Arduino Hands-On 2
CS5968 / ART4455

Disclaimer

- Many of these slides are mine
- But, some are stolen from various places on the web
  - todbot.com – Bionic Arduino and Spooky Arduino class notes from Tod E.Kurt
  - ladyada.net – Arduino tutorials by Limor Fried
Getting Input (Digital)

- Switches make or break a connection
- But Arduino wants to see a voltage
  - Specifically, a “HIGH” (5 volts)
  - or a “LOW” (0 volts)

How do you go from make/break to high/low?

Switches

- Digital inputs can “float” between 0 and 5 volts
- Resistor “pulls down” input to ground (0 volts)
- Pressing switch sets input to 5 volts
- Press is HIGH
  Release is LOW

Why do we need the “pull down” resistor?
Another Switch

- Resistor pulls up input to 5 volts
- Switch sets input to 0 volts
- But now the sense is inverted
  - Press is LOW
  - Release is HIGH

“pull-up”

A Switch

- “gap”
- Connect when pushed
- Always connected together

Pressing the button, “closes the gap”
Using a Switch

Using digitalRead()

- In setup(): use pinMode(myPin, INPUT) to make pin an input
- In loop(): use digitalWrite(myPin) to get switch position
  - If doing many tests, use a variable to hold the output value of digitalRead().
  - e.g. val = digitalRead(myPin)
digitalRead(pin);

// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;  // the number of the pushbutton pin
const int ledPin = 13;    // the number of the LED pin

// variables hold values that will change:
int buttonState = 0;       // variable for reading the pushbutton status

void setup() {
    pinMode(ledPin, OUTPUT); // initialize the LED pin as an output:
    pinMode(buttonPin, INPUT); // initialize the pushbutton pin as an input:
}

void loop(){
    buttonState = digitalRead(buttonPin); // read the state of the pushbutton value:

    if (buttonState == HIGH) {
        digitalWrite(ledPin, HIGH); // turn LED on:
    } else {
        digitalWrite(ledPin, LOW); // turn LED off:
    }
}

Moving on…

- Write a program that reads the value on an input pin
  - Use the button to change from blinking fast to blinking slow
Moving on…

```
int ledPin = 13; // choose the pin for the LED
int inPin = 7;  // choose the input pin (for a pushbutton)
int val = 0;    // variable for reading the pin status
int delayVal = 100;

void setup(){
    pinMode(ledPin, OUTPUT); // declare LED as output
    pinMode(inPin, INPUT);   // declare pushbutton as input
}

void loop(){
    val = digitalRead(inPin); // read input value

    if( val == HIGH )
        delayVal = 1000;
    else
        delayVal = 100;

    digitalWrite(ledPin, HIGH); // blink the LED and go OFF
    delay(delayVal);
    digitalWrite(ledPin, LOW);
    delay(delayVal);
}
```

Multiple Switches

Same sub-circuit, just duplicate
Make Your Own Switches

- Anything that makes a connection
- Wires, tin foil, tinfoil balls, ball bearings
- Pennies!
- Nails, bolts, screws

- Or repurpose these tiny switches as bump detectors or closure detectors
Analog Input

To computers, analog is chunky

- Many states, not just two (HIGH/LOW)
- Number of states (or “bins”) is resolution
- Common computer resolutions:
  - 8-bit = 256 states
  - 16-bit = 65,536 states
  - 32-bit = 4,294,967,296 states
Analog Input on Arduino

- Our version uses ATMega328p
  - six ADC inputs (Analog to Digital Converter)
  - Voltage range is 0-5v
  - Resolution is 10 bits (digital values between 0-1023)
  - In other words, 5/1024 ~ 4.8mV is the smallest voltage change you can measure

- `analogRead(pin);`
  - reads an analog pin
  - returns a digital value between 0-1023
  - analog pins need no pinMode declaration

---

Sure sure, but how to make a varying voltage? With a *potentiometer*. Or just *pot*.

