Spooky Projects

Introduction to Microcontrollers with Arduino

Class 3

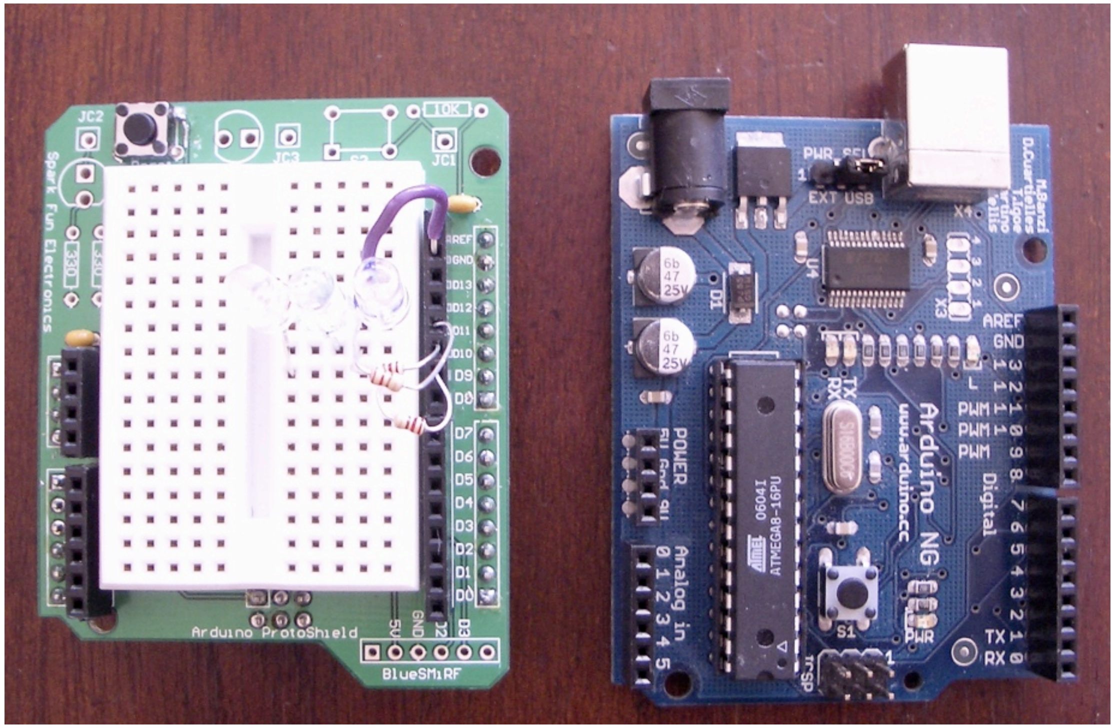
21 Oct 2006 - machineproject - Tod E. Kurt

What’s For Today

- Controlling Arduino from a computer
- Controlling a computer from Arduino
- Servomotors
- R,G,B LED color mixing

Remove ProtoShield

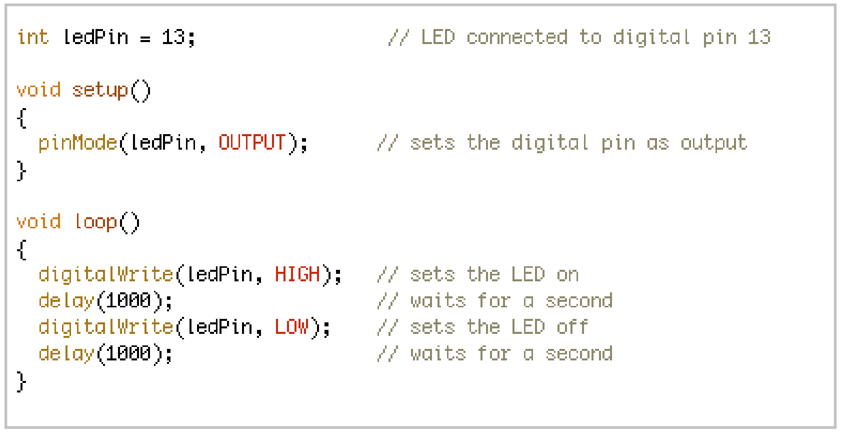
First half of class, we don’t need it
And we want to observe the Arduino board



“Shields down, cap’n!”

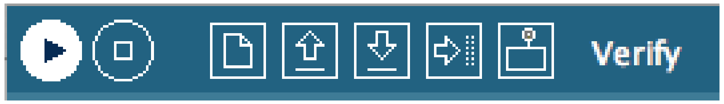
Recap: Programming

Edit

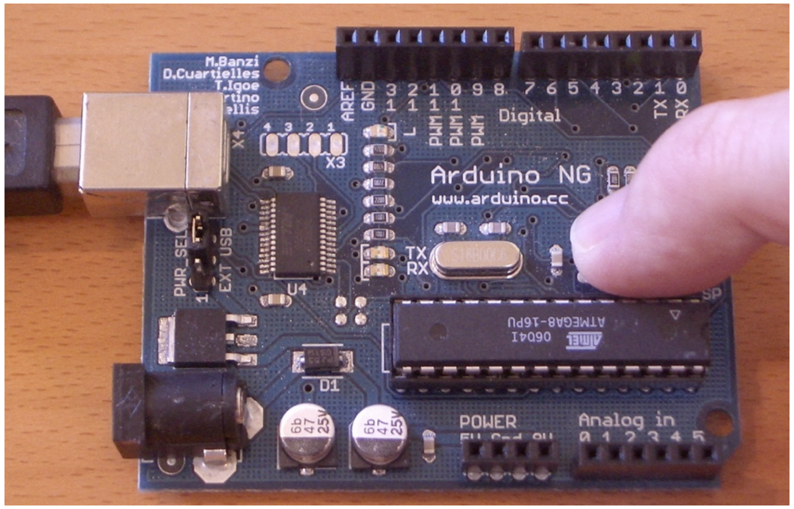


```
int ledPin = 13;                 // LED connected to digital pin 13

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

void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

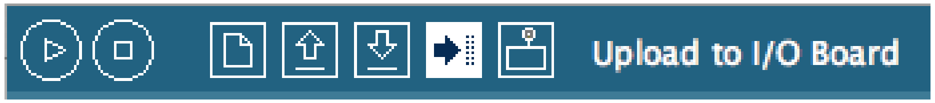
Compile



Reset



Upload



Remember: always start from a known working system

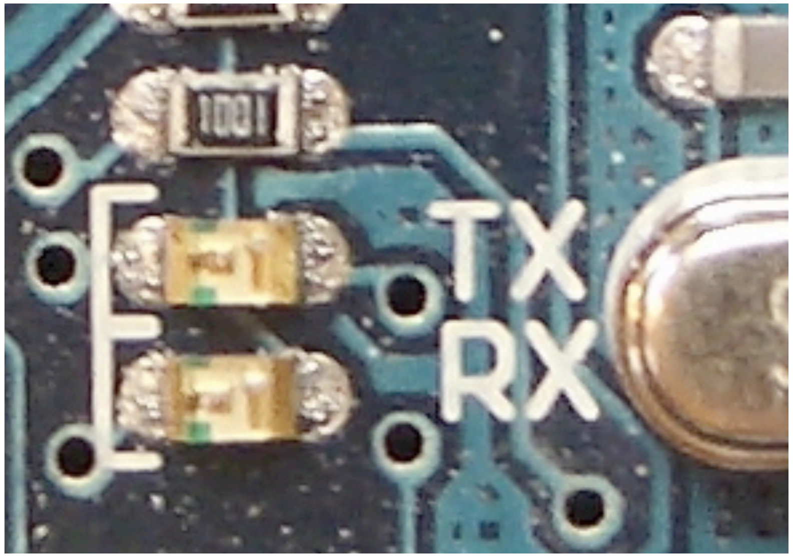
Communicating with Others

- 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

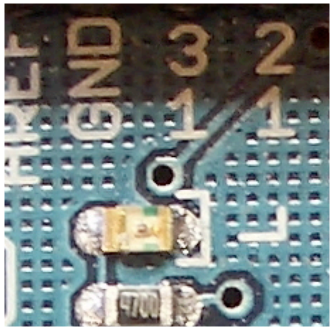
Can talk to not just computers.
Most things more complex than simple sensors/actuators speak serial.

Watch the TX/RX LEDS

- TX – sending to PC
- RX – receiving from PC
- Used when programming or communicating



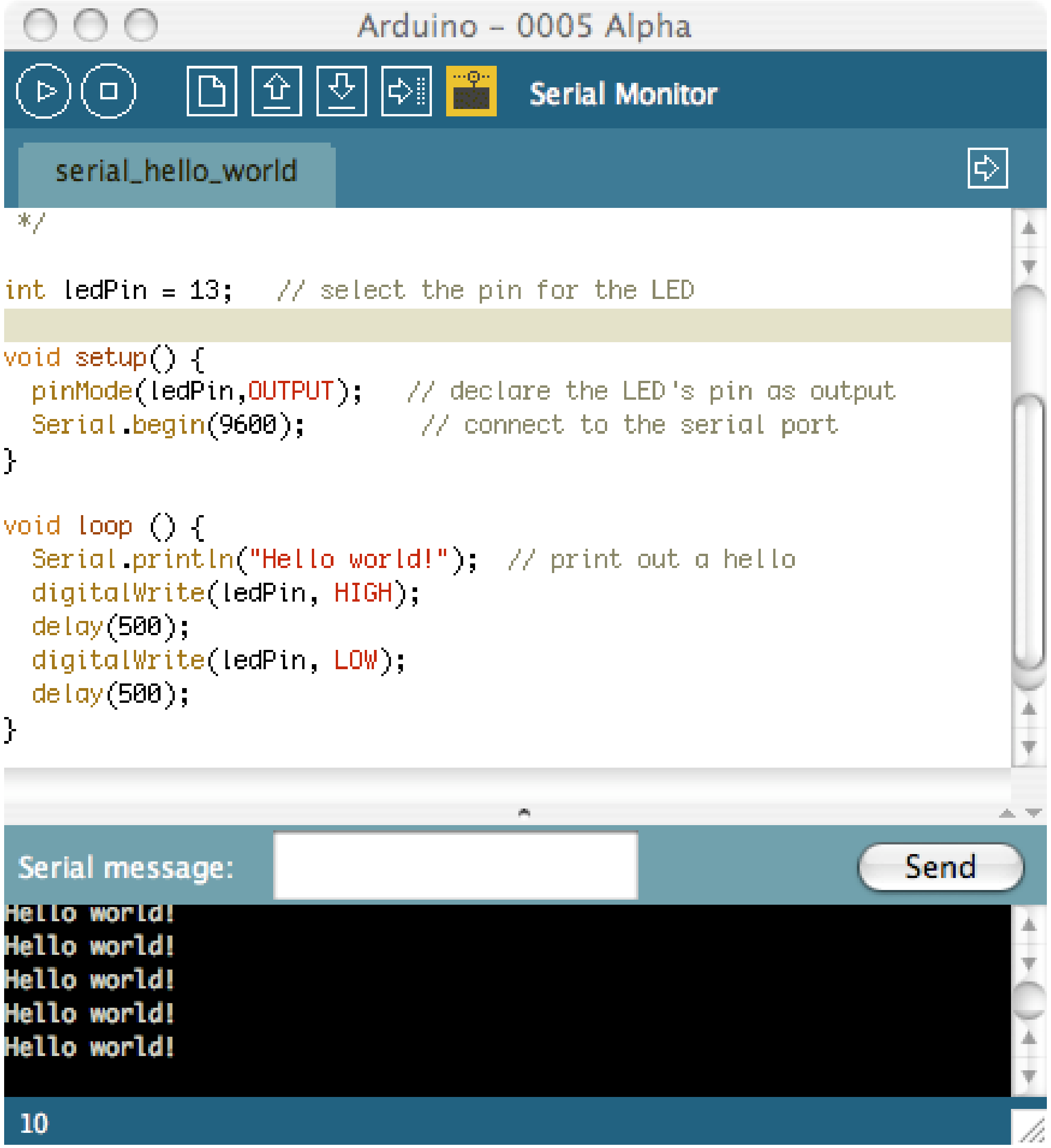
(and keep an eye on that pesky pin13 LED too)



Arduino Says “Hi”

“serial_hello_world”

- Send “Hello world!” to your computer
  (and blink LED)
- Click on “Serial Monitor” to see output
- Watch TX LED compared to pin13 LED



