 호출할 때 msTime 매개변수를 더 작게 사용함으로 작은 회원을 만드는 코드를 작성 할 수 있다. 예를 들면 만약 maneuver(-200, 200, 375)에서 375를 300으로 변경한다면 왼쪽 회전이 더 작게 될 것이다. 450으로 변경한다면 왼쪽 회전에 좀더 정밀 해 질 것이다.

- ✓ AvoidTableEdge를 수정한다. 그러면 낭떠러지 코스의 끝을 따라가게 한다. ABOT이 낭떠러지를 만날 때 작은 회전을 실행시키기 위해 maneuver 내의 msTime 매개변수 값을 수정한다. 떨어지기 전에 회전하게 만들려면 값을 얼마로 해야 할까?

- ✓ ABOT이 운행하는데 탁자 모서리를 따라가는 대신에 구역 내에서 주행하도록 하기 위해 모서리로부터 떨어진 축으로 시험한다.

제7장 요약

이 장에서는
변조된 적외선을 38kHz로 방출하고 반사를 감지하는 적외선 LED 한 쌍을 사용하는 것에 초점을 둔다. 로봇 주행 센서 시스템을 사용하여 다음 내용을 다룬다.

전자

- 적외선 LED는 무엇이고 이것의 양극과 음극을 구별하는 방법은 무엇인가?
- 변조된 적외선 수신부는 무엇이고 마이크로 컨트롤러 회로를 사용하는 방법은 무엇인가?
- 저항값이 LED를 시리즈에 연결했을 때 미치는 영향은 무엇인가?

프로그래밍

- 새로운 목적을(적외선 신호를 변조하기 위해서, 양쪽 센서를 둘 다 비슷한 출력 신호로 만드는 방법)위해 익숙한 tone 함수를 사용하는 방법은?
- 두 개의 센서와 비슷한 출력을 가진 다른 센서에 사용하기 위해 하나의 센서 시스템으로 부터 스케치를 재사용하는 방법은?
- 특별한 행동(장애물 피하기)을 위해 설계된 현재 있는 스케치를 새로운 동작으로(낭떠러지 감지) 만들기 위해 재사용하는 방법은?

로봇 기술

직접 접촉하지 않고 자율 센서 주행을 위한 물체를 감지하기 위해 적외선 방출부와 수신부 한 쌍을 사용하여 한다.

엔지니어링 기술들

- 사람이 볼 수 없는 상태(변조된 적외선 빛이 반사되는)를 센서가 감지하는 상태를 알려주기 위해 가시광선 지시자(red LED)를 사용한다.
- 서브시스템과 문제해결을 위해 더 많이 연습한다.

제8장 주행거리로 ABOT 제어하기

제7장에서 우리는 적외선 LED와 수신기를 사용하여 물체에 닫지 않고 물체를 감지하였다. 만약 IR 센서로부터 거리를 잴 수 있는 기능을 가진다면 더 멋지지 않을까? 이전 장에서 사용한 것과 동일한 하드웨어로 가까운 거리를 대략 감지할 수 있다. 이것으로도 ABOT이 충돌하지 않고 따라 갈 수 있도록 하기에 충분하다.

IR LED/ 수신 회로로 거리 결정하기

이 장에서는 이전장과 동일한 IR LED/수신기 회로를 사용한다.

필요한 부품:

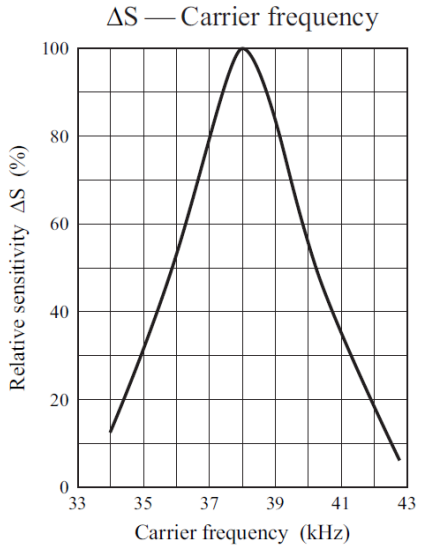
(1) 자

(1) 빈용지

- 만약 이전 장에 만든 ABot이라면 IR LED에 2 kΩ 저항을 사용하였다.
- 만약 이전 장의 회로를 분해하였다면, 7장의 실습 #1을 반복해야 한다. 또한, TestBothIrAndIndicators를 사용하여 시스템이 제대로 작동되는지 확인해야 한다.

실습1: 주파수 테스트

여기에 IR 수신기의 특정 회사의 데이터 시트가 있다.(파나소닉 PNA4602M; 여러분들의 키트에는 아마 다른 회사 제품을 사용하고 있을 것 같다).



이 그래프에서 IR 검출기는 38.5 kHz에서 가장 감도가 좋다.—곡선의 가장 높은 지점을 피크 감도(*peak sensitivity)*라 한다. 피크의 양쪽의 곡선이 얼마나 갑자기 떨어지는지 알 수 있다. 이 IR 검출기는 38kHz가 아닌 다른 주파수로 켜거나/끄면 감도가 낮아진다. 40kHz로 LED를 켜면 38kHz에 비해 감도가 반으로 줄어든다. 42 kHz에서 LED를 비추면 20% 밖에 감지를 못한다. IR LED 신호가 38 kHz로부터 더 멀어질수록 감도가 낮아지므로, 물체에 가까워야만 IR 신호의 반사파를 수신할 수 있다.