- [Diagram of a potentiometer](image-url)
Potentiometers

Moving the knob is like moving where the arrow taps the voltage on the resistor

- turned anti-clockwise
  - 0 volts
  - +5V
  - gnd

- turned clockwise
  - 5 volts
  - +5V
  - gnd

- somewhere in the middle
  - 2.3 volts
  - +5V
  - gnd

Arduino Analog Input

- Red to Vcc
- Purple to A0
- Blue to Gnd
```c
int sensorPin = 0;      // select the input pin for the potentiometer
int ledPin = 13;          // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from the sensor

void setup() {
  pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT:
  // Note that you don’t need to declare the Analog pin – it’s always input
}

void loop() {
  sensorValue = analogRead(sensorPin); // read the value from the sensor:
  digitalWrite(ledPin, HIGH); // turn the ledPin on
  delay(sensorValue); // stop the program for <sensorValue> milliseconds:
  digitalWrite(ledPin, LOW); // turn the ledPin off:
  delay(sensorValue); // stop the program for for <sensorValue> milliseconds:
}
```

Moving on…

- Write a program to read an analog value from a pot and use that value to control the brightness of an LED
- Fade the LED by turning the pot
- Useful function is `map(value, fromlow, fromhigh, tolow, tohigh);`
  
  ```
  y = map(x, 0, 1023, 50, 150);
  ```
- Also remember `analogWrite(pin,value);`
- PWM value from 0-255
potFade

```c
int potPin = 0;   // the analog input pin from the pot
int ledPin = 9;   // pin for LED (a PWM pin)
int val;    // Variable to hold pot value

void setup () {
    pinMode(ledPin, OUTPUT);   // declare ledPin as output
    pinMode(potPin, INPUT);                 // potPin is in input
}

void loop() {
    val = analogRead(potPin);   // read the value from the pot
    val = map(val, 0, 1023, 100, 255);  // map to reasonable values
    analogWrite(ledPin, val);
}
```

What good are pots?

- Anytime you need a ranged input
  - (we’re used to knobs)
- Measure rotational position
  - steering wheel, etc.
- But more importantly for us, potentiometers are a good example of a resistive sensor
Sensing the Dark

- Pots are example of a \textit{voltage divider}
- Voltage divider splits a voltage in two
- Same as two resistors, but you can vary them

\[ +5V \quad \text{resistor} \quad +5V \]

Sensing the Dark: Photocells

- aka. photoresistor, light-dependent resistor
- A \textit{variable} resistor
- Brighter light $\Rightarrow$ lower resistance
- Photocells you have range approx. 0-10k

\[ \text{schematic symbol} \]
Photocell Circuit

![Circuit Diagram]

Photocell Arduino Sketch

Can use as before, sketch “analog_read_led”

```c
int potPin = A2; // select the input pin for the potentiometer
int ledPin = 13; // select the pin for the LED
int val = 0; // variable to store the value coming from the sensor

void setup() {
  pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT
}

void loop() {
  val = analogRead(potPin); // read the value from the sensor
  digitalWrite(ledPin, HIGH); // turn the ledPin on
  delay(val); // stop the program for some time
  digitalWrite(ledPin, LOW); // turn the ledPin off
  delay(val); // stop the program for some time
}
```

Wave your hand over it = blink faster
Point it towards the light = blink slower
Moving on…

- Connect a photocell instead of a pot to your fading circuit
- Do you get the same range of fade as with the pot?
- Why or why not?

Resistive sensors

- thermistor (temperature)
- force sensors (pressure)
- flex sensor (bend, deflection)
- also air pressure and others
LED Brightness Functions

Then turn those numbers into an array

```
// the table containing the "curve" the brightness should take
byte bright_table[] = { 30, 30, 30, 40, 50, 60, 70, 80, 90, 100,
                      110, 120, 130, 140, 150, 160, 170, 180, 190, 200,
                      210, 220, 230, 240, 250, 260, 270, 280, 290, 300,
                      200, 190, 180, 170, 160, 150, 140, 130, 120, 110,
                      100, 90, 80, 70, 60, 50, 40, 30, 30, 30 };
int max_count = 50;  // number of entries in the bright_table
```