```
 */

int ledPin = 13;   // select the pin for the LED

void setup() {
  pinMode(ledPin,OUTPUT);   // declare the LED's pin as output
  Serial.begin(9600);        // connect to the serial port
}

void loop () {
  Serial.println("Hello world!");  // print out a hello
  digitalWrite(ledPin, HIGH);
  delay(500);
  digitalWrite(ledPin, LOW);
  delay(500);
}
```

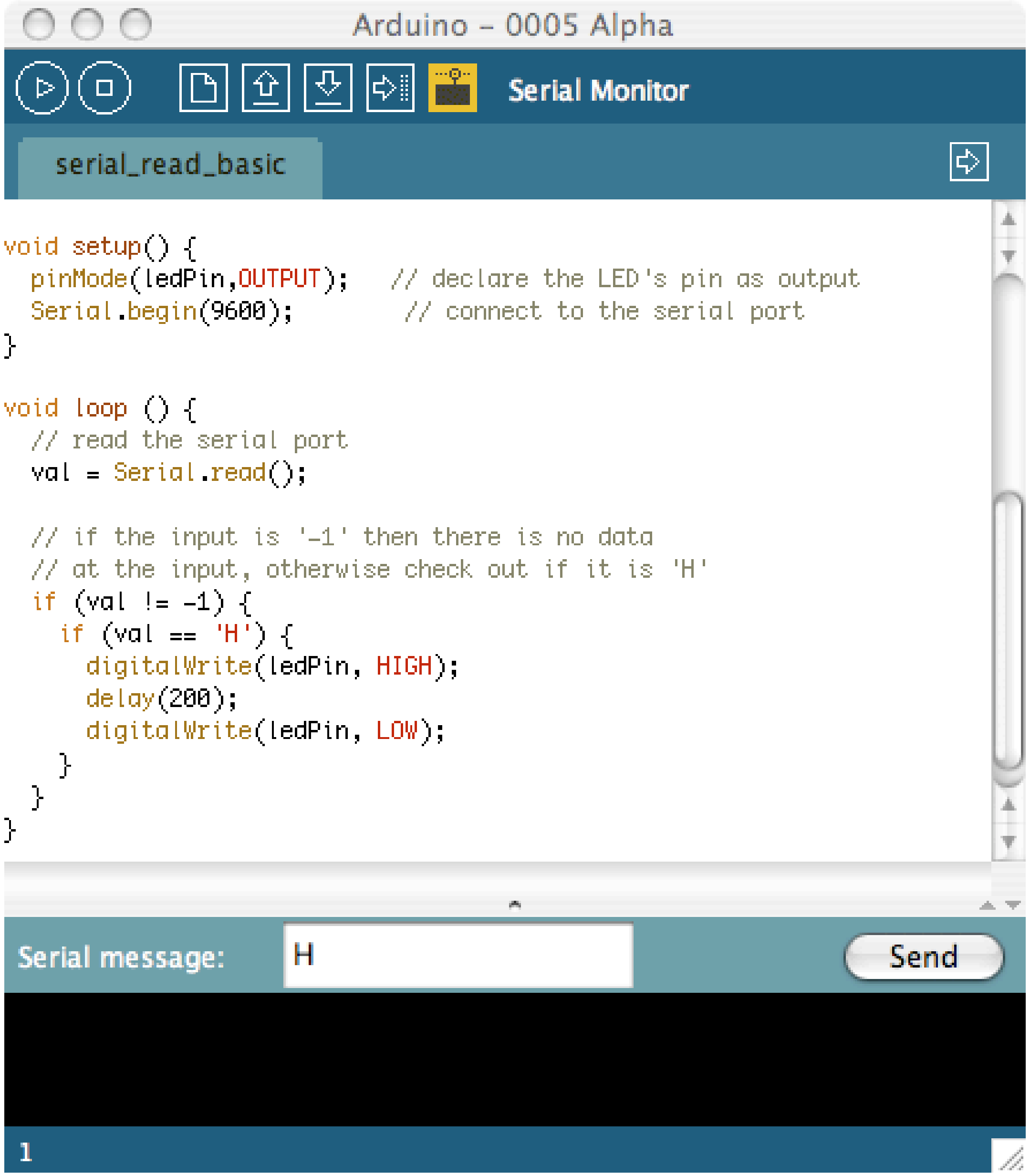
```
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
```

This sketch is located in the handout, but it’s pretty short.
Use on-board pin 13 LED, no need to wire anything up.

Telling Arduino What To Do

"serial_read_basic"

- You type "H"
  – LED blinks
- In "Serial Monitor" type "H", press Send
- Watch pin 13 LED



```
void setup() {
  pinMode(ledPin,OUTPUT);   // declare the LED's pin as output
  Serial.begin(9600);        // connect to the serial port
}

void loop () {
  // read the serial port
  val = Serial.read();

  // if the input is '-1' then there is no data
  // at the input, otherwise check out if it is 'H'
  if (val != -1) {
    if (val == 'H') {
      digitalWrite(ledPin, HIGH);
      delay(200);
      digitalWrite(ledPin, LOW);
    }
  }
}
```

This sketch is in "Examples/serial_comm/serial_read_basic".
Notice how you might not always read something, thus the "-1" check.
Can modify it to print "hello world" after it receives something, but before it checks for 'H'.
This way you can verify it's actually receiving something.

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 terminal from the olde days

Serial Communications

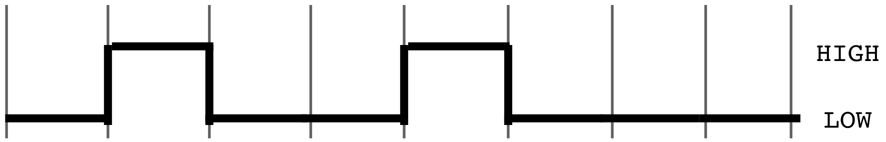
- “Serial” because data is broken down into bits, each sent one-by-one on a single wire:

'H'

= 0 1 0 0 1 0 0 0

= L H L L H L L L

=

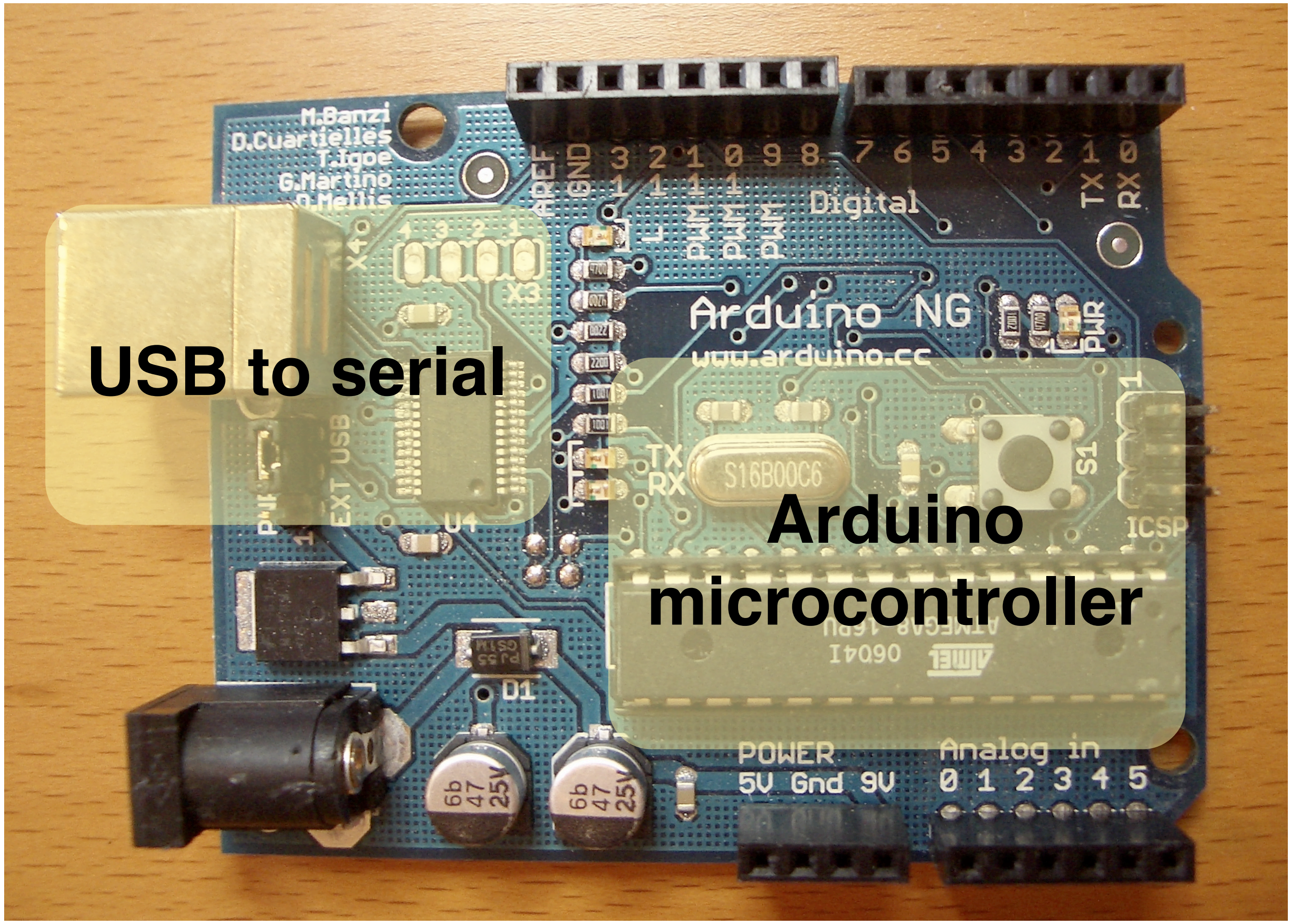


- Toggle a pin to send data, just like blinking an LED
- Only a single data wire is needed to send data. One other to receive.

Note, a single data wire. You still need a ground wire.

Arduino & USB-to-serial

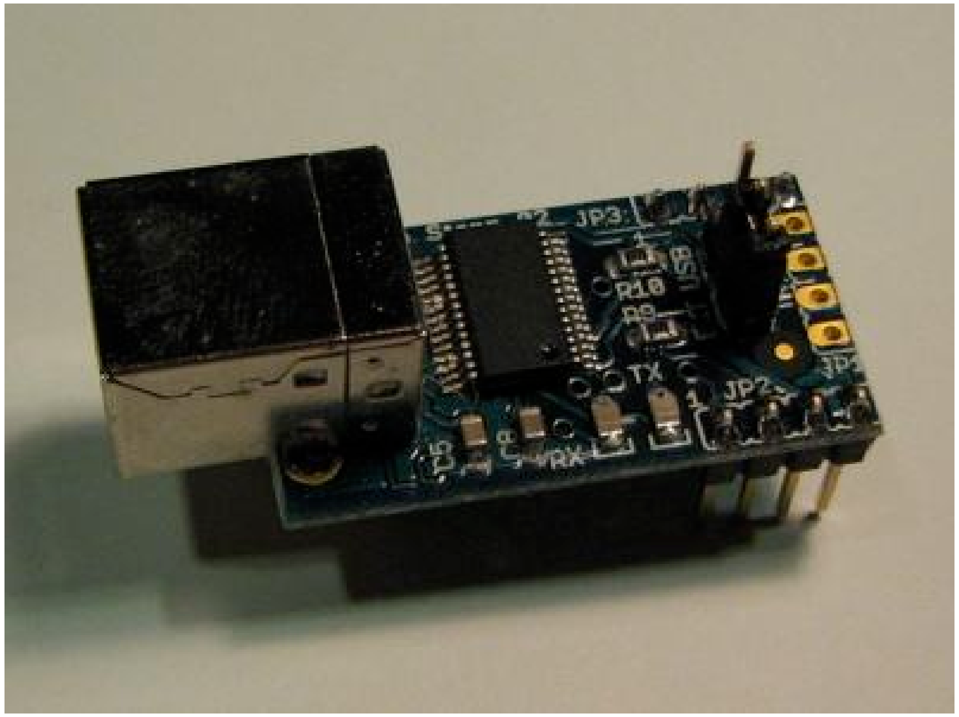
Arduino board is really two circuits



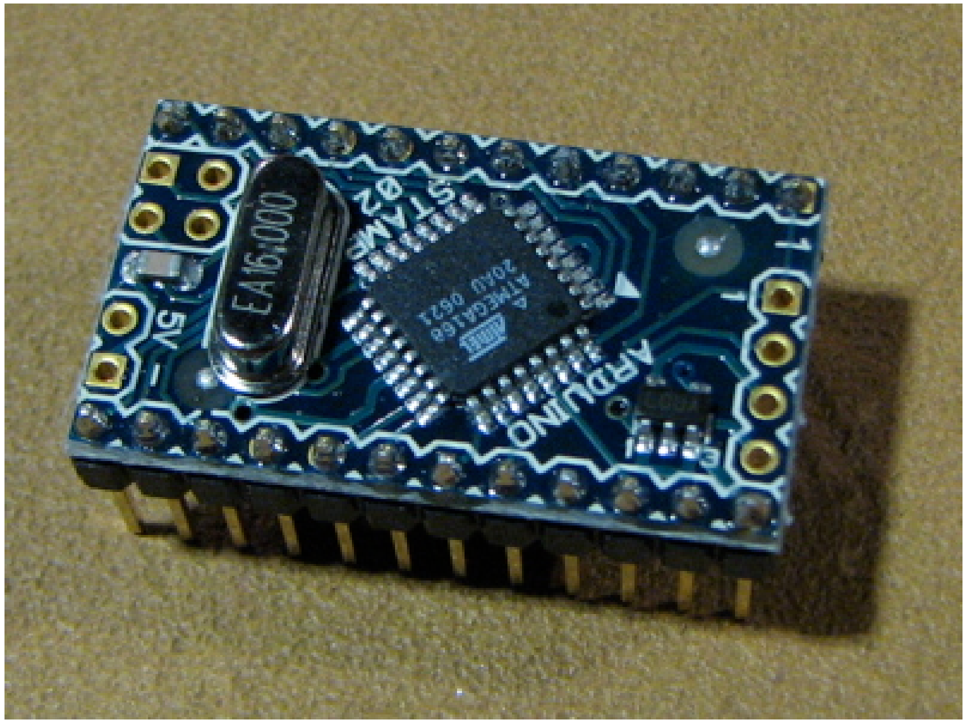
Original Arduino boards were RS-232 serial, not USB.

New Arduino Mini

Arduino Mini separates the two circuits



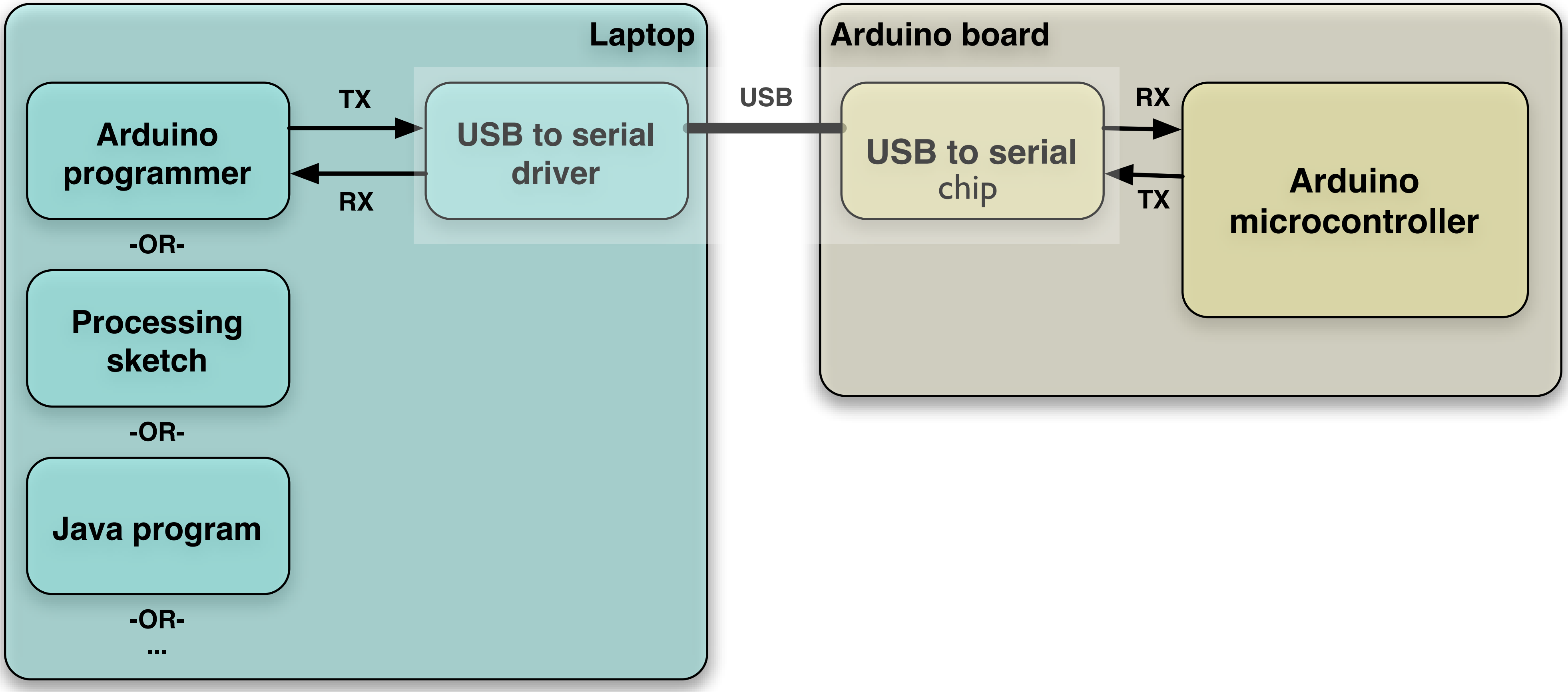
Arduino Mini USB adapter



Arduino Mini

aka. “Arduino Stamp”
If you don’t talk with a computer, the USB-to-serial functionality is superfluous.

Arduino to Computer



USB is totally optional for Arduino
But it makes things easier

Original Arduino boards were RS-232 serial, not USB.

Arduino & USB

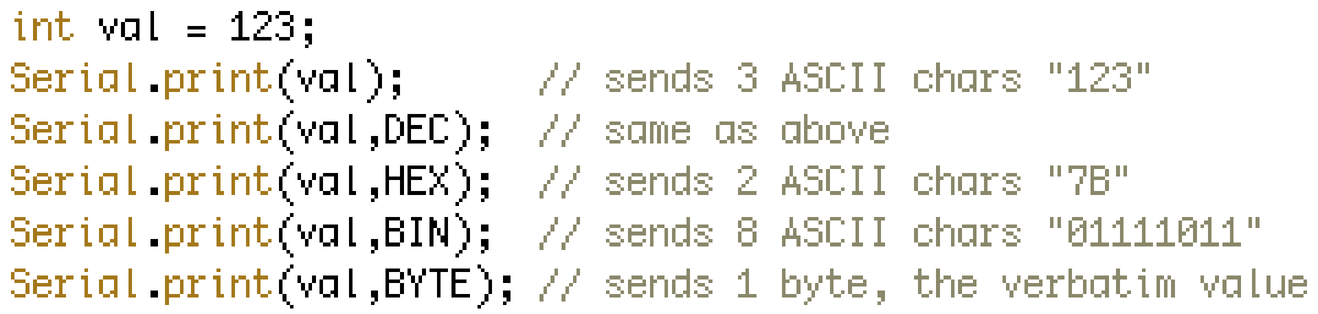
- Because Arduino is all about serial,
- And not USB,
- Interfacing to things like USB flash drives, USB hard disks, USB webcams, etc. is *not* possible

Also, USB is a host/peripheral protocol. Being a USB “host” means needing a lot of processing power and software, not something for a tiny 8kB microcontroller.

It can be a peripheral. In fact, there is an open project called “AVR-USB” that allows AVR chips like used in Arduino to be proper USB peripherals. See: http://www.obdev.at/products/avrusb/

Controlling the Computer

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

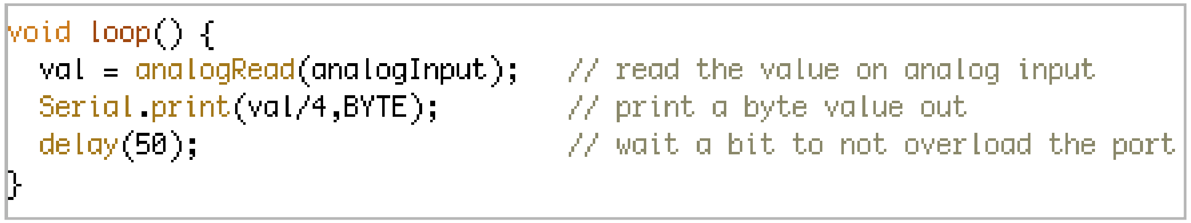