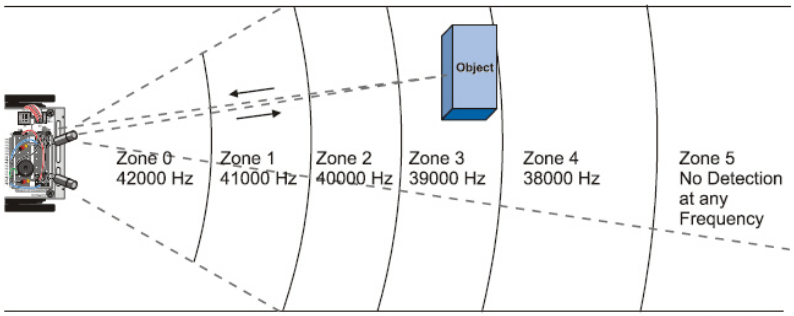
가장 감도가 높은 주파수(38 kHz)에서 가장 멀리 있는 것을 감지할 수 있다, 반면 감도가 낮은 주파수는 가까이 있는 물체만 감지할 수 있다. 이것이 물체와의 거리를 대략 계산할 수 있도록 한다. 어떤 주파수를 선택하고, 가장 감도가 좋은 것부터 가장 감도가 좋지 못한 것까지 테스트하자. 가장 감도가 높은 주파수부터 시도하자. 만약 물체가 감지되면, 다음 차례의 감도로 감지를 해보자. IR 감지기에서 더 이상 IR을 감지하지 못하는 주파수가 어떤 것인가에 따라 거리를 대략

계산할 수 있다.

!	**주파수 휩쓸기**는 다양한 입력 주파수에 대한 회로의 출력을 시험하는 기술이다.

거리 감시를 위한 주파수 휩쓸기 프로그래밍

다음의 다이어그램은 ABot이 주파수를 사용하여 거리를 어떻게 계산하는지 보여주는 예이다. 이 예에서 Zone 3 구역에 물체가 있는 경우로, 38000과 39000 Hz를 전송하면 물체를 감지할 수 있으나, 40000, 41000, 또는 42000 Hz에서는 감지할 수 없다. 만약 물체를 Zone 2 구역으로 이동하면 38000, 39000, 그리고 40000 Hz 가 전송되면 감지되나 41000 또는 42000 Hz에서는 감지되지 않는다.



NOTE: This diagram is just an illustration. The actual detection range is several cm away from the front of the BOE Shield-Bot and also only covers a few cm of distance detection.

다음 스케치의 irDistance() 함수는 위의 기술을 사용하여 거리를 측정한다. loop() 함수 코드에서 irDistance()가 호출되고 IR LED와 수신되는 양쪽 핀의 쌍을 위한 값을 전달한다. 예를 들면, 루프 함수는 ABot의 검출 존인 “보이는 영역”의 왼쪽에서 물체와의 거리를 int irLeft = irDistance(9, 10)로 확인한다.

```
// IR 거리 측정 함수
int irDistance(int irLedPin, int irReceivePin)
{
```

```
  int distance = 0;
  for(long f = 38000; f <= 42000; f += 1000) {
    distance += irDetect(irLedPin, irReceivePin, f);
  }
  return distance;
}
```

만약 물체가 없으면 irDetect() 함수는 1을 리턴하고, 물체가 있으면 0을 리턴 한다.

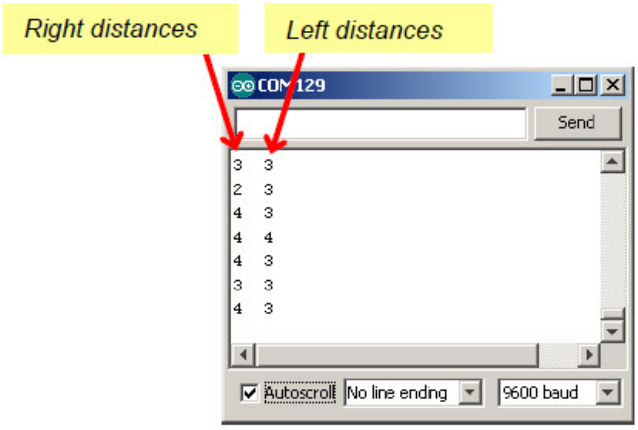
distance += irDetect(irLedPin, irReceivePin, f); 수식은 물체가 검출되지 않은 수를 세어본다. 루프를 마치면, distance 변수는 IR 감지기가 물체 감지를 못하였는지를 세어서 넣는다. 물체를 감지하지 못할수록 IR 감지기가 물체가 인식하지 못한 것이다. 물체를 인식하지 못할수록 더 멀리 있는 셈이다. 그래서 distance 값은 앞에 나온 그림에서처럼 존(zone)의 번호를 리턴하게 된다.