Use any pattern of numbers you like
but they must range between 0-255

- 0 = full off
- 127 = half on
- 255 = full on

LED Brightness Functions

Once you have your table...

```
// the table containing the "curve" the brightness should take
byte bright_table[] = { 30, 30, 30, 40, 50, 60, 70, 80, 90, 100,
                      110, 120, 130, 140, 150, 160, 170, 180, 190, 200,
                      210, 220, 230, 240, 250, 260, 270, 280, 290, 300,
                      200, 190, 180, 170, 160, 150, 140, 130, 120, 110,
                      100, 90, 80, 70, 60, 50, 40, 30, 30, 30 };
int max_count = 50;  // number of entries in the bright_table
```

...the rest is just programming

1. Get a bright_table value
2. Send it out with analogWrite()
3. Advance counter into bright_table
4. Wait a bit
5. Repeat
Glowing Eyes Sketch

```cpp
int potPin = 0;
int ledPin = 10;

// the table containing the 'curve' the brightness should take
byte bright_table[] = { 30, 30, 30, 40, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 259, 240, 230, 220, 210, 200, 190, 180, 170, 160, 150, 140, 130, 120, 110, 100, 90, 80, 70, 60, 50, 40, 30, 20, 10, 0 };

int max_count = 50; // number of entries in the bright_table
int count = 0; // position within the bright_table
int val = 0; // variable for reading pin status

void setup() {
pinMode(ledPin, OUTPUT); // sets the digital pin as output
}

void loop() {
analogWrite(ledPin, bright_table[count]); // sets the LED bright
if (count > max_count) { // moves counter to next position in table
    count = 0;
} // if at end of table, back to start
val = analogRead(potPin);
val = val/4; // scale it down so it's quicker
delay(val);
}
```

Communicating with Others

- Arduino can use same USB cable for programming and to talk with computers
- Talking to other devices uses the "Serial" commands
  - `Serial.begin()` – prepare to use serial
  - `Serial.print()` – send data to computer
  - `Serial.read()` – read data from computer
Serial from Arduino to PC

- `Serial.begin(baud-rate);`
  - baud-rate is 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 57600, or 115200
  - Sets serial bit rate
- `Serial.print(arg);`
  - sends arg to the serial output – can be number or string
  - `Serial.print(arg, format);` // formats the arg
  - format can be BYTE, BIN, OCT, DEC, HEX
- `Serial.println(arg);`
  - Same, but also prints a newline to the output

```cpp
void setup() {
  Serial.begin(9600);   // init the serial port
}

void loop() {
  Serial.println("Hello World!");  // print to the screen!
  delay(500);       // Wait so you don’t print too fast
}
```

Send data to PC
Checking on Analog Inputs

```cpp
int sensorPin = 0;  // select the input pin for the potentiometer
int ledPin = 13;     // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor

void setup() {
    pinMode(ledPin, OUTPUT);  // declare the ledPin as an OUTPUT:
    Serial.begin(9600);              // Init serial communication at 9600 baud
}

void loop() {
    sensorValue = analogRead(sensorPin); // read the value from the sensor:
    Serial.print("Sensor value is: ");  // print a message
    Serial.println(sensorValue, DEC);   // print the value you got
    delay(500);                                             // wait so you don't print too much!
}
// VERY useful for getting a feel for the range of values coming in
// map(value, inLow, inHigh, outLow, outHigh);
```

Serial From PC to Arduino

- `Serial.available();`  
  returns an int that tells you how many bytes remain in the input buffer

- `Serial.read();`  
  returns the next byte waiting in the input buffer

- `Serial.flush();`  
  clear the input buffer of any remaining bytes
Serial Read Example

```c
int incomingByte = 0; // for incoming serial data
void setup() {
    Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}
void loop() {         // send data only when you receive data:
    if (Serial.available() > 0) { // read the incoming byte:
        incomingByte = Serial.read();
        // say what you got:
        Serial.print("I received: ");
        Serial.println(incomingByte, DEC);
    }
}
```

Arduino Says “Hi”

“SerialHelloWorld”

Sends “Hello world!” to your computer

Click on “Serial Monitor” button to see output

Watch TX LED compared to pin 13 LED
Telling Arduino What To Do

"SerialReadBasic"

You type “H”, LED blinks

In “Serial Monitor”, type “H”, press Send

Serial.available() tells you if data present to read

Arduino Communications

is just serial communications

- Psst, Arduino doesn’t really do USB
- It really is “serial”, like old RS-232 serial
- All microcontrollers can do serial
- Not many can do USB
- Serial is easy, USB is hard
Serial Communications

- “Serial” because data is broken down into bits, each sent one after the other down a single wire.