```
int val = 123;
Serial.print(val);      // sends 3 ASCII chars "123"
Serial.print(val,DEC);  // same as above
Serial.print(val,HEX);  // sends 2 ASCII chars "7B"
Serial.print(val,BIN);  // sends 8 ASCII chars "01111011"
Serial.print(val,BYTE); // sends 1 byte, the verbatim value
```

Controlling the Computer

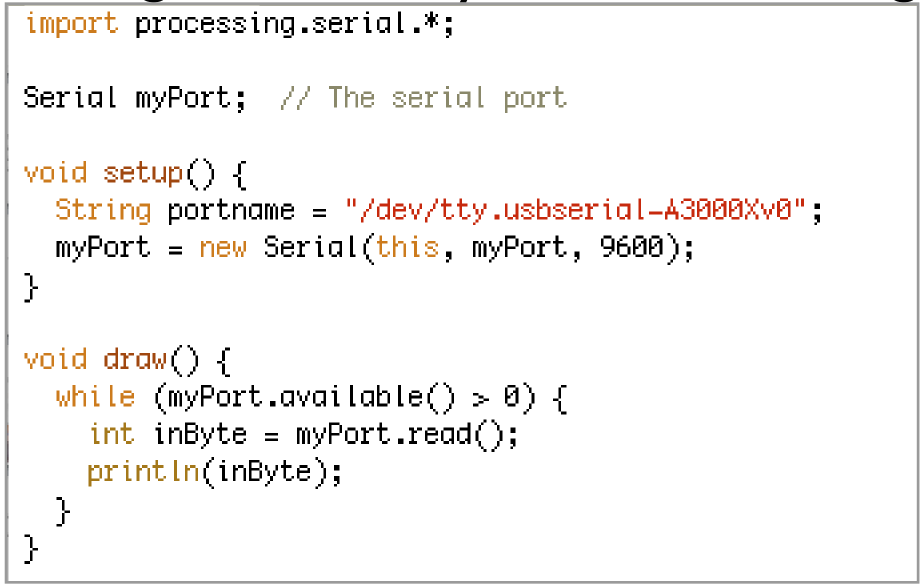
You write one program on Arduino, one on the computer

In Arduino: read sensor, send data as byte



```
void loop() {
  val = analogRead(analogInput);   // read the value on analog input
  Serial.print(val/4,BYTE);        // print a byte value out
  delay(50);                       // wait a bit to not overload the port
}
```

In Processing: read the byte, do something with it



```
import processing.serial.*;

Serial myPort;  // The serial port

void setup() {
  String portname = "/dev/tty.usbserial-A3000Xv0";
  myPort = new Serial(this, myPort, 9600);
}

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

But writing Processing programs is for another time

Controlling the Computer

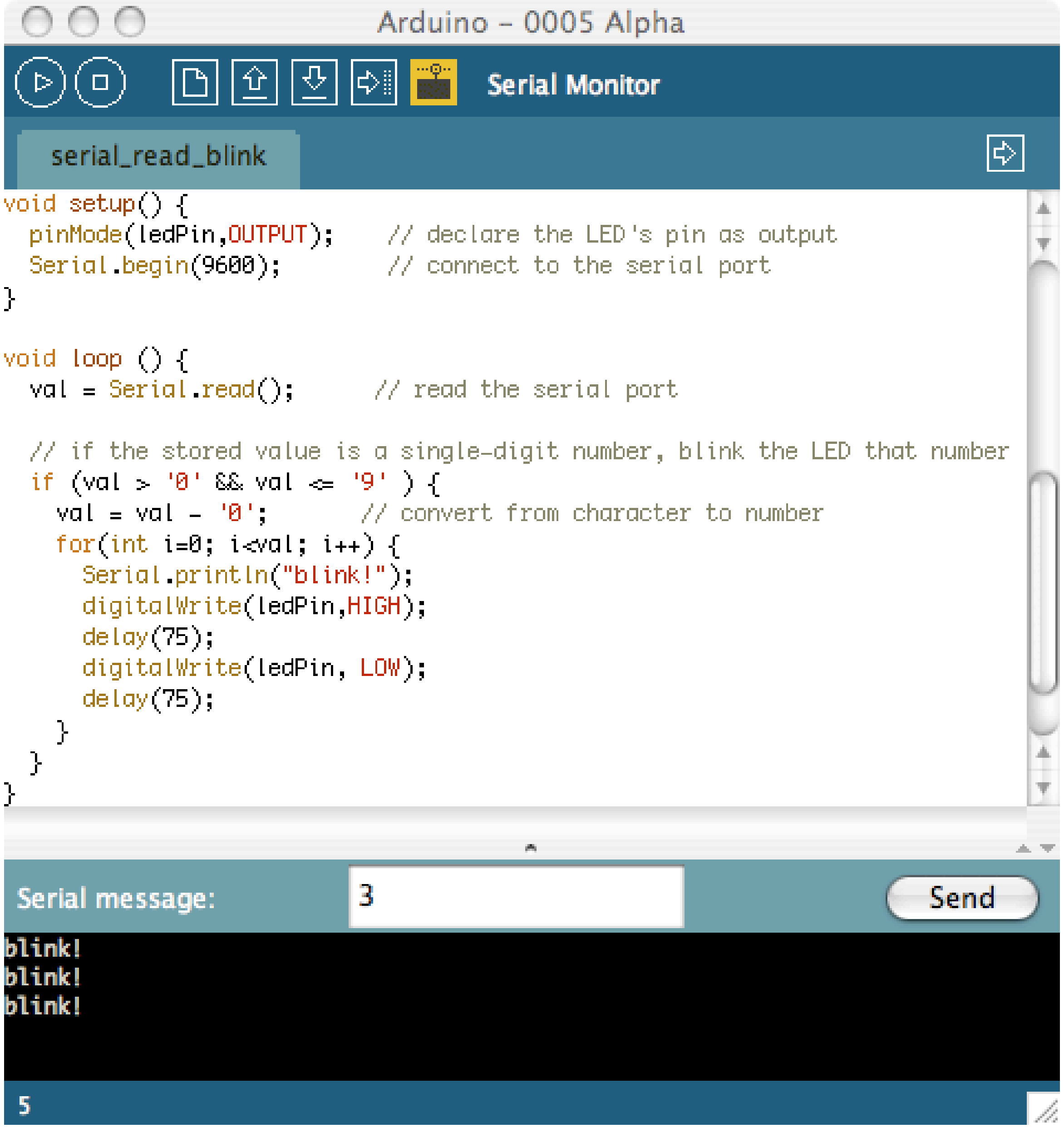
- Receiving program on the computer can be in any language that knows about serial ports
  - C/C++, Perl, PHP, Java, Max/MSP, Python, Visual Basic, etc.
- Pick your favorite one, write some code for Arduino to control

If interested, I can give details on just about every language above.

Another Example

“serial_read_blink”

- Type in a number 1-9 and LED blinks that number
- Converts number typed into usable number



```
void setup() {
  pinMode(ledPin,OUTPUT);     // declare the LED's pin as output
  Serial.begin(9600);         // connect to the serial port
}

void loop () {
  val = Serial.read();      // read the serial port

  // if the stored value is a single-digit number, blink the LED that number
  if (val > '0' && val <= '9' ) {
    val = val - '0';          // convert from character to number
    for(int i=0; i<val; i++) {
      Serial.println("blink!");
      digitalWrite(ledPin,HIGH);
      delay(75);
      digitalWrite(ledPin, LOW);
      delay(75);
    }
  }
}
```

Serial message: 3

```
blink!
blink!
blink!
```

5

This sketch is also in the handout

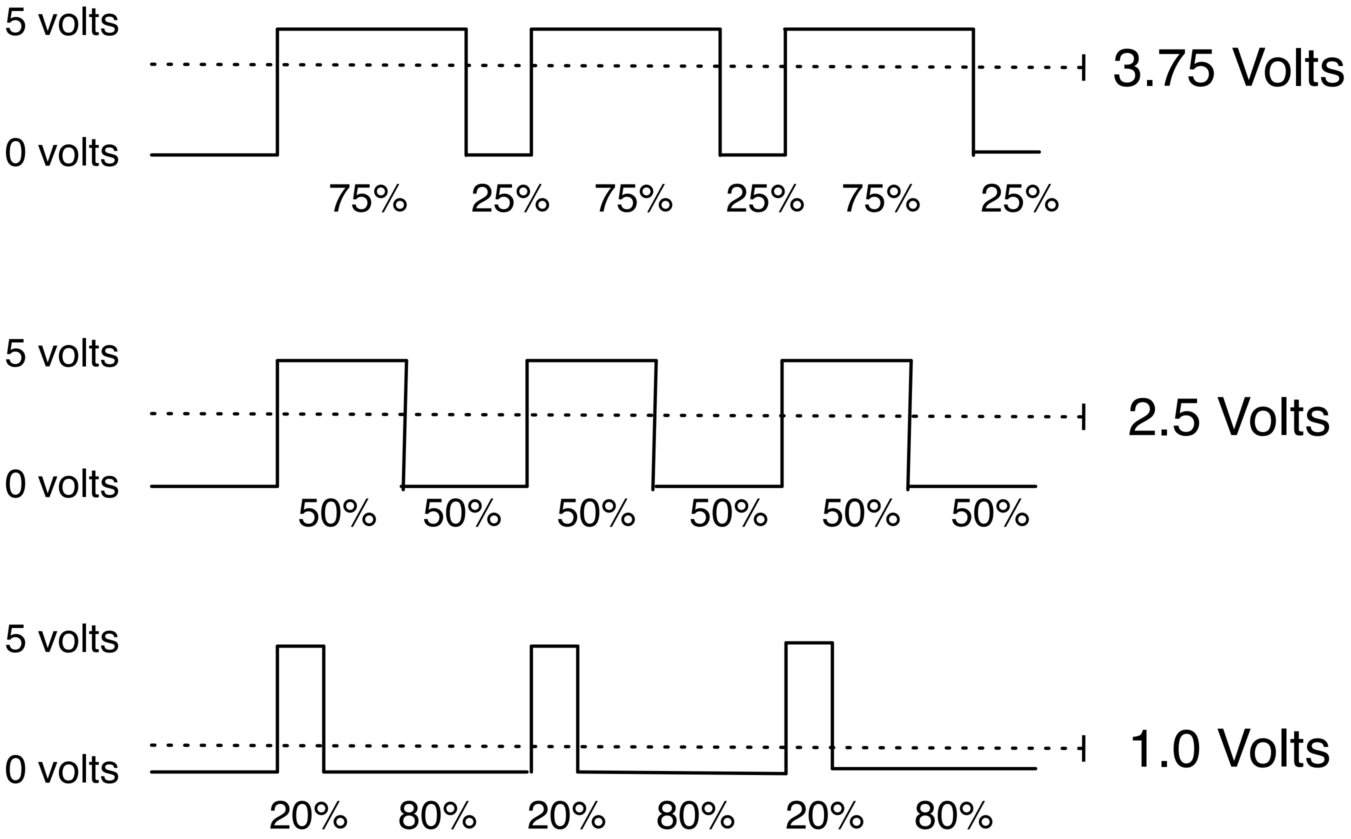
Pulse Width Modulation

- More commonly called "PWM"
- Computers can't output analog voltages
  - Only digital voltages (0 volts or 5 volts)
- But you can fake it
  - if you average a digital signal flipping between two voltages.
- For example...