양쪽의 거리를 출력

양쪽 IR LED/수신기 쌍이 대충 같게 거리를 감지하는지 테스트하는 것은 중요하다. 똑 같을 필요는 없지만 너무 다르면 ABOT이 앞에 있는 물체를 따라가기가 힘이 드는데, 왜냐하면 한 쪽 방향으로 자꾸 돌아가려 하기 때문이다. 부(sub)시스템 테스트를 많이 할 필요가 있다.
거리 측정이 일치하지 않게 되는 공통적인 이유로는 IR LED에 저항이 서로 같지 않은 경우이다. 예를 들면 한 쪽 IR LED는 1 kΩ 저항이고 다른 쪽 저항은 2 kΩ 저항이면, 한쪽이 다른 쪽 보다 물체가 가까워 보인다. 드물긴 하지만 다른 가능성으로는 한쪽 IR 감지기가 훨씬 다른 쪽 보다 감도가 높은 경우가 있다. 이 경우는, 좀 큰 저항을 해당 하는 쪽에 달면 IR 헤드라이트의 강도가 조금 줄어들고 차이 나던 측정값이 비슷해진다.

예제 스케치 – DisplayBothDistances

다음의 캡처 된 화면은 시리얼 모니터에서 DisplayBothDistances 프로그램으로 존의 감지를 측정한 것이다. 노이즈로 인해서 값이 조금은 들쭉날쭉 하는데, 같은 쌍끼리 비교해서 1의 값 정도가 서로 차이 나는 것은 문제가 되지 않는다.



- 아두이노에 DisplayBothDistances 코드를 입력하고, 저장한 후 ABOT에 업로드 한다.
- 박스(Box), 책, 병 또는 유사한 물체를 타깃으로 활용하여 거리 검출기를 테스트 하자.

- ABOT으로부터 떨어졌다가 물체를 가까이 이동시켜 보자. 아주 작게 이동해도 값이 변하는 지점까지 옮겨보자.
- 각각 검출 존의 거리 중심 지점을 찾고 기록하자.
- 오른쪽 IR 검출기에 대해서 반복하자.
- 만약 한쪽이 다른 쪽보다 두 배 더 먼 거리를 검출한다면, IR LED에 연결된 저항을 확인해 보자. 양쪽 모두 2 kΩ (붉은색-검은색-붉은색) 이어야 한다. 서로 같지 않으면 IR LED 저항을 조절하여 양쪽이 비슷하게 되도록 하자.

```
/*
 * ABOT - DisplayBothDistances
 * 왼쪽과 오른W고의 IR 상태를 시리얼 모니터에 출력
 * 거리 범위 0에서 5까지.  수 cm의 매우 짧은 거리
 * 각각의 감지기 앞에서 특정 할 수 있다. 대부분은 0은 너무 가깝고
 * 5는 너무 멀다.
 */
void setup(){                                  // Built-in initialization block
  tone(4, 3000, 1000);                         // Play tone for 1 second
  delay(1000);                                 // Delay to finish tone
  pinMode(10, INPUT);  pinMode(9, OUTPUT);     // Left IR LED & Receiver
  pinMode(3, INPUT);  pinMode(2, OUTPUT);      // Right IR LED & Receiver
  Serial.begin(9600);                          // Set data rate to 9600 bps
}
void loop(){                                   // Main loop auto-repeats
  int irLeft = irDistance(9, 10);              // Measure left distance
  int irRight = irDistance(2, 3);              // Measure right distance
  Serial.print(irLeft);                        // Display left distance
  Serial.print("   ");                         // Display spaces
  Serial.println(irRight);                     // Display right distance
  delay(100);                                  // 0.1 second delay
}
// IR distance measurement function
int irDistance(int irLedPin, int irReceivePin){
  int distance = 0;
```

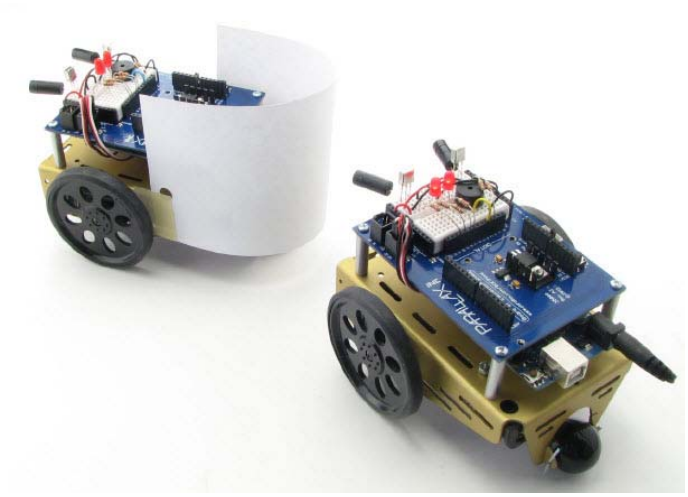
```
  for(long f = 38000; f <= 42000; f += 1000) {
    distance += irDetect(irLedPin, irReceivePin, f);
  }
  return distance;
}
// IR Detection function
int irDetect(int irLedPin, int irReceiverPin, long frequency){
  tone(irLedPin, frequency, 8);              // IRLED 38 kHz for at least 1 ms
  delay(1);                                   // Wait 1 ms
  int ir = digitalRead(irReceiverPin);       // IR receiver -> ir variable
  delay(1);                                   // Down time before recheck
  return ir;                                 // Return 1 no detect, 0 detect
}
```

여러분의 차례 – 거리에 대한 다양한 테스트

- 다른 물체를 이용하거나 색깔이 다르거나 재질이 다른 종이나 물체를 이용해서 거리를 측정해보세요. 어떤 색상과 표면이 가장 잘 검출이 되나요? 가장 잘 되지 않는 것은?