- The single ASCII character ‘B’ is sent as:

\[
\begin{align*}
'B' &= 0 1 0 0 0 0 1 0 \\
&= L H L L L L H L \\
&= \text{HIGH} \hspace{1cm} \text{LOW}
\end{align*}
\]

- Toggle a pin to send data, just like blinking an LED

- You could implement sending serial data with `digitalWrite()` and `delay()`

- A single data wire needed to send data. One other to receive.

Arduino & USB-to-serial

Arduino board is really two circuits

![Arduino board with USB to serial and microcontroller](image)
Arduino Mini

Arduino Mini separates the two circuits

Arduino Mini USB adapter  Arduino Mini

Arduino to Computer

USB is totally optional for Arduino
But it makes things easier
Arduino & USB

• Since Arduino is all about serial
• And not USB,
• Interfacing to things like USB flash drives, USB hard disks, USB webcams, etc. is not possible

Controlling the Computer

• Can send sensor data from Arduino to computer with Serial.print()
• There are many different variations to suite your needs:

```
int val = 123;
Serial.print(val);   // sends 3 ASCII chars "123"
Serial.print(val,DEC); // same as above
Serial.print(val,HEX); // sends 2 ASCII chars "7B"
Serial.print(val,BIN); // sends 8 ASCII chars "0b111011"
Serial.print(val,BYTE); // sends 1 byte, the verbatim value
```
Controlling the Computer

You write one program on Arduino, one on the computer

**In Arduino:** read sensor, send data as byte

```c
void loop() {
  val = analogRead(analogInput);  // read the value on analog input
  Serial.print(val/4);            // print a byte value out
  delay(500);                    // wait a bit to not overload the port
}
```

**In Processing:** read the byte, do something with it

```java
import processing.serial.*;
Serial myPort;  // The serial port

void setup() {
  String portname = ""/dev/tty.usbserial-A30088n6B";
  myPort = new Serial(this, myPort, 9600);
}

void draw() {
  while (myPort.available() > 0) {
    int inByte = myPort.read();
    println(inByte);
  }
}
```

---

Controlling the Computer

- Receiving program on the computer can be in any language that knows about serial ports
- C/C++, Perl, PHP, Java, Max/MSP, Python, Visual Basic, etc.
- Pick your favorite one, write some code for Arduino to control
ASCII codes

Standard byte codes for characters

Mysterious `val = val - '0';` statement converts the byte that represents the character to a byte of that number

For example, if the character is '3', the ASCII code is 51

The ASCII code for '0' is 48

So, 51 - 48 = 3

This converts the character '3' into the number 3

ASCII code: 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 0 (ES). The DEL code can be generated by the CTRL + BS/F key.
Moving on… Servos

- Servo motors are small DC motors that have a range of motion of 0-180°
- Internal feedback and gearing to make it work
- Easy three-wire interface
- Position is controlled by PWM signals
PWM

Output voltage is averaged from on vs. off time

\[ \text{output\_voltage} = \left( \frac{\text{on\_time}}{\text{off\_time}} \right) \times \text{max\_voltage} \]

- 3.75 Volts
  - 5 volts
  - 0 volts
  - 75% 25% 75% 25% 75% 25%

- 2.5 Volts
  - 5 volts
  - 0 volts
  - 50% 50% 50% 50% 50%

- 1.0 Volts
  - 5 volts
  - 0 volts
  - 20% 80% 20% 80% 20% 80%

PWM

- Used everywhere
  - Lamp dimmers, motor speed control, power supplies, noise making

- Three characteristics of PWM signals
  - Pulse width range (min/max)
  - Pulse period (= 1/pulses per second)
  - Voltage levels (0-5V, for instance)
Servomotors

- Can be positioned from 0-180° (usually)
- Internal feedback circuitry & gearing takes care of the hard stuff
- Easy three-wire PWM 5V interface

Servos are Awesome

- DC motor
- High-torque gearing
- Potentiometer to read position
- Feedback circuitry to read pot and control motor
- All built in, you just feed it a PWM signal
Our servos are: weight: 9g,
speed: 0.12s/60deg at 4.8v,
torque (@4.8v) 17.5oz/in (1kg/cm)
voltage range: 3.0 – 7.2v
Servo Mounts & Linkages