PWM

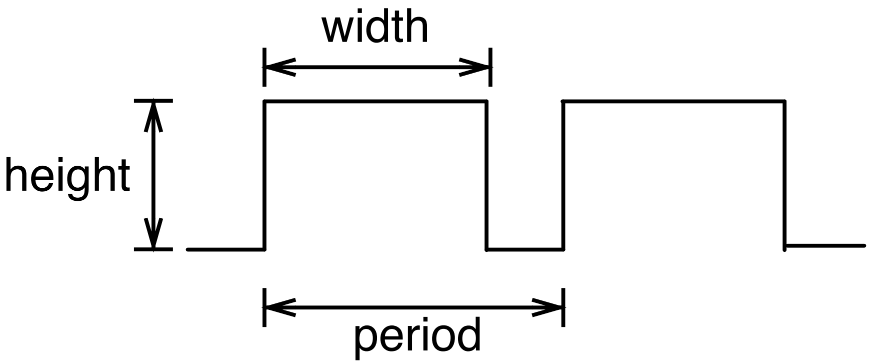
Output voltage is averaged from on *vs*. off time

output_voltage = (on_time / off_time) * max_voltage



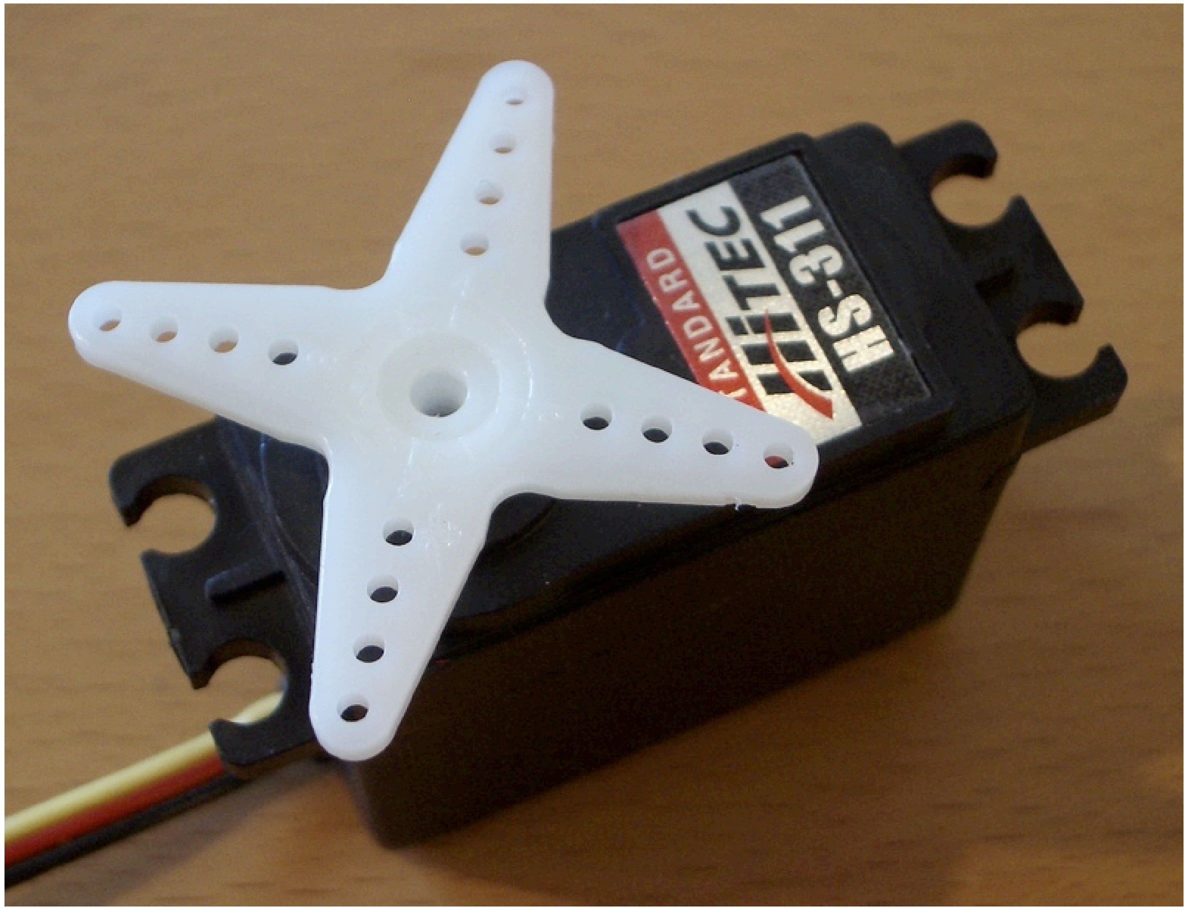
PWM

- 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°
- Internal feedback circuitry & gearing takes care of the hard stuff
- Easy three-wire PWM 5V interface



More specifically, these are R/C hobby servos used by remote control enthusiasts
In general, "servomotor" is a motor with an inherent feedback mechanism that allows you to send position commands to it without requiring you to do the position reading.

Servos, good for what?

- Roboticists, movie effects people, and puppeteers use them extensively
- Any time you need controlled, repeatable motion
- Can turn rotation into linear movement with clever mechanical levers

Even clothes use servos now: http://www.technologyreview.com/read_article.aspx?id=17639&ch=infotech

Servos

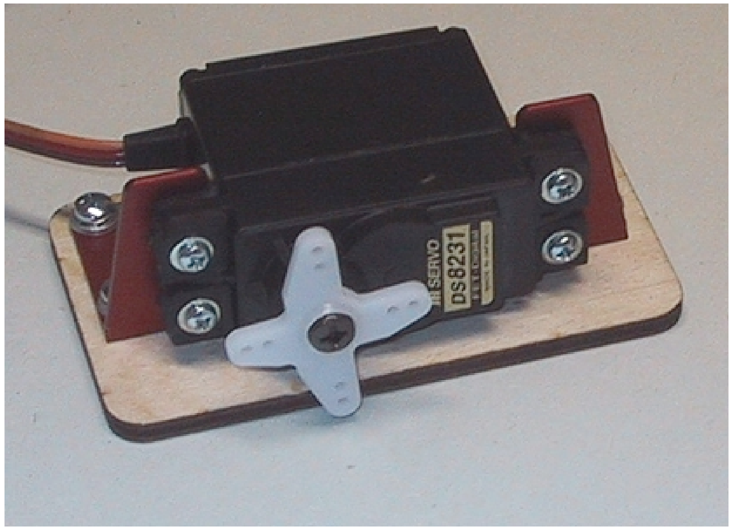
- Come in all sizes
  - from super-tiny
  - to drive-your-car
- But all have the same 3-wire interface

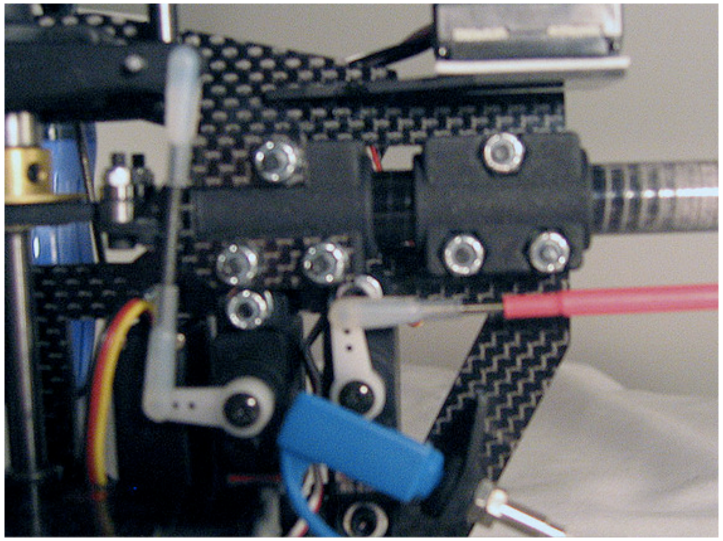
9g

157g

http://rctoys.com/
http://hobbypeople.net/

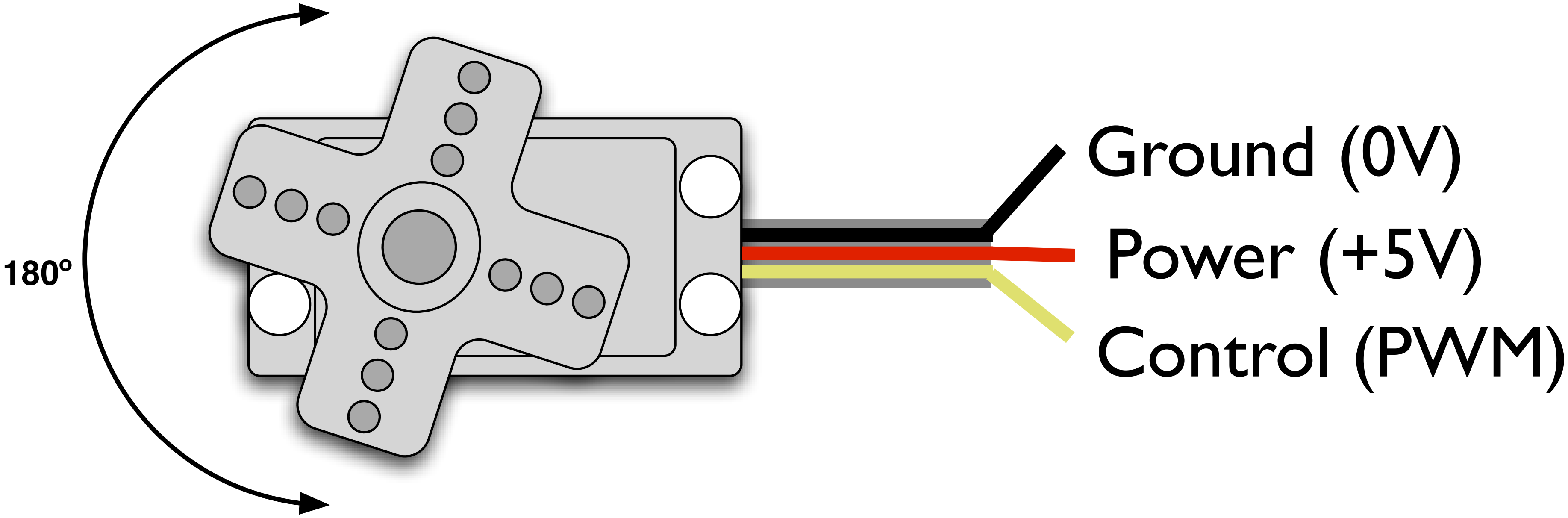
Servo Mounts & Linkages





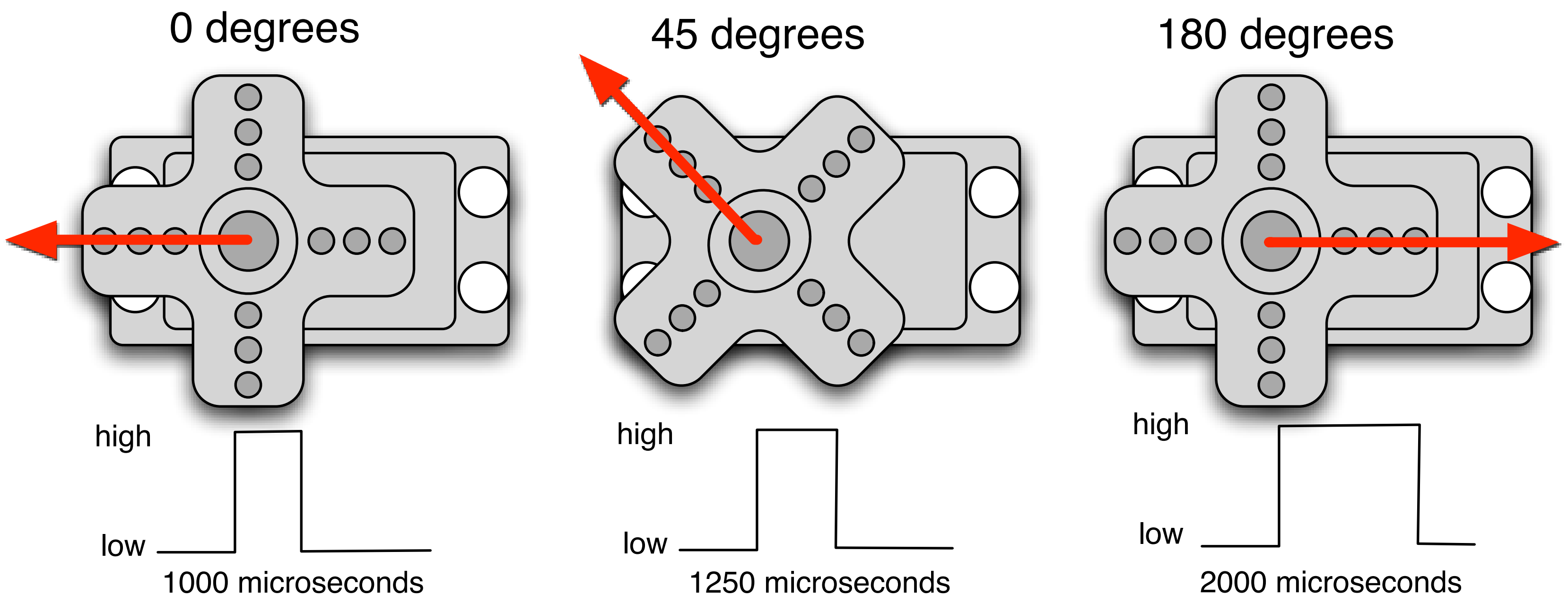
mounting bracket: http://www.sierragiant.com/prod28.html
sdfsdf