실습2: ABOT 추적하기

선두를 따라가는 ABOT을 위해서는 선두와의 대충 거리를 알아야 한다. 만약 선두가 너무 멀어지면 스케치는 이것을 감지하고 ABOT을 앞으로 움직여야 한다. 이와 같이 너무 선두와 너무 가까우면, 스케치는 감지를 하고 ABOT을 뒤로 움직여야 한다. 이 스케치의 목적은 ABOT과 선두-물체나 앞에 있는 ABOT과의 일정 거리를 유지시키는 것이다.

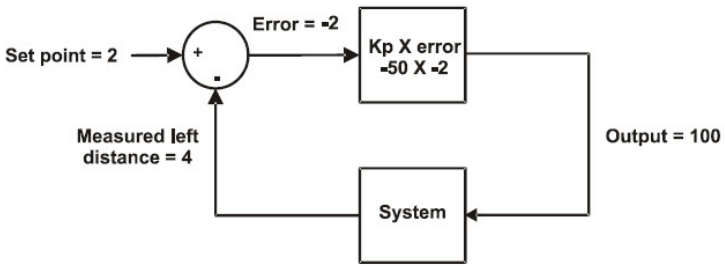


제어 시스템의 몇 가지 용어들

기계가 자동으로 측정된 값으로 관리되면 이를 제어 시스템(*control system)*이라 한다. 시스템이 유지하려는 값을 설정점(*set point*)이라 한다. 전자 제어 시스템에서는 센서(*sensor*)가 측정을 하고 설정점으로 되돌아가기 위해 기계적 작동 장치(*actuators*)를 트리거링에 의해 반응을 하기 위해 프로세서(*processor*)를 사용한다. 우리 기계는 ABOT이다. 측정된 값으로 우리가 유지하기 원하는 것은 선두 물체와의 거리이므로 설정점이 2이다. (2번 존).
기계의 프로세서는 아두이노이다. IR LED/수신기 쌍은 선두 물체까지의 거리 값을 측정하는 센서이다. 만약 측정된 값이 설정점 거리와 다르면, 필요에 따라 서보는 기계적 동작 장치로 ABOT을 앞으로나 뒤로 이동한다.

비례 제어 내부 살펴보기

연속해서 값을 측정하고 설정된 값을 유지하기 위해서 오류의 비율에 따라 출력을 조절하는 *닫힌 루프(Closed loop)* 제어는 ABOT 그림자 차량에서 잘 작동이 된다. 사실, 아래 그림의 제어 루프에서 핵심은 스케치에서 한 줄의 코드로 줄여진다. 이 블록 다이어그램(*block diagram*)은 비례 제어 프로세스를 설명하는데, ABOT은 바퀴의 속도를 IR LED/수신기 쌍이 측정한 거리 감지에 기초하여 제어한다.



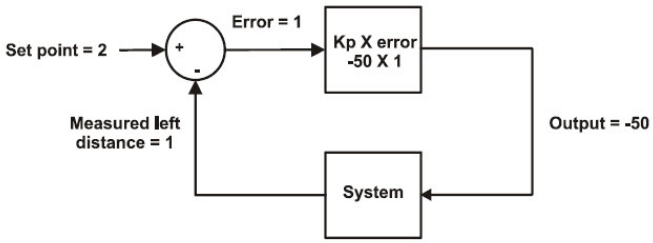
이 블록 다이어그램은 왼쪽 IR 거리 센서에 적용할 수 있고 서보의 출력 또는 오른쪽에도 적용할 수 있다. 사실, 여러분들의 코드는 동일한 제어 루프를 가지고 있으며, ABOT의 양쪽을 위해 각각 하나씩 가지고 있다.

다음 예제를 따라가 보자. 왼쪽 위쪽에서 설정점은 set point = 2;로 2이다. 우리는 ABOT이 선두 물체와 거리가 2지역(zone-2)으로 유지되도록 희망한다. 아래 쪽의 측정된 거리는 4지역으로 선두 물체와 너무 멀리 떨어져 있다.

원으로 된 심볼(원은 합쳐지는 지점: *summing junction* 이라함)로 화살표가 되어 있는데 +는 설정점에 대해 더하며 빼기 -는 측정된 거리의 에러(*error*)로 여기서는 2 - 4 = -2 가 된다.

그 다음 에러(error) 값은 위쪽이 박스로 들어간다.—작동 블록(*operator block*) 이 블록은 에러 값에 (Kp: *proportionality constant*) -50을 곱한다. 이 예제에서 작동 블록은 -2 × -50 = 100, 따라서 100 이 출력(*output*)이다. 스케치에서, 이 출력 값은 maneuver() 함수로 전달된다. 이것은 선두 물체로 향해 가까이 가기 위해 전 속력으로 전진한다.

다음 블록 다이어그램은 다른 예를 보여 준다. 이번에는 측정된 거리가 1이다, 선두 물체에 대해서 너무 가깝다는 의미이다. 그래서 이번에는 에러가 1이며, 1×-50 = -50. -50 이 maneuver() 함수에 전달되고 서보를 반대방향으로 반속도로 돌려서 ABOT이 선두 물체로부터 뒤로 물러나게 한다.