Lots of ways to mount a servo

And turn its rotational motion into other types of motion

---

Servo Control

- PWM freq is 50 Hz (i.e. every 20 milliseconds)
- Pulse width ranges from 1 to 2 milliseconds
  - 1 millisecond = full anti-clockwise position
  - 2 milliseconds = full clockwise position
Servo Movement

0 degrees  90 degrees  180 degrees
1000 microsecs  1500 microsecs  2000 microsecs

In practice, pulse range can range from 500 to 2500 microsecs

Servo and Arduino
First, add some jumper wires to the servo connector

Gnd
Power
PWM control
Servo Example Program

```cpp
#include <Servo.h>  // include the built-in servo library
Servo myservo;  // create a servo object to control the servo (one per servo)
int pos = 0;   // variable to store the servo position

void setup() {
  myservo.attach(9);  // attach servo control to pin 9
}

void loop() {
  for (pos = 0; pos < 180; pos++) {     // go from 0 to 180 degrees
    myservo.write(pos);  // move the servo
    delay(15); // give it time to get there
  }
  for (pos = 180; pos>=1; pos--) {    // wave backwards
    myservo.write(pos);  // move the servo
    delay(15); // give it time to get there
  }
}
```

Servo Functions

- **Servo** is a class
  - **Servo myservo;**  // creates an instance of that class
- **myservo.attach(pin);**
  - attach to an output pin (doesn’t need to be PWM pin!)
  - Servo library can control up to 12 servos on our boards
  - but a side effect is that it disables the PWM on pins 9 and 10
- **myservo.write(pos);**
  - moves servo – pos ranges from 0-180
- **myservo.read();**
  - returns the current position of the servo (0-180)
Moving on…

- Write a program to control the position of the servo from a pot, or from a photocell
  - remember pot `analogRead();` values are from 0-1023
  - measure the range of values coming out of the photocell first?
  - use `Serial.print(val);` for example
  - use `map(val, in1, in2, 0, 180);` to map in1-in2 values to 0-180
  - Can also use `constrain(val, 0, 180);`

Side Note - Power

- Servos can consume a bit of power
  - We need to make sure that we don’t draw so much power out of the Arduino that it fizzes
  - If you drive more than a couple servos, you probably should put the servo power pins on a separate power supply from the Arduino
  - Use a wall-wart 5v DC supply, for example
Robo Cat Toy Idea

Tape on a pipe cleaner, and using random behavior similar to the “Candlelight” sketch, make a randomly moving cat toy.

Servo/Light Assignment

- Use a photocell on the input
  - put in series with 10k ohm resistor
- use a servo on the output
  - connect to a PWM pin
- make the servo do something in response to the amount of light falling on the photocell
Summary – Whew!

- **LEDs** – use current limiting resistors (remember color code!)
  - drive from `digitalWrite(pin,val);` for on/off
  - drive from `analogWrite(pin,val);` for PWM dimming (values from 0-255)

- **Buttons** – current limiting resistors again
  - active-high or active low (pullup or pulldown)
  - read with `digitalRead(pin);`

- **Potentiometers (pots)** – voltage dividers with a knob
  - use with `analogRead(pin);` for values from 0-1023

Summary – Whew!

- **Photocells** – variable resistors
  - use with current-limiting resistors (to make voltage divider)

- **Serial communications** – read a byte, or write a value
  - communicate to the Arduino environment, or your own program

- **Servos** – use Servo library to control motion
  - might need external power supply
  - range of motion 0-180°

- Also `setup()` and `loop()` functions, and various C programming ideas
More Later…

- **DC Motors**
  - use transistors as switches for larger current loads

- **Stepper motors**
  - Sort of like servos, but with continuous range of motion
  - Can also be more powerful

- **I2C serial bus**
  - Various LED driver chips
  - Other serially-controlled devices

- **Piezo buzzers**
  - Make some noise!
  - But you can also use them as input devices to sense movement

- **IR motion sensors**
  - Simple motion and also distance sensors

- **Accelerometers**
  - Wii nunchucks, for example

- **Others?**