Servos



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

Servo Movement



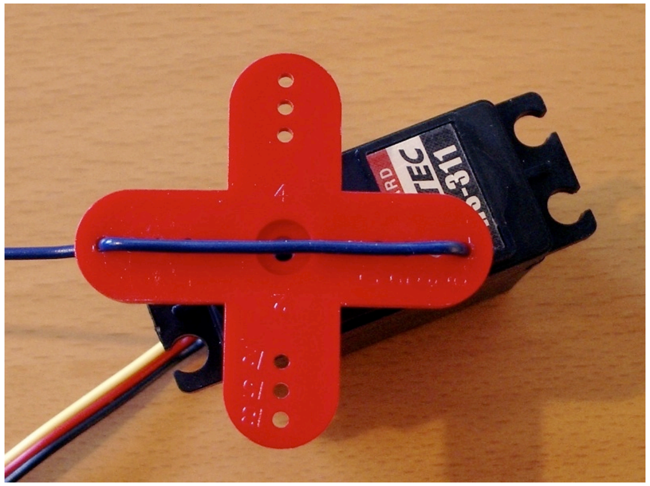
- To position, send a pulse train from 1 to 2 ms
- To hold a position, pulses must repeat
- Takes time to rotate, so pulse too fast & it won't move

1 millisecond = 1000 microsecond

See http://www.societyofrobots.com/actuators_servos.shtml

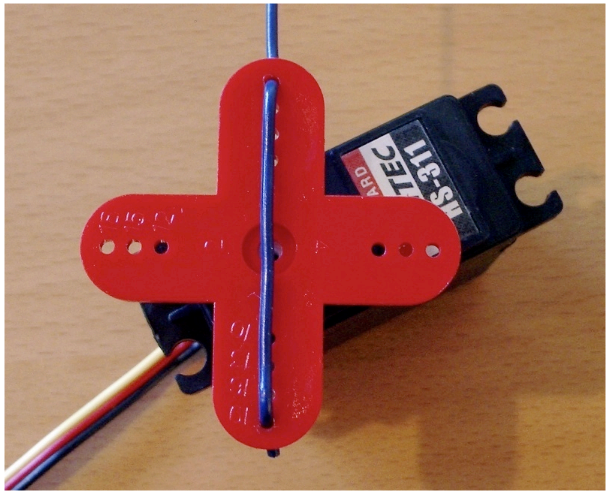
Servo Movement

0 degrees



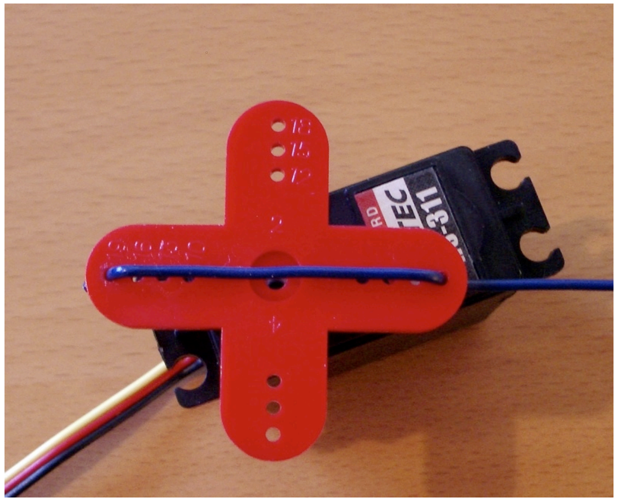
1000 microsecs

90 degrees



1500 microsecs

180 degrees



2000 microsecs

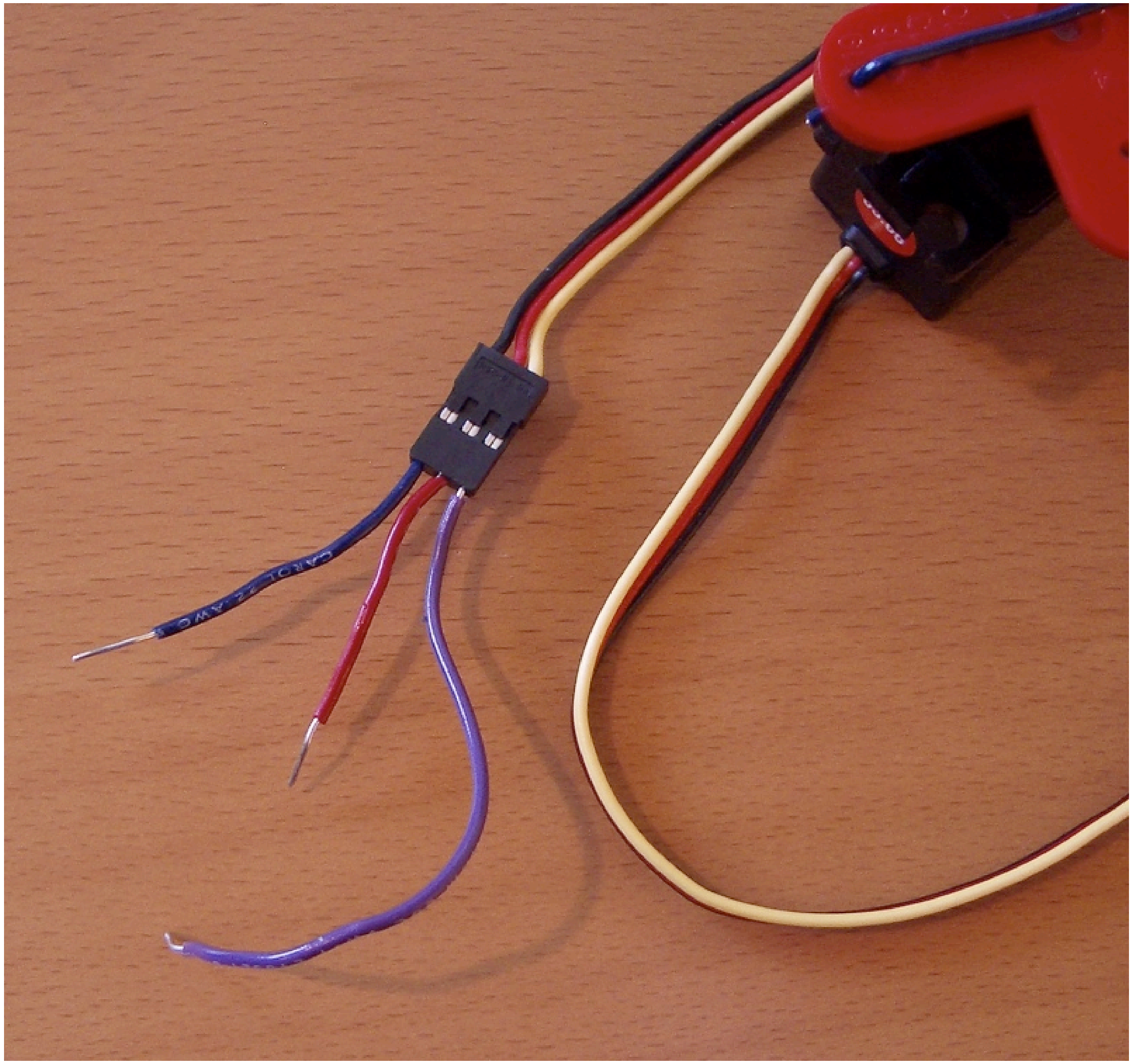
In practice, pulse range can be 500 to 2500 microsecs

(and go ahead and add a wire marker to your servo like the above)

Put the red “arm” on your servo. Needs a philips screwdriver.
Many commercial servo drivers have a calibration setting to deal with servo variability

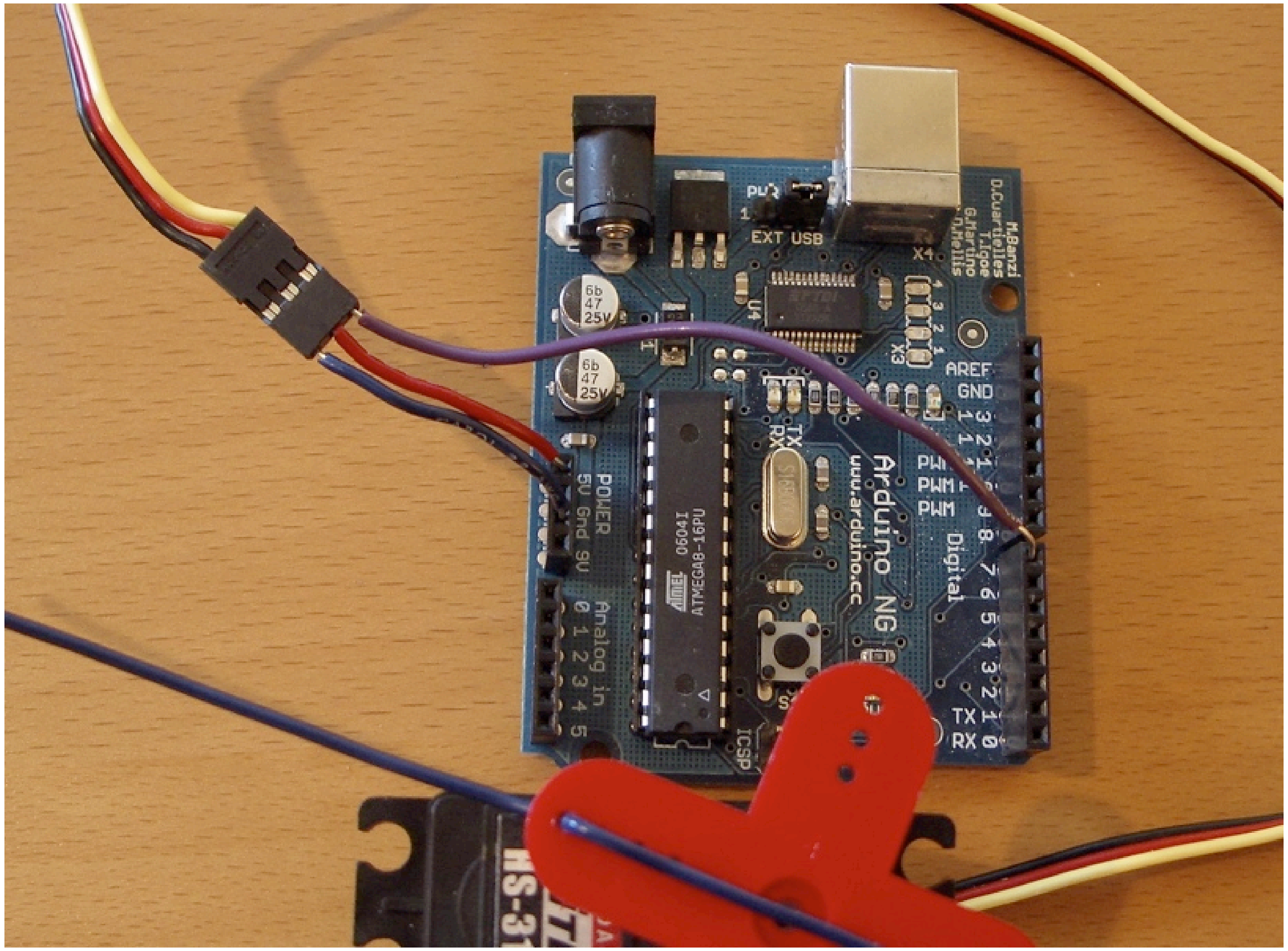
Servo and Arduino

First, add some jumper wires to the servo connector



Servo and Arduino

Plug power lines in,
Plug signal to digital pin 7

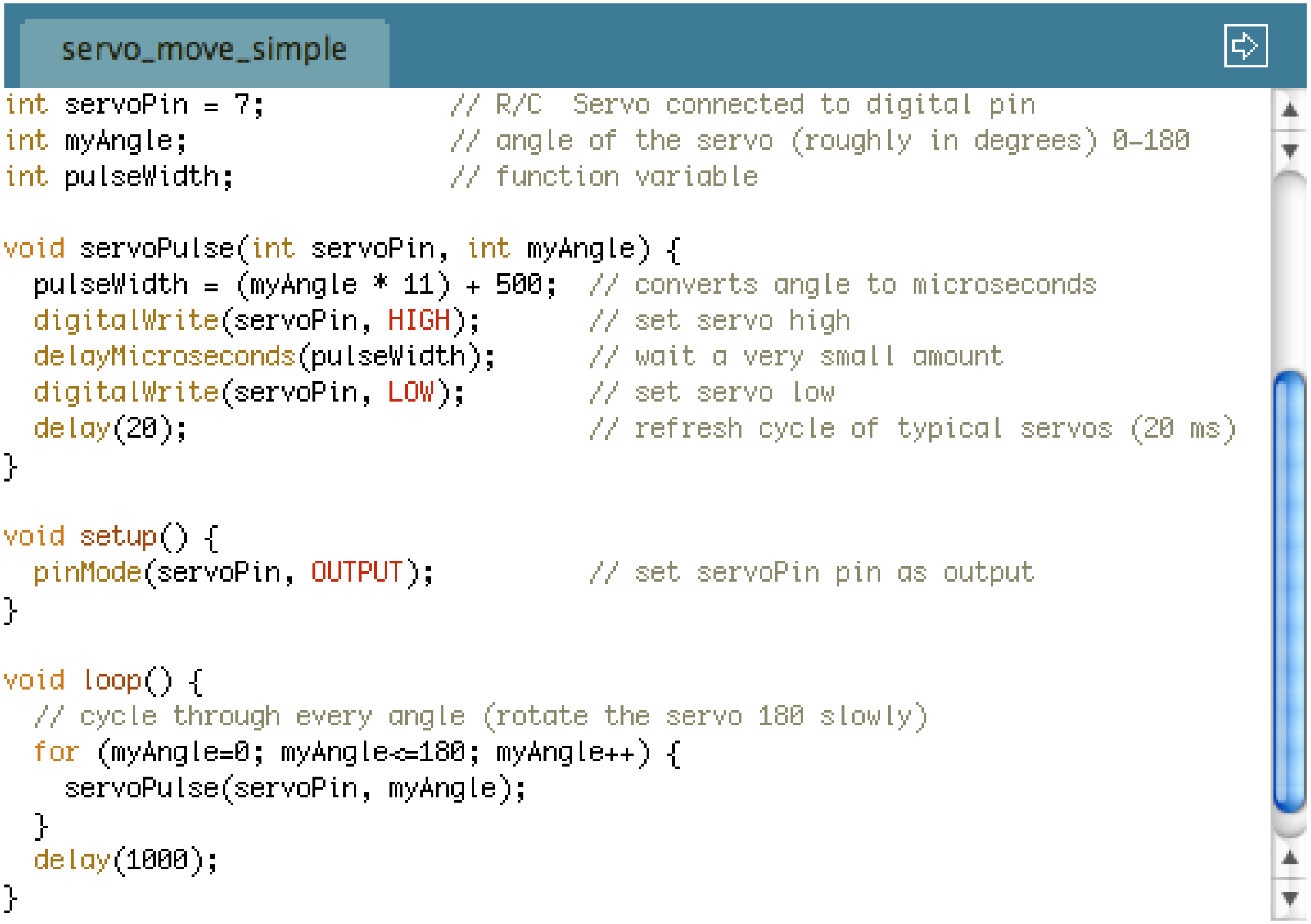


Moving a Servo

Move the servo across its full range of motion

“servo_move_simple”