루프의 그 다음을 통해서 측정되는 거리 값은 바뀔 수 있지만 그래도 괜찮다. 거리를 측정함에도 불구하고 이 제어 루프는 어떤 오류도 정정하면서 이동하도록 서보를 움직인다. 정정은 항상 에러에 비례(*proportional*)한다. 관련된 두 가지 계산:

설정점(set point) - 측정거리(measured distance) = error;
오류(error) x Kp = 구동을 위한 출력(output for maneuver)

…스케치를 위해서 쉽게 합쳐질 수 있고 순서를 변경하여 하나의 수식으로 만들 수 있음:

(구동을 위한 출력:Output for maneuver) =
(거리설정점:Distance set point - 측정된거리:Measured distance) x Kp

만약 스케치를 더 자세히 살펴보고 싶다면, FollowingABot이 어떻게 작동되는지 살펴보자.

여러분의 차례 – 다른 거리로 제어 루프를 검증하기

- 비례 제어 루프가 모든 6개의 측정 거리에 대해서 바르게 응답하는 것을 증명하기 위해서 다음 테이블을 채우세요.
- 블록 다이어그램에서 6개 중에 이미 2개를 채워 두었으므로 4개만 채

우면 된다.

Condition	Measured Distance	Set Point	Error (set point–measured)	Output (Kp × error)	Maneuver Result
	0	2			
Too close	1	2	2–1=1	2×–50=–50	Slow reverse
Just right	2	2			
	3	2			
Way too far	4	2	2–4=–2	–2×–50=–100	Fast forward
	5	2			

예제 스케치: FollowingABot

FollowingABot 스케치에서 비례 제어 루프는 1초당 약 25번 토의를 한다. 그래서 모든 비례 제어 계산과 서보의 속도는 초당 25번 갱신이 된다. 이 결과로 ABOT은 손이나, 책이나 다른 로봇을 따라 갈 수 있다.

- FollowingABot 코드를 입력하고, 저장하고 업로드 하자.
- 8 ½ x 11" 종이를 이용해서 장애물 벽을 ABOT앞에 두어서 종이와 ABOT이 일정한 거리를 유지할 수 있도록 하자.(이것은 이전에 언급한 노이즈로 인해서 약간 덜석 덜석 한다)
- 종이를 살며시 회전시켰을 때 ABOT이 따라서 회전해야 한다.
- 종이 시트를 ABOT 주변으로 이끌어 간다. ABOT도 이것을 따라서 앞으로 가야 한다.
- 종이를 ABOT에 가깝게 했을 때 ABOT은 뒤로 물러나야 한다.

```
/*
 * ABOT - FollowingABot
 * Use proportional control to maintain a fixed distance between
 * ABOT and object in front of it.
 */

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

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

const int setpoint = 2;                         // Target distances
const int kpl = -50;                            // Proportional control constants
const int kpr = -50;

void setup()                                    // Built-in initialization block
{
  pinMode(10, INPUT);  pinMode(9, OUTPUT);    // Left IR LED & Receiver
  pinMode(3, INPUT);  pinMode(2, OUTPUT);     // Right IR LED & Receiver

  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
}

void loop()                                     // Main loop auto-repeats
{
  int irLeft = irDistance(9, 10);             // Measure left distance
  int irRight = irDistance(2, 3);             // Measure right distance

  // Left and right proportional control calculations
  int driveLeft = (setpoint - irLeft) * kpl;
  int driveRight = (setpoint - irRight) * kpr;

  maneuver(driveLeft, driveRight, 20);        // drive levels set speeds
}

// IR distance measurement function
```

```
int irDistance(int irLedPin, int irReceivePin)
{
  int distance = 0;
  for(long f = 38000; f <= 42000; f += 1000) {
    distance += irDetect(irLedPin, irReceivePin, f);
  }
  return distance;
}

// IR Detection function

int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);              // IRLED 38 kHz for at least 1 ms
  delay(1);                                  // Wait 1 ms
  int ir = digitalRead(irReceiverPin);       // IR receiver -> ir variable
  delay(1);                                  // Down time before recheck
  return ir;                                 // Return 1 no detect, 0 detect
}

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
}
```

FollowingABot이 어떻게 작동할까?

FollowingABot은 3개의 전역 변수를 선언한다. setpoint, kpl, 및 kpr. 어느 위치에서든지 setpoint는 상수로 값이 2이다. 이와 같이 kpl은 -50이고, kpr도 상수로 -50이다.

```
const int setpoint = 2; // 목표거리
const int kpl = -50;    // 비례 제어 상수
const int kpr = -50;
```

!	이런 값들을 상수로 정하면 편리한 점은 변경이 필요할 때 스케치의 첫 부분에 있는 이들 값을 쉽게 변경할 수 있다. 스케치 첫 부분에 있는 이 값들은 사용된 모든 곳에 영향을 미친다. 만약 kpl을 -50에서 -45로 바꾸면 kpl을 사용하는 모든 곳의 값은 -45로 바뀐다. 이것은 특히 비례제어 루프에서 오른쪽과 왼쪽값을 지정하는 실험을 할 때 유용하다.

loop() 함수에서의 첫 작업은 현재 거리를 특정하기 위해 irDistance() 함수를 호출한다. 그리고 irLeft와 irRight 변수 값에 복사한다.

```
void loop()                                  // 자동 반복을 위한 루프
{
  int irLeft = irDistance(9, 10);           // 왼쪽 거리 측정
  int irRight = irDistance(2, 3);           // 오른쪽 거리 측정
```

간단한 제어 루프 계산법은 기억하고 있나요?