- Uses `delayMicroseconds()` for pulse width
- Uses `delay()` for pulse frequency



```
int servoPin = 7;            // R/C  Servo connected to digital pin
int myAngle;                 // angle of the servo (roughly in degrees) 0-180
int pulseWidth;              // function variable

void servoPulse(int servoPin, int myAngle) {
  pulseWidth = (myAngle * 11) + 500;  // converts angle to microseconds
  digitalWrite(servoPin, HIGH);       // set servo high
  delayMicroseconds(pulseWidth);      // wait a very small amount
  digitalWrite(servoPin, LOW);        // set servo low
  delay(20);                          // refresh cycle of typical servos (20 ms)
}

void setup() {
  pinMode(servoPin, OUTPUT);          // set servoPin pin as output
}

void loop() {
  // cycle through every angle (rotate the servo 180 slowly)
  for (myAngle=0; myAngle<=180; myAngle++) {
    servoPulse(servoPin, myAngle);
  }
  delay(1000);
}
```

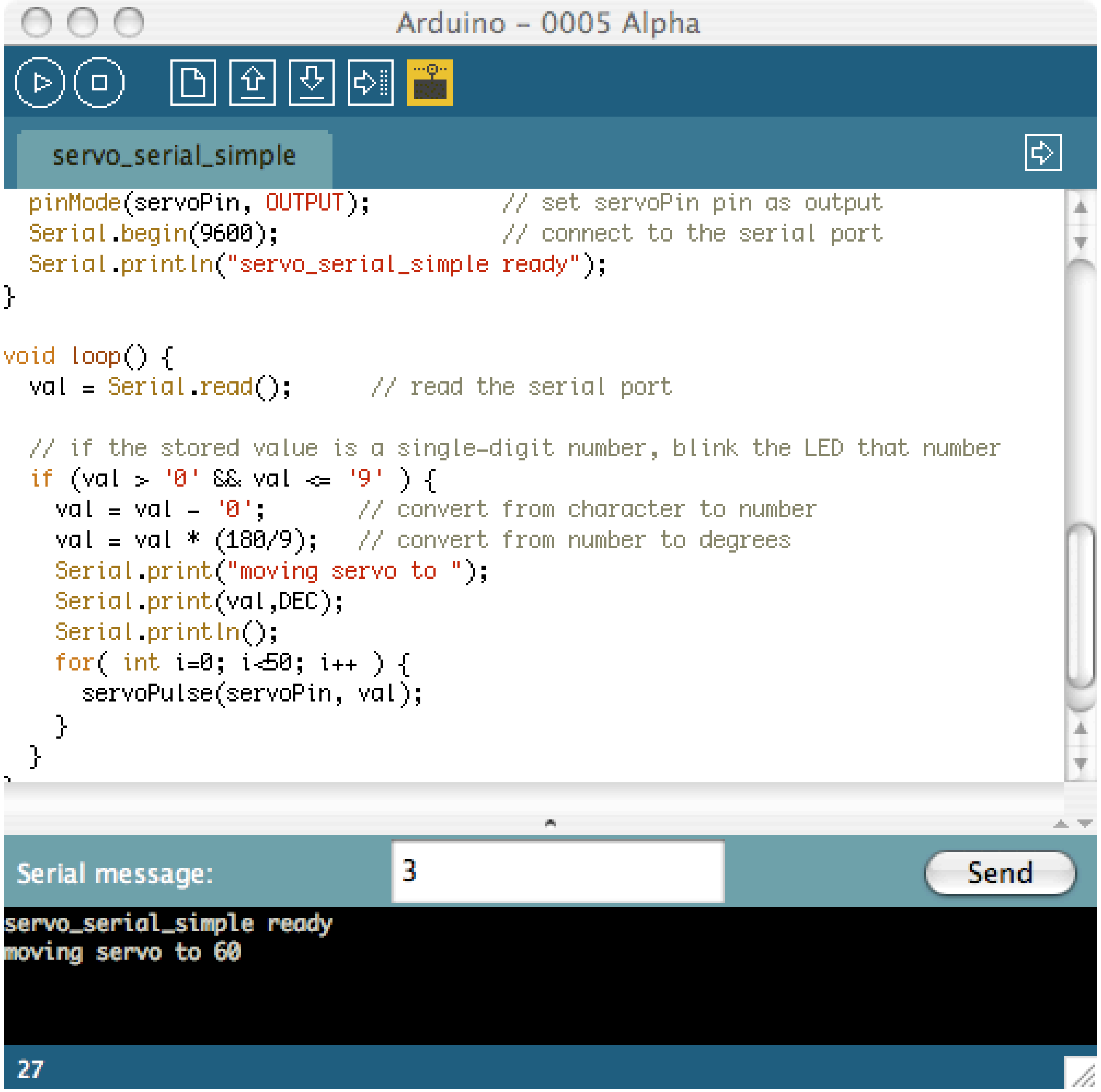
Sketch is in the handout
Created a custom function to handle making servo pulses
New function “delayMicroseconds()”. Like “delay()”, but μsec instead of msec.
(and actually, just delaying 20 msec is kinda wrong. should be: 20 – (pulsewidth/1000)

Serial-controlled Servo

“servo_serial_simple”

Drive the servo by pressing number keys

Takes the last servo example and adds the last serial example to it.



```
  pinMode(servoPin, OUTPUT);          // set servoPin pin as output
  Serial.begin(9600);                 // connect to the serial port
  Serial.println("servo_serial_simple ready");
}

void loop() {
  val = Serial.read();      // read the serial port

  // if the stored value is a single-digit number, blink the LED that number
  if (val > '0' && val <= '9' ) {
    val = val - '0';        // convert from character to number
    val = val * (180/9);    // convert from number to degrees
    Serial.print("moving servo to ");
    Serial.print(val,DEC);
    Serial.println();
    for( int i=0; i<50; i++ ) {
      servoPulse(servoPin, val);
    }
  }
```

Serial message: 3

```
servo_serial_simple ready
moving servo to 60
```

This sketch is located in the handout.
Why that for loop? Because it takes time for the servo to get to a position and it has no memory.

Controlling Arduino

- Any program on the computer, not just the Arduino software, can control the Arduino board
- On Unixes like Mac OS X & Linux, even the command-line can do it:

```
demo% export PORT=/dev/tty.usbserial-A3000Xv0
demo% stty -f $PORT 9600 raw -parenb -parodd cs8 -hupcl -cstopb clocal
demo% printf "1" > $PORT   # rotate servo left
demo% printf "5" > $PORT   # go to middle
demo% printf "9" > $PORT   # rotate servo right
```

Unix is rad.

Take a Break

Servo Timing Problems

- Two problems with the last sketch
  - When `servoPulse()` function runs, nothing else can happen
  - Servo isn’t given periodic pulses to keep it at position

If a servo is not being constantly told what to do, it goes slack and doesn’t lift/push/pull

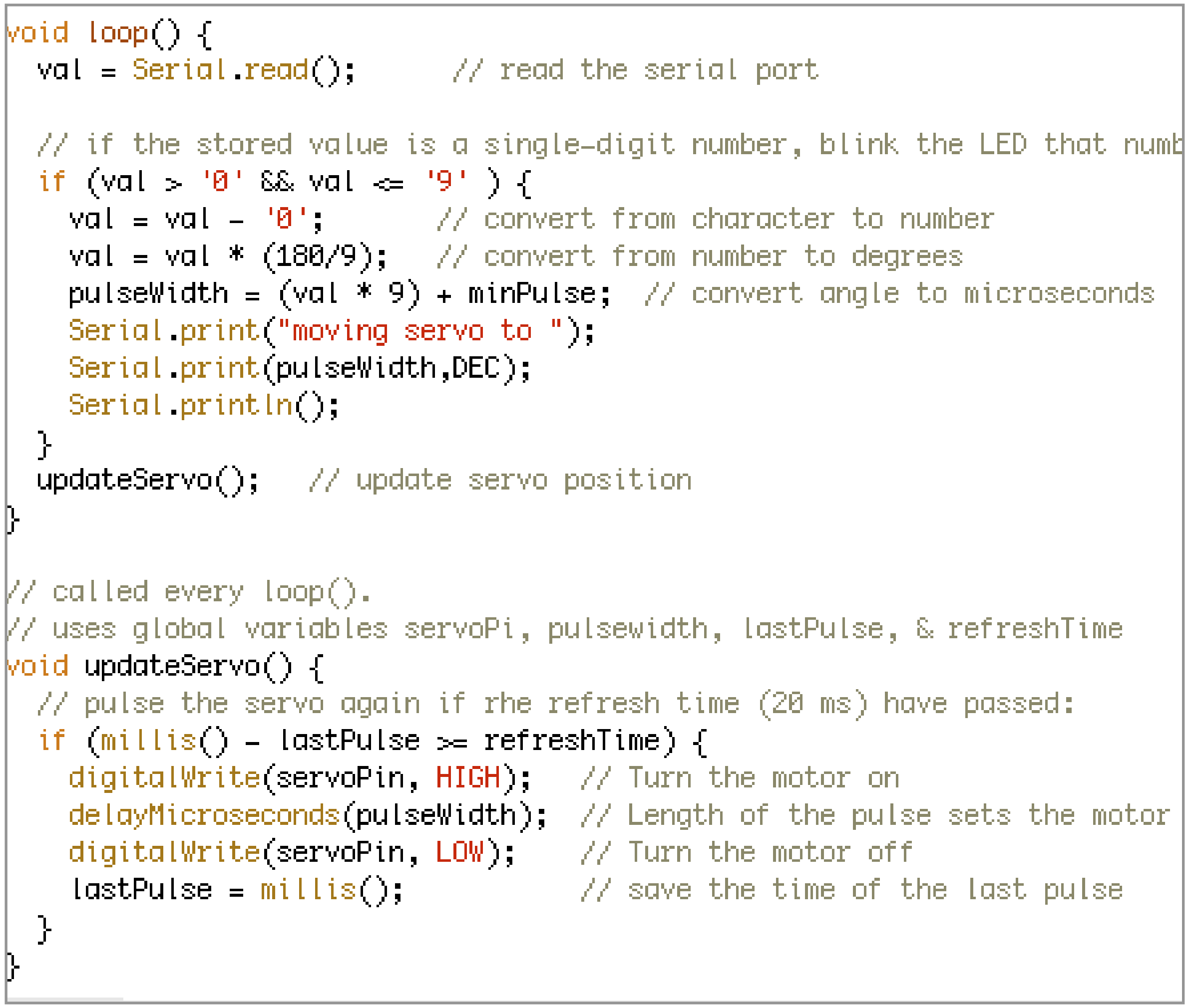
Better Serial Servo

“servo_serial_better”

Works just like
servo_serial_simple
(but better)

Update the servo when needed, not just when called at the right time

Uses “millis()” to know what time it is



```
void loop() {
  val = Serial.read();      // read the serial port

  // if the stored value is a single-digit number, blink the LED that numb
  if (val > '0' && val <= '9' ) {
    val = val - '0';        // convert from character to number
    val = val * (180/9);    // convert from number to degrees
    pulseWidth = (val * 9) + minPulse;  // convert angle to microseconds
    Serial.print("moving servo to ");
    Serial.print(pulseWidth,DEC);
    Serial.println();
  }
  updateServo();   // update servo position
}

// called every loop().
// uses global variables servoPi, pulsewidth, lastPulse, & refreshTime
void updateServo() {
  // pulse the servo again if rhe refresh time (20 ms) have passed:
  if (millis() - lastPulse >= refreshTime) {
    digitalWrite(servoPin, HIGH);   // Turn the motor on
    delayMicroseconds(pulseWidth);  // Length of the pulse sets the motor
    digitalWrite(servoPin, LOW);    // Turn the motor off
    lastPulse = millis();           // save the time of the last pulse
  }
}
```

This sketch is located in the handout.
Trades memory use (the extra variables), for more useful logic.
Can call updateServo() as often as you want, servo is only moved when needed.

Multiple Servos

- The `updateServo()` technique can be extended to many servos
- Only limit really is number of digital output pins you have
- It starts getting tricky after about 8 servos though

Arduino PWM

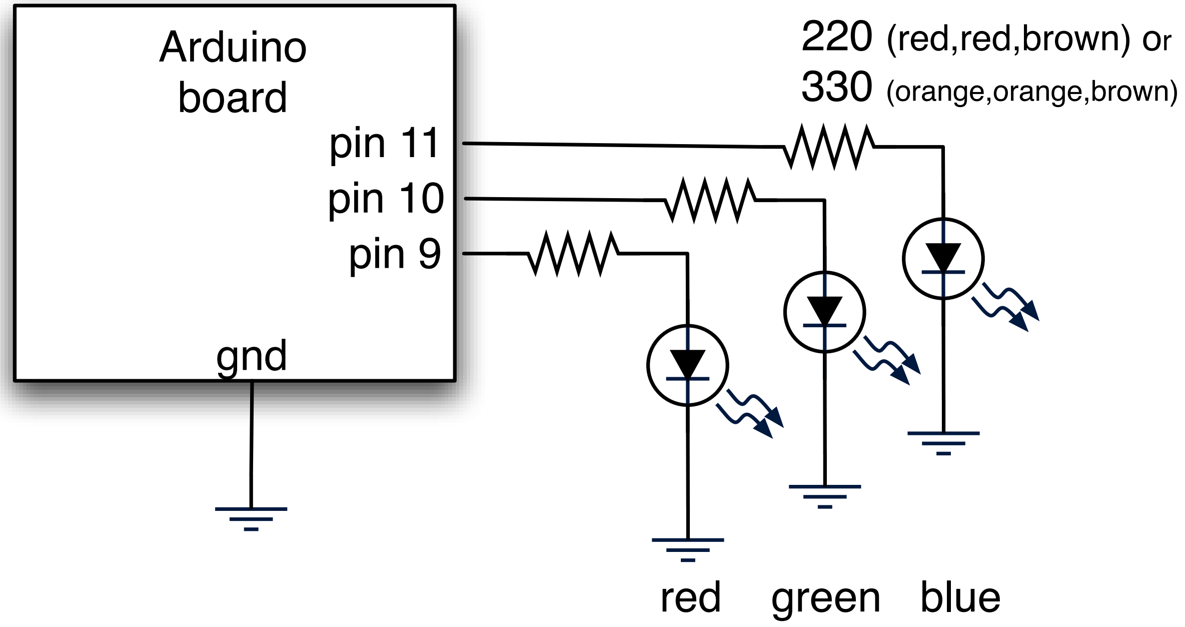
why all the software, doesn't Arduino have PWM?

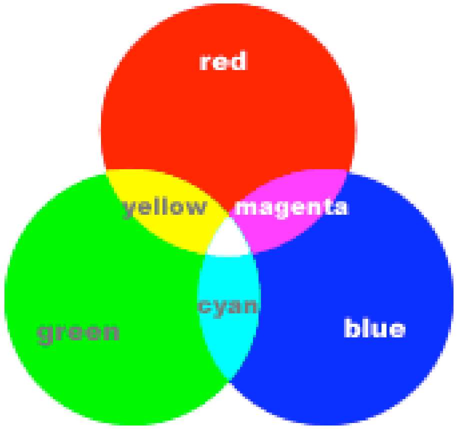
- Arduino has built-in PWM
- On pins 9,10,11
- Use `analogWrite(pin,value)`
- It operates at a high, fixed frequency (thus not usable for servos)
- But great for LEDs and motors
- Uses built-in PWM circuits of the ATmega8 chip -» no software needed

The PWM speed used for analogWrite() is set to 30 kHz currently.
When programming AVRs, PWM speed can be set to just about any value.

R,G,B LEDs

Three PWM outputs and *three* primary colors.
Just screams to be made, doesn't it?

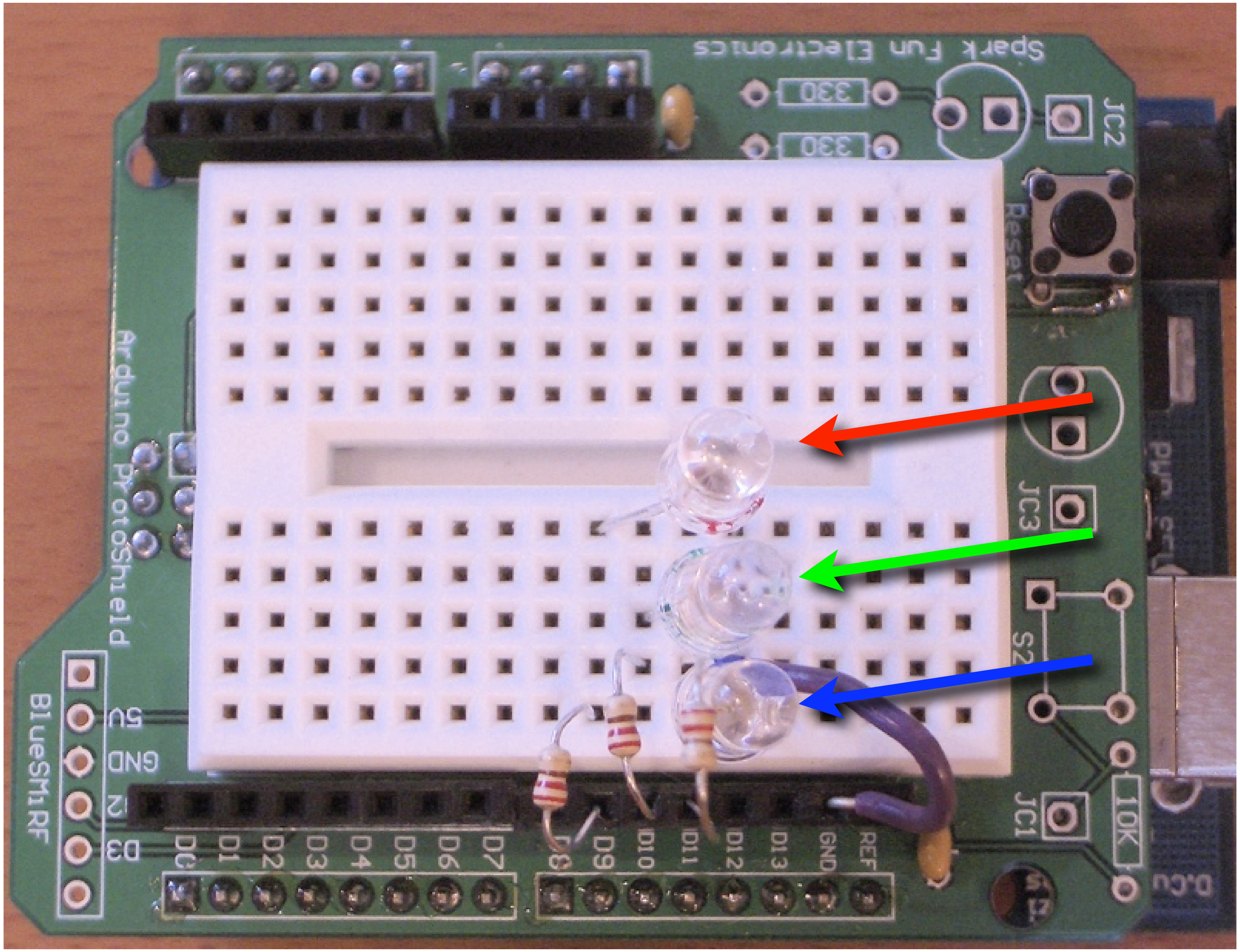




With RGB you can make any color
(except black)

Put back on the ProtoShield for this.
Use either the 220 or 330 ohm resistors in your kit, if you don't have enough of one or the other
I have lots more 220 if you need them

R,G,B LEDs



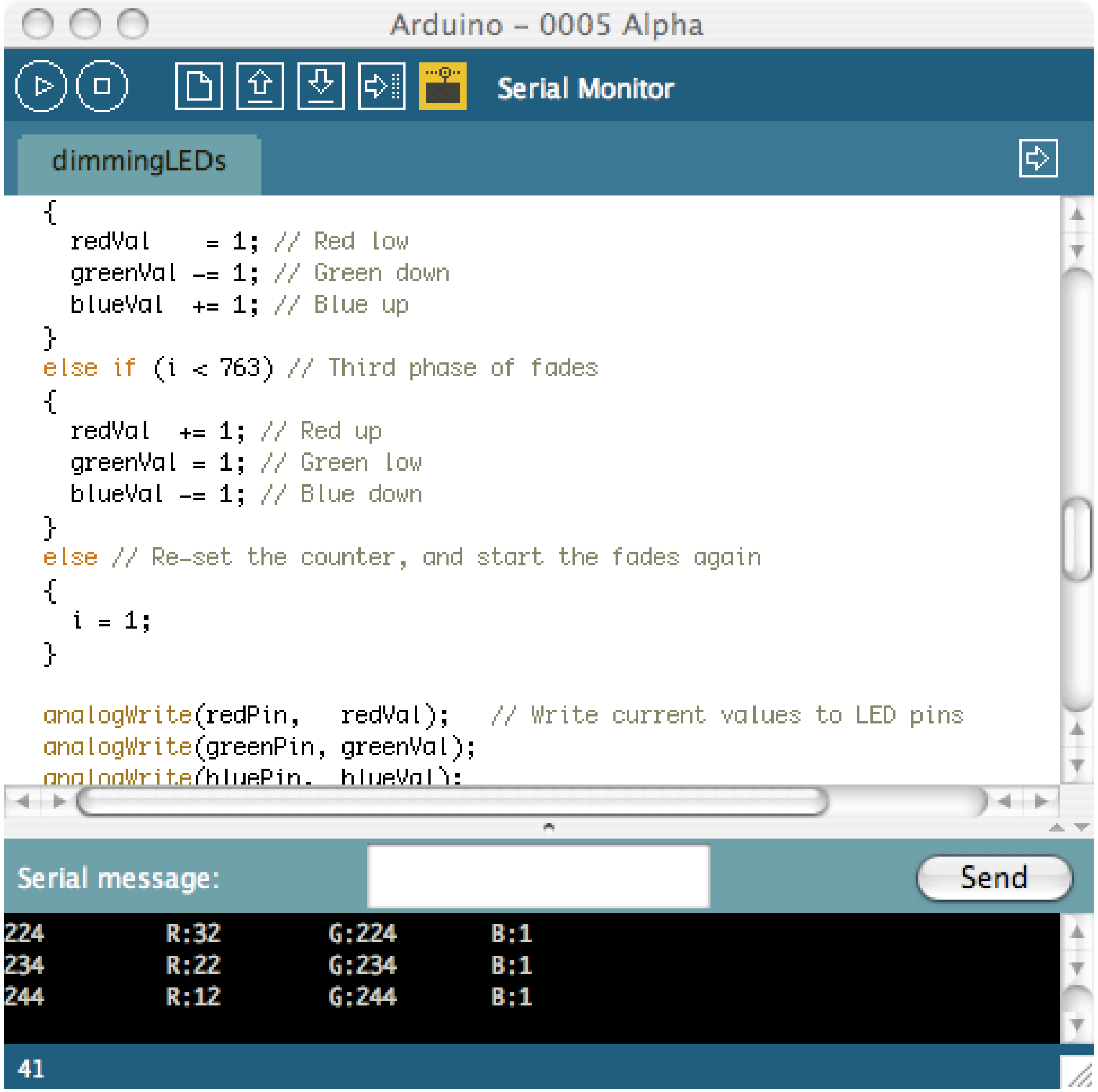
Cut leads of resistors and LEDs to make for a more compact circuit.
Also, less likely to short against itself.

RGB Color Fading

"dimmingLEDs"

Slow color fading and mixing

Also outputs the current color values to the serial port



```
  {
    redVal     = 1; // Red low
    greenVal -= 1; // Green down
    blueVal   += 1; // Blue up
  }
  else if (i < 763) // Third phase of fades
  {
    redVal   += 1; // Red up
    greenVal = 1; // Green low
    blueVal -= 1; // Blue down
  }
  else // Re-set the counter, and start the fades again
  {
    i = 1;
  }

  analogWrite(redPin,   redVal);   // Write current values to LED pins
  analogWrite(greenPin, greenVal);
  analogWrite(bluePin,  blueVal);
```

```
224        R:32        G:224       B:1
234        R:22        G:234       B:1
244        R:12        G:244       B:1
```

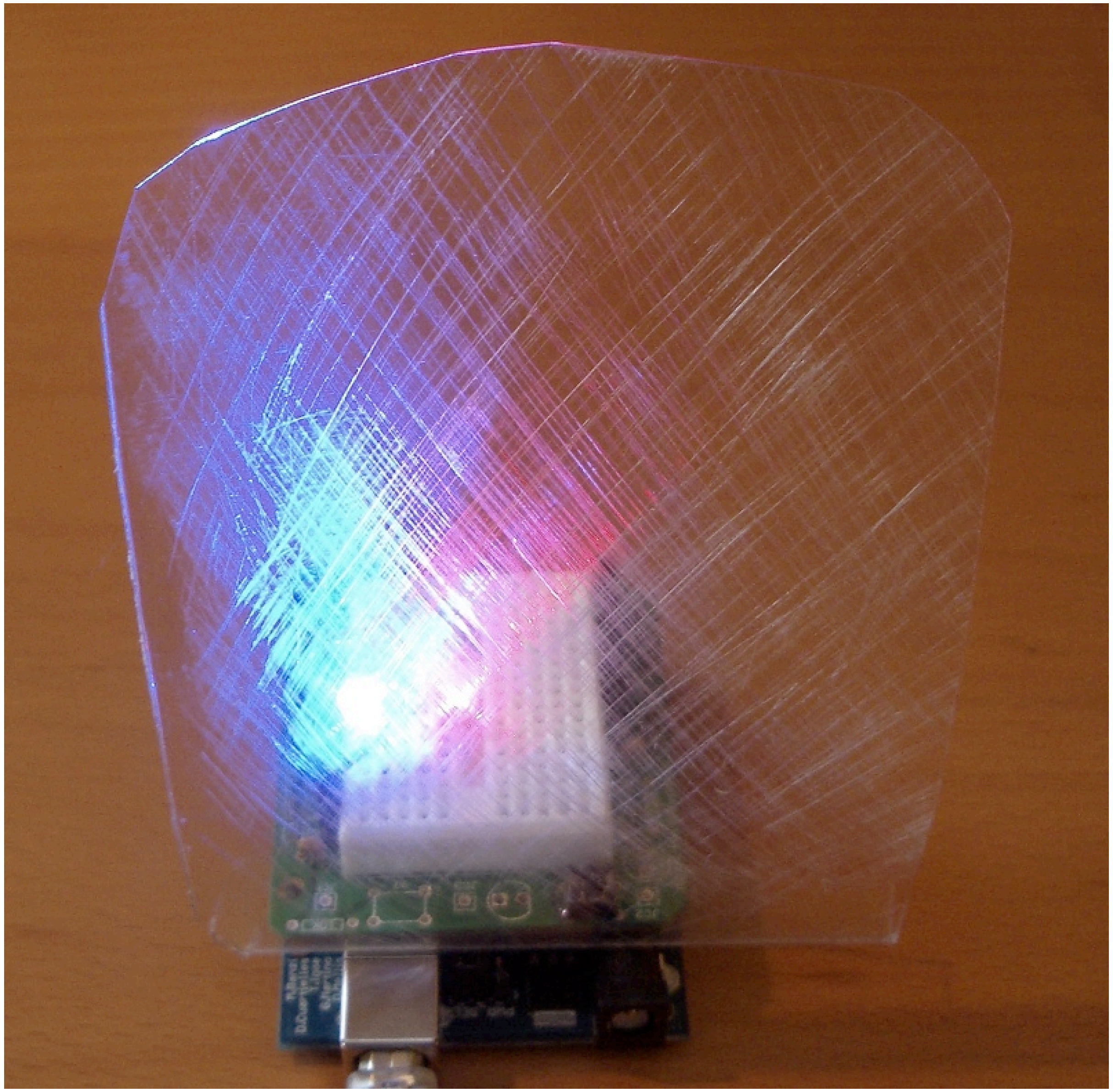
This sketch is located in the handout.