구동을 위한 출력: Output for maneuver =
(설정된 거리값:Distance set point - 측정된 거리값: Measured distance) x Kp

다음의 두 줄은 오른쪽과 왼쪽의 제어 루프를 위한 계산을 수행하고 동작의 출력 값으로 변수 driverLeft와 driveRight에 저장한다.

```
  // 왼쪽과 오른쪽의 비례 제어 계산
```

```
int driveLeft = (setpoint - irLeft) * kpl;
int driveRight = (setpoint - irRight) * kpr;
```

이제, driveLeft와 driveRight는 서보 속도를 설정하기 위한 구동 함수로 전달 될 준비가 되었다.

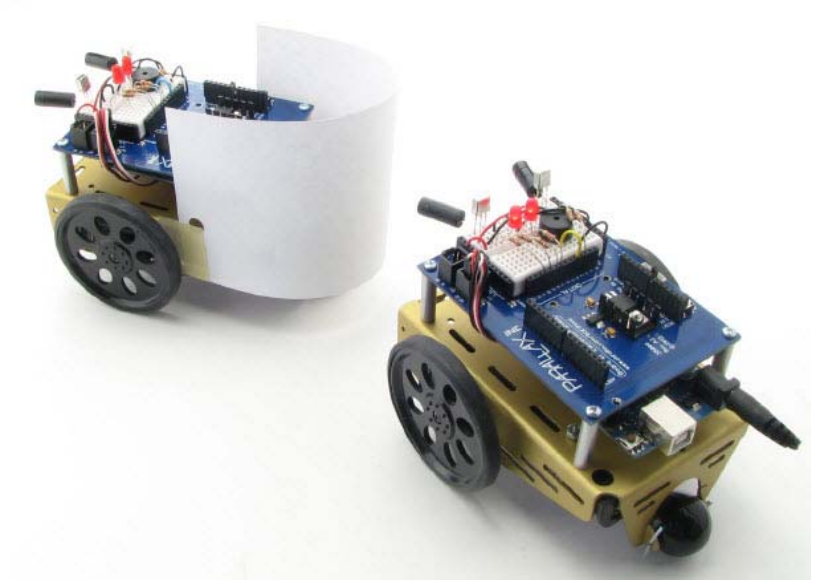
```
maneuver(driveLeft, driveRight, 20);  // 속도 설정을 위한 구동 값
```

각 호출에서 구동이 20ms만 지속되므로, 루프 함수를 20ms동안 반복해서 지연을 시킨다. IR 거리 검색에서 20ms가 또 걸리므로 루프는 40ms 반복 시간을 갖는다. 샘플링 비율의 측면에서 초당 25 샘플에 해당한다.

!	**샘플링 비율에 대한 샘플링 간격** 샘플링 간격은 하나의 샘플에서 다음 샘플까지의 시간이다. 샘플링 비율은 샘플 하는 주기를 나타낸다. 만약 다음 공식을 기억하면 쉽게 계산할 수 있다: *sampling rate = 1 ÷ sample interval*

선두 따라가기

여기서는 리더 ABOT을 그림자 ABOT이 따라 오도록 해보자. 선두 로봇은 FastIrRoaming (주행 속도를 +/- 40으로 낮추어)의 수정된 버전으로 움직인다. 그림자 ABOT은 FollowingABot을 실행하여 움직인다. 하나의 리더 로봇은 체인을 이어서 6~7개의 그림자 ABOT을 따라오게 한다. 체인 안에 있는 그림자 ABOT들에게 종이 패널만 추가하면 된다.



- 만약 ABOT이 1대만 작동된다면, 여러분이 리더가 될 수 있다. 선두 물체로는 책이나 병이나 또는 손을 활용할 수 있다.
- 2개 이상의 ABOT이 있다면, 꼬리 부분에 종이 패널을 장착하고 그림에 있는 것처럼 그림자 AbOT에게 리더 로봇이 더 잘 보이도록 하자. 그림자 로봇들을 체인으로 만든다면, 각각에 종이 패널을 끼워야 한다.
- SlowerIrRoamingForLeaderBot 이 리더 ABOT에 설치되는 프로그램이다.
- 그림자 ABOT에는 각각 FollowingABot 프로그램을 넣는다. 각각의 그림자 로봇 IR LED는 수평을 유지하면서 약간 왼쪽과 오른쪽을 향하도록 한다(위 아래로 향하면 안 된다).
- 그림자 ABOT을 선두 ABOT이나 다른 안내하는 물체의 뒤에 두자. 그림자 ABOT은 고정된 거리로 선두를 따라 가야 하며, 손이나 가까운

벽 등의 다른 물체로 부터는 방해를 받지 않도록 해야 한다.

```
/*
 * ABOT - SlowerIrRoamingForLeaderBot
 * Adaptation of RoamingWithWhiskers with IR object detection instead of
 * contact switches
 */

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

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

void setup()                                // Built-in initialization block
{
  pinMode(10, INPUT);
  pinMode(9, OUTPUT);   // Left IR LED & Receiver
  pinMode(3, INPUT);
  pinMode(2, OUTPUT);    // Right IR LED & Receiver

  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
}

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

  int irLeft = irDetect(9, 10, 38000);      // Check for object on left
  int irRight = irDetect(2, 3, 38000);      // Check for object on right

  if((irLeft == 0) && (irRight == 0))       // If both sides detect
```

```
  {
    maneuver(-40, -40, 20);                  // Backward 20 milliseconds
  }
  else if(irLeft == 0)                      // If only left side detects
  {
    maneuver(40, -40, 20);                   // Right for 20 ms
  }
  else if(irRight == 0)                     // If only right side detects
  {
    maneuver(-40, 40, 20);                   // Left for 20 ms
  }
  else                                       // Otherwise, no IR detects
  {
    maneuver(40, 40, 20);                    // Forward 20 ms
  }
}

int irDetect(int irLedPin, int irReceiverPin, long frequency)
{
  tone(irLedPin, frequency, 8);              // IRLED 38 kHz for at least 1 ms
  delay(1);                                   // Wait 1 ms
  int ir = digitalRead(irReceiverPin);       // IR receiver -> ir variable
  delay(1);                                   // Down time before recheck
  return ir;                                 // Return 1 no detect, 0 detect
}

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 차례 – 상수를 가지고 실습하기

그림자 ABOT의 행동을 바꾸기 위해서 지점과 비례상수의 설정을 조정할 수 있다. 다음의 실험을 하는 동안 손이나 종이 조각을 사용하여 그림자 ABOT을 따라 오도록 하자.

- kpr과 kpl 상수 값을 15에서 100까지를 이용하여 FollowingABot 움직이도록 하자. 물체를 따라올 때의 ABOT이 어떻게 다르게 반응하는지 기록하자.
- 세팅 지점 상수를 조정하기 위해 0~4까지의 값을 시도해 보자.

어떤 이상한 행동을 하게 되는데 한 예로 만약 0으로 포인트 값을 지정하면 뒤로 움직이지 않는다. 왜 그런지 살펴볼까?

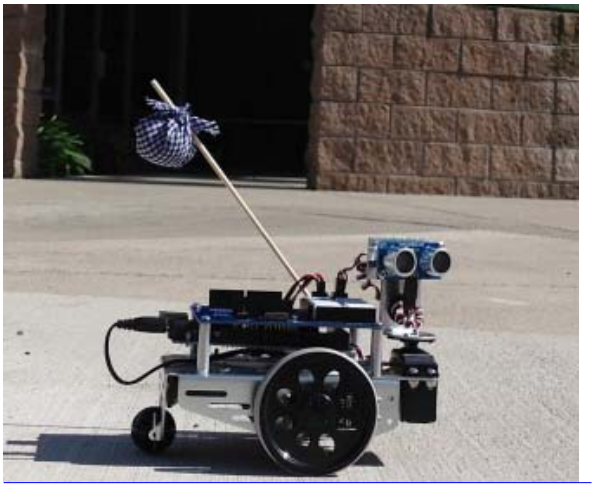
- 실습 #1에서 포인트 값이 0이 될 때까지 제어 루프 실습을 반복하자. 측정된 어떤 값으로 포인트 값을 0으로 하고 뒤로 이동하도록 할 수 있을까?

실습3: 다음은 무엇을 할까?

축하 한다! 이 책을 성공적으로 마쳤다. 이제 여러분들은 기초를 마쳤으며, ABOT에 기능을 추가하거나 더 자세히 연구할 준비가 되었으며 여기에 몇 가지 아이디어가 있다:

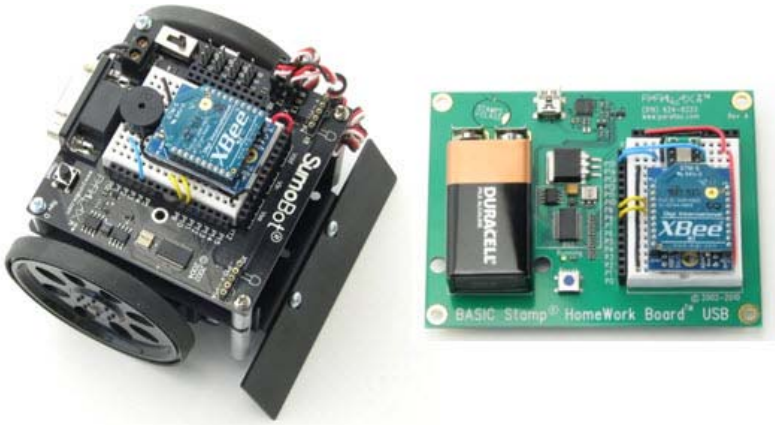
Ping))) 초음파 거리 센서

- Ping)) sensor: #28015 at the Parallax store
- Check out the Ping))) KickStart code for Arduino here
- Mounting Bracket Kit #570-28015 at the Parallax store
- Try the Roaming Ping))) Shield-Bot project here



기울기 감지를 위한 두 축 가속도계

- #28017 at the Parallax store
- Check out the Accelerometer KickStart code for Arduino here.