It just ramps up and down the red,green,& blue color values and writes them with analogWrite()

from http://www.arduino.cc/en/Tutorial/DimmingLEDs

Mood Light

Diffuser made from piece of plastic scratched with sandpaper



Also, can use plastic wrap scrunched up to make an interesting diffuser.

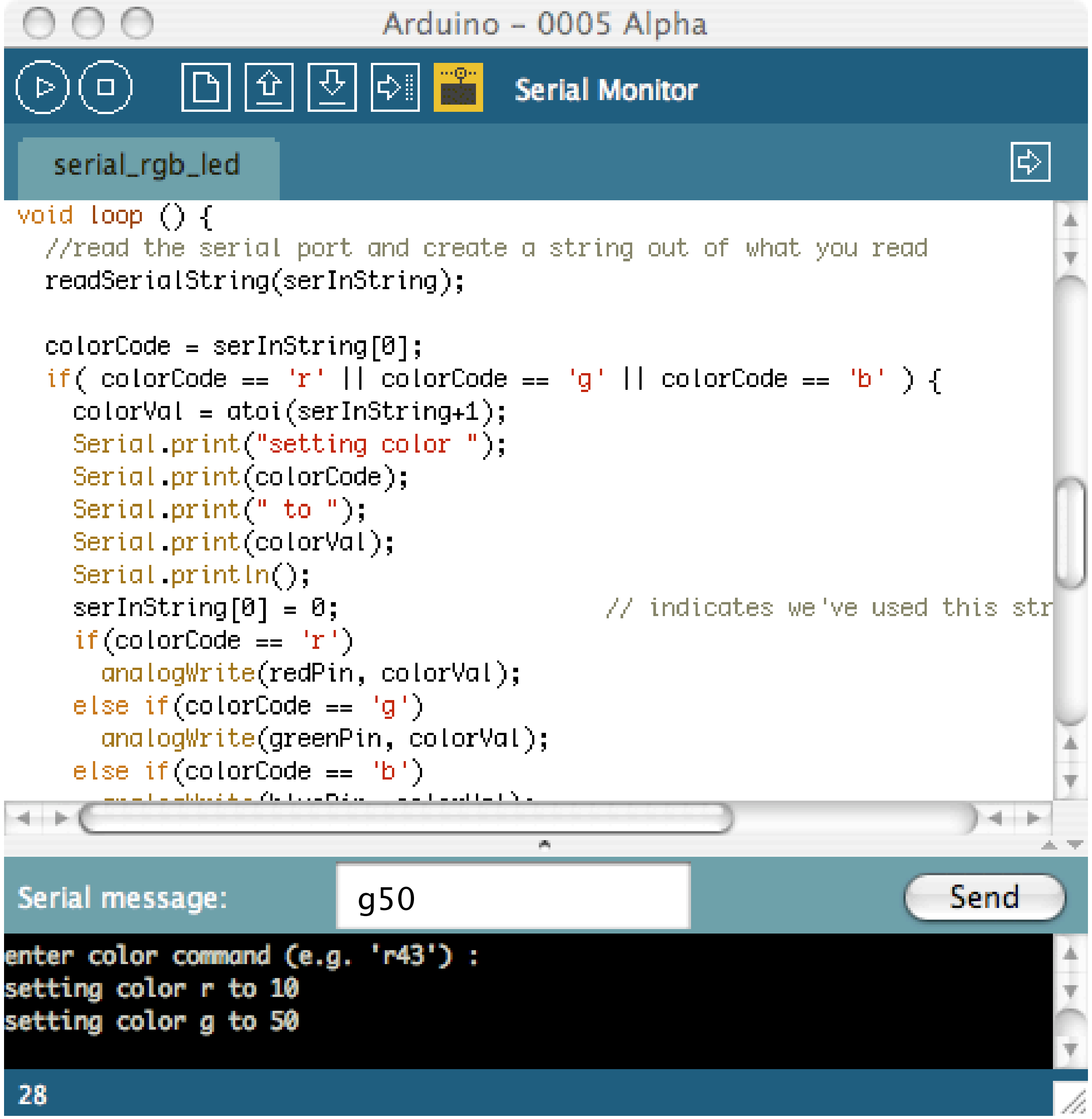
Serial-controlled RGB

"serial_rgb_led"

Send color commands to Arduino

e.g. "r200", "g50", "b0"

Sketch parses what you type, changes LEDs



```
void loop () {
  //read the serial port and create a string out of what you read
  readSerialString(serInString);

  colorCode = serInString[0];
  if( colorCode == 'r' || colorCode == 'g' || colorCode == 'b' ) {
    colorVal = atoi(serInString+1);
    Serial.print("setting color ");
    Serial.print(colorCode);
    Serial.print(" to ");
    Serial.print(colorVal);
    Serial.println();
    serInString[0] = 0;                   // indicates we've used this str
    if(colorCode == 'r')
      analogWrite(redPin, colorVal);
    else if(colorCode == 'g')
      analogWrite(greenPin, colorVal);
    else if(colorCode == 'b')
      analogWrite(bluePin, colorVal);
```

Serial message: g50

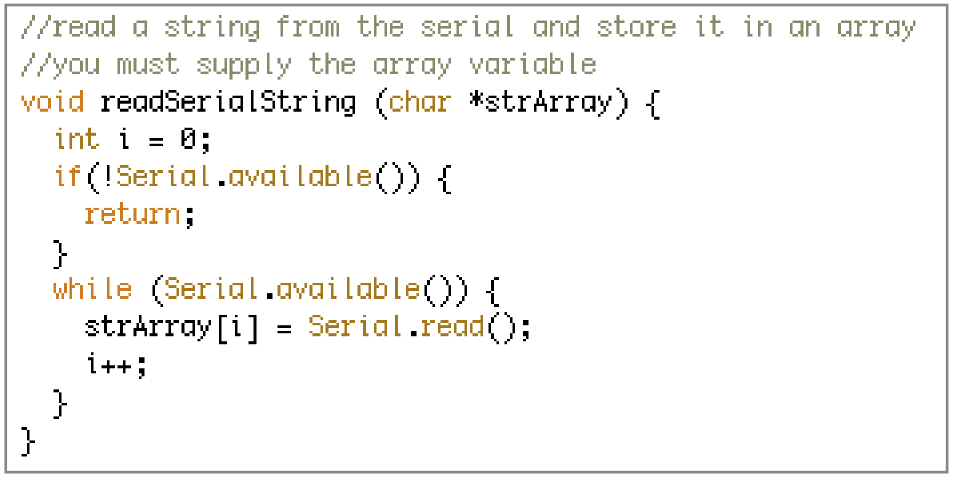
```
enter color command (e.g. 'r43') :
setting color r to 10
setting color g to 50
```

28

This sketch is located in the handout.
Color command is two parts: colorCode and colorValue
colorCode is a character, 'r', 'g', or 'b'.
colorValue is a number between 0-255.
Sketch shows rudimentary character string processing in Arduino

Reading Serial Strings

- New Serial function in last sketch: "`Serial.available()`"
- Can use it to read all available serial data from computer
- Great for reading strings of characters
- The "`readSerialString()`" function at right takes a character string and sticks available serial data into it



```
//read a string from the serial and store it in an array
//you must supply the array variable
void readSerialString (char *strArray) {
  int i = 0;
  if(!Serial.available()) {
    return;
  }
  while (Serial.available()) {
    strArray[i] = Serial.read();
    i++;
  }
}
```

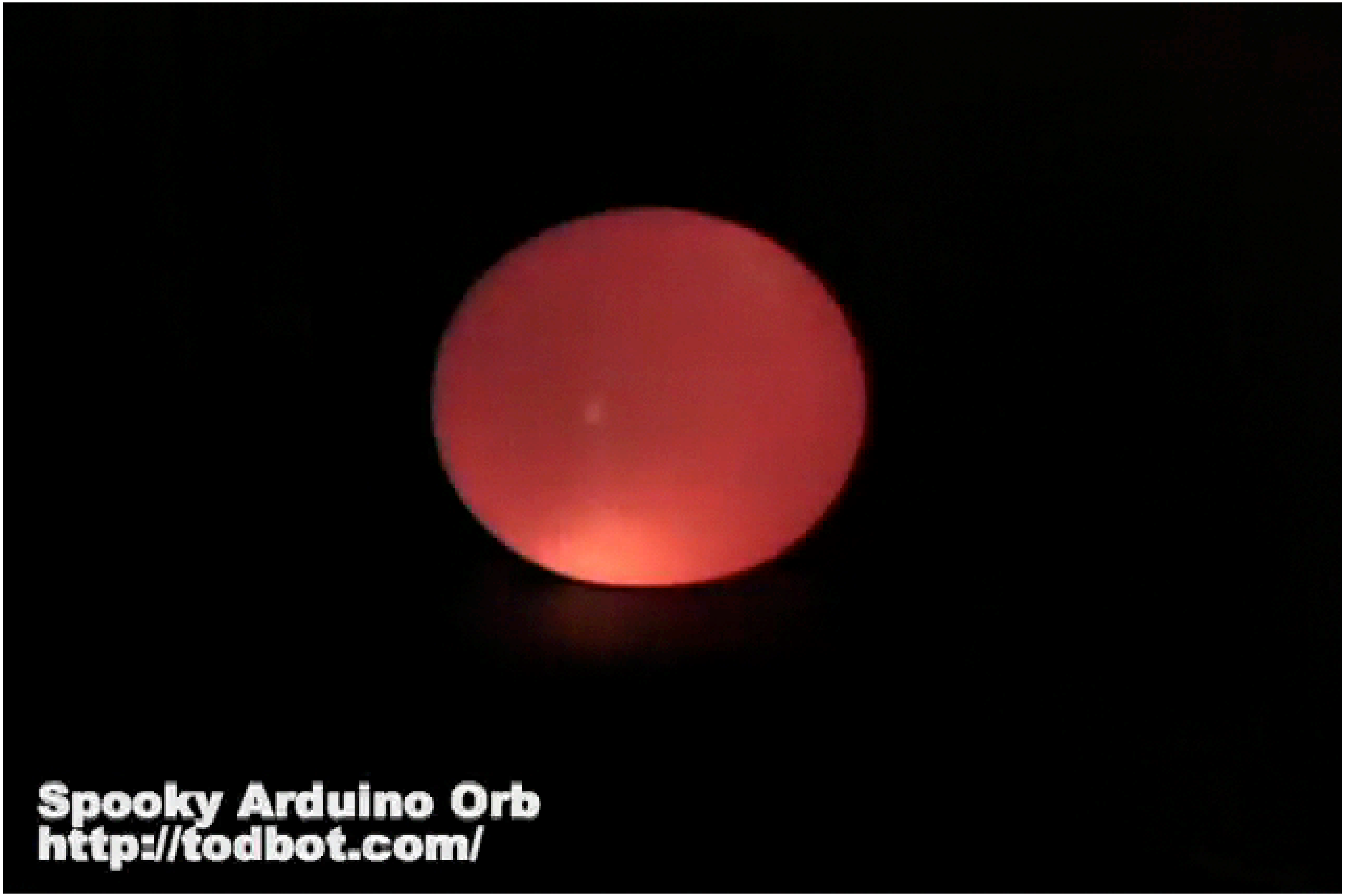
Pay no attention to the pointer symbol ("*")
Must be careful about calling readSerialString() too often or you'll read partial strings

Going Further

- R,G,B LEDS
  - You can pretty easily replicate the Ambient Orb ($150) functionality
  - Make a status display for your computer
  - Computer-controlled accent lighting (a wash of color against the walls)

Ambient Orb doesn't connect to computer though. Uses the pager network.
Ambient Devices: http://www.ambientdevices.com/

Glowing Orb



Going Further

- Servos
  - Mount servo on a video camera – computer-controlled camera motion
  - Make a robot (a little obvious)
  - Lots of spooky uses
    - they're the core of movie animatronics

I'm not too mechanical, so I don't have many concrete and still working examples of servo use.

Going Further

- Serial communications
  - Not just for computer-to-Arduino communications
  - Many other devices speak serial
  - Older keyboards & mice speak are serial (good for sensors!)
  - Interface boards (graphic LCDs, servo drivers, RFID readers, Ethernet, Wi-Fi)

Serial Examples

to Wi-Fi   to Ethernet

to graphic LCD

to 8-servo controller

Lantronix Wi-Port and Lantronix Xport http://lantronix.com/

Seetron Serial Graphic display and Mini SSC http://www.seetron.com/slcds.htm http://www.seetron.com/ssc.htm

Serial Examples

to Roomba

“Hacking Roomba”, out in a few weeks, by me. ;-)
http://hackingroomba.com/

Next Week

- All about piezos
- Building a melody player
- Using piezos as pressure & knock sensors
- Using Processing with Arduino
- Stand-alone Arduino

END Class 3

http://todbot.com/blog/spookyarduino

Tod E. Kurt

tod@todbot.com