- See the XBee Tilt-control SumoBot project here

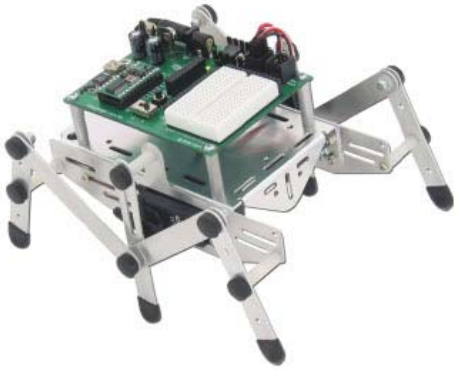


무선 제어와 통신을 위한 XBee RF 모듈과 어댑터;

- Visit the XBee page at the Parallax store for options
- Check out the XBee KickStart code for Arduino here
- See the XBee Tilt-control SumoBot project here

6-족 걷기로 기어가는 ABot

- #30055 at the Parallax store



모든 지형 탐색을 위한 탱크 발자국 키트

- #28106 at the Parallax store

계속 조사해보고 즐거운 시간 가지자!!

제8장 요약

이장에서는 뒤따라가는 ABOT에서 IR LED/수신기 쌍이 거리를 감지하는데 사용되었다. 새로운 몇 가지 개념에 대해서 친숙해 졌다:

전자

- IR 수신기의 피크 감도 찾기.
- IR 수신기의 감도 특징 곡선을 활용한 주파수 휩쓸기를 이용한 거리 감지.
- 양쪽 IR LED 수신기의 감도에 따라 저항 값 맞추기- 하드웨어 조정하기.

프로그래밍

- 주파수 휩쓸기를 생성하기 위한 for 반복 루틴을 작성하기.
- 비례 제어 루프를 활용하기 위한 스케치의 작성.
- 정수 상수 const int를 이용하여 비례 제어 루프의 설정점과 비례 계수를 설정하기
- 제어 루프의 비례 상수를 세밀하게 조절하는 시스템- 소프트웨어로 조정하기
- 샘플링 주기와 샘플링 간격이란 무엇이며 서로 어떤 관련이 있는지.

로봇 스킬

- 그림자 차량의 자동 주행을 위한 프로세서, 센서 및 액추에이터에 의한 폐쇄 비례 제어 시스템의 설정.

공학 스킬

- 간단한 폐회로 비례 제어 시스템을 위한 합치는 부분과 작동 블록 등의 블록다이어그램의 이해.
- 폐쇄 제어 시스템의 설정점, 에러, 비례 상수 및 출력값 등에 대한 이해.
- 부시스템의 테스트와 문제해결

제8장 도전

질문

1. FREQOUT 명령어로 35 kHz 주파수를 보내게 될 때 IR 탐지기에서 수신감도는 어떻게 됩니까? 36 kHz 일 때의 수신 감도는?
2. 상수를 선언할 때 사용되는 키워드는?
3. IR 거리 측정하기 위해 주파수의 리스트를 휩쓸기 위해 사용되는 문장은?
4. 접합 지점(summing junction) 이란?
5. 우리의 비례 제어를 위한 블록 폐쇄회로 블록다이어그램에서 오류 항목을 계산하기 위해 사용하는 2가지 값은?
6. 루프 함수에서 샘플 비율을 설정하기 위해서 지연을 어떻게 해야 하는가?

연습문제

1. 5개 대신 4개의 값만 주파수 휩쓸기(sweep) 하기 위한 코드의 세그 먼트를 작성하시오
2. kpl을 -45와 kpr을 -55fh 초기화하기 위한 전역 상수를 적으세요.

프로젝트

1. 손으로 ABOT의 앞부분을 잡고 뒤쪽으로 밀면 회전 없이 앞으로 나가도록 스케치를 적으세요.

제8장 해답

질문 해답

1. 상대감도는 35 kHz 에서 30%이고, 36 kHz에서는 50%.
2. 변수 선언 앞에 const 키워드 넣기.
3. 38,000에서 인덱싱을 시작하고 각 반복에서 1,000으로 증가 루프.
4. 접합 지점은 블록다이어그램의 한 부분으로 2개의 입력이 함께 추가되어(또는 하나가 다른 하나에 감산되어) 하나의 출력이 되는 것.
5. 오류 항목은 원하는 설정점의 수준으로부터 감산되는 수준을 측정한 값.
6. loop() 함수의 각 반복으로 거리의 샘플이 취해진다면 어느 정도는 각 샘플 사이에 걸리는 시간이 얼마나 될지 결정해야 한다. 이것을 샘플 간격 이라 하고 1 ÷ 샘플 간격(sample interval) = 샘플 율(sampling rate).

연습문제 해답

1. 상태 조건을 f <=42000 에서 f <= 41000 로 낮춘다.

```
for(long f = 38000; f <= 42000; f += 1000) {
  distance += irDetect(irLedPin, irReceivePin, f);
}
```

2. 선언

```
const int setpoint = 2;      // 목표 거리
const int kpl = -45;         // 비례 제어 계수
const int kpr = -55;
```

프로젝트 해답

1. 빠르고 간단한 해법은 FollowingABot 스케치에서 driveLeft와 driveRight 값의 평균을 취하는 것이다. 이 하나의 값을 조정에서 오른쪽과 왼쪽 속도 파라미터로 적용할 수 있다.

```
void loop()                                // Main loop auto-repeats
{
  int irLeft = irDistance(9, 10);      // Measure left distance
  int irRight = irDistance(2, 3);      // Measure right distance

  // Left and right proportional control calculations
  int driveLeft = (setpoint - irLeft) * kpl;
  int driveRight = (setpoint - irRight) * kpr;

  int drive = (driveLeft +  driveRight)/2; // Average drive levels

  maneuver(drive, drive, 20);          // Apply same drive to both

  delay(10);                           // 0.1 second delay
}
```