PHYSICAL COMPUTING 3: ANALOG OUTPUT

CSE 599 Prototyping Interactive Systems | Lecture 4 | April 11

Jon Froehlich • Jasper O'Leary (TA)

DESIGN USE BUILD

MAKEABILITY LAB

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

UNIVERSITY of WASHINGTON

PHYSICAL COMPUTING 3

TODAY'S LEARNING GOALS

What is **analog output?**

What is **PWM** and why does it matter?

How to use **analogWrite()**

Using **Serial Plotter** for debugging

Introduction to **vibration motors** and how to use them

(Partial) introduction to **potentiometers** and **analog input** (if time)

ASSIGNMENT 1

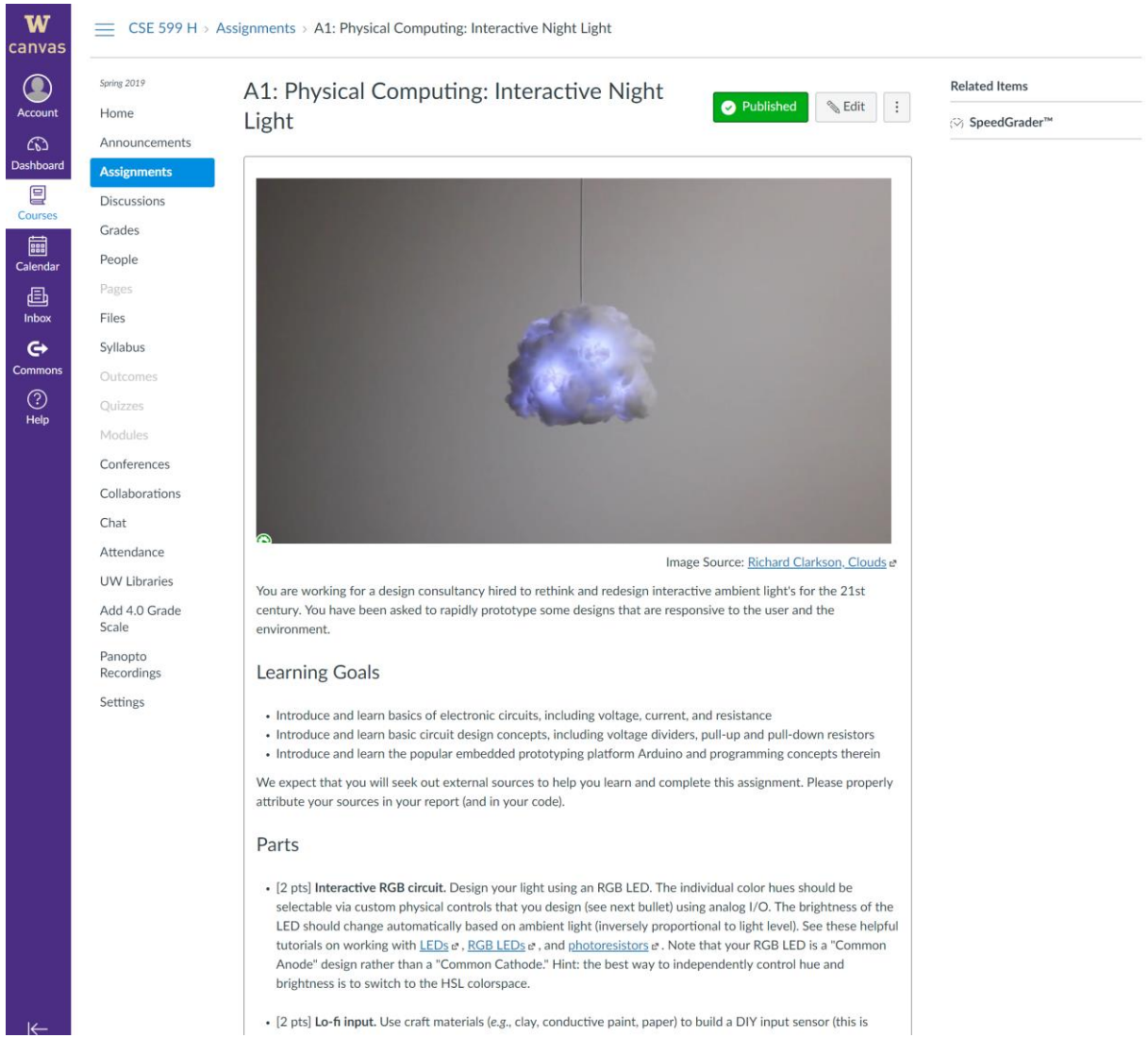
A1: INTERACTIVE NIGHT LIGHT

Due April 18th

We'll do live demos + feedback

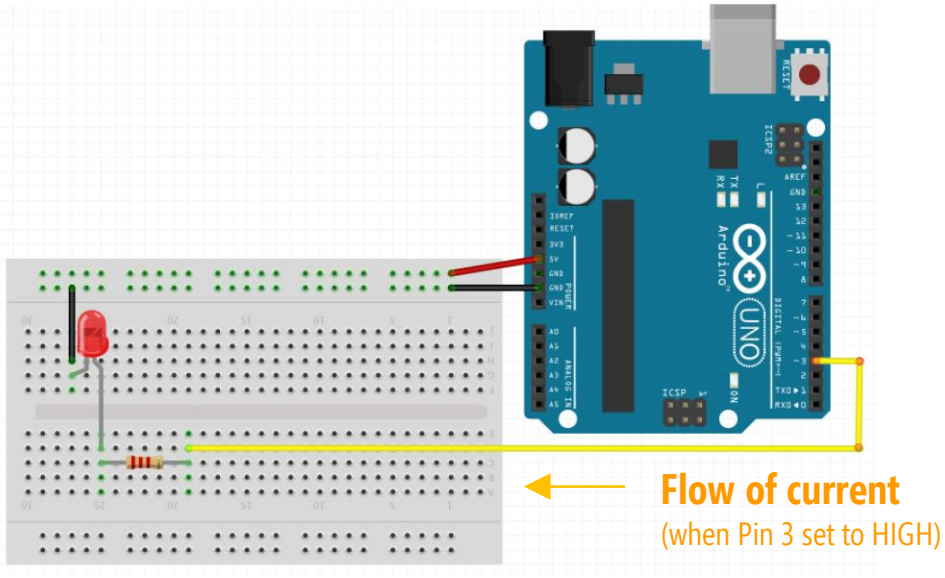
You should be working on this!

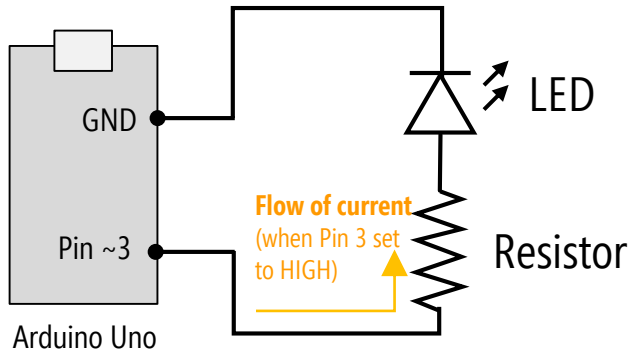
I want you to learn new concepts on your own through making and reinforce concepts from lecture



LAST TIME: WE BUILT THIS! LET'S DO IT AGAIN…

Build Circuit: Flash LED on/off via the pin 3





Write Code: Flash LED on/off via the pin 3

Blink | Arduino 1.8.8

Blink

```
/*
 * Simply turns on and off pin 3 once a second
 * By Jon Froehlich for CSE590
 * http://makeabilitylab.io
 *
 * Adapted from the official Arduino Blink example:
 *   File -> Examples -> 01. Basics -> Blink
 */

const int LED_OUTPUT_PIN = 3;

// The setup function runs once when you press reset or power the board
void setup() {
  // Because pins 0 - 13 can either be input or output, we must specify
  // how we're using the pin by using pinMode. In this case, we want to
  // control an LED, so set the pin to OUTPUT
  pinMode(LED_OUTPUT_PIN, OUTPUT);
}

// The loop function runs over and over again forever
void loop() {
  digitalWrite(LED_OUTPUT_PIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                          // delay is in milliseconds; so wait one second
  digitalWrite(LED_OUTPUT_PIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                          // wait for a second
}
```

Now let's **fade** our LED on and off. To do this, we have to use **analogWrite** rather than **digitalWrite**.

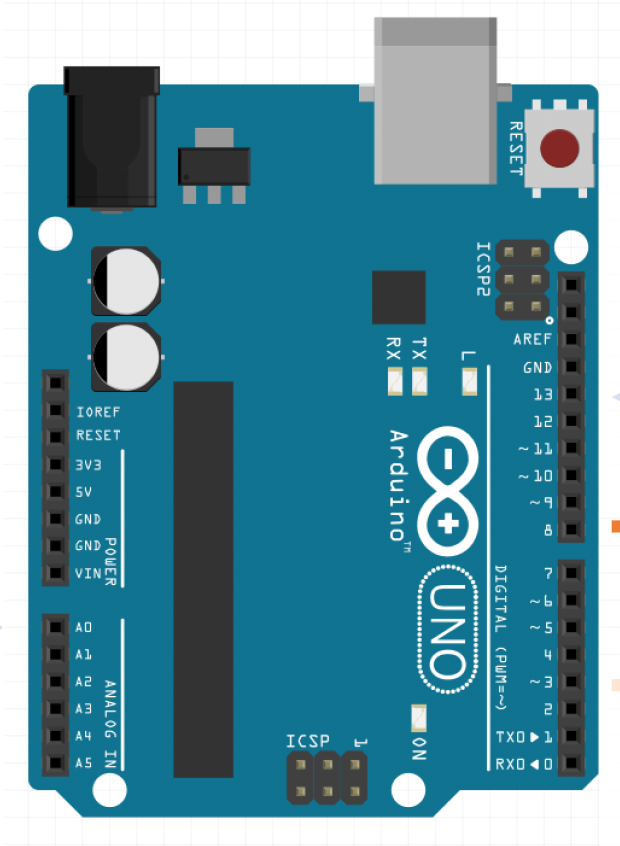
INTRODUCTION TO I/O

INPUT VS. OUTPUT

Input and output is with respect to the microcontroller board. Input is anything that the Arduino Uno reads in (*e.g.*, from a sensor) and output is anything that the Arduino Uno writes out (*e.g.*, to a light or motor)

Analog Input on A0

Reads in any value between 0V or 5V using the analogRead function. In this case, the value of a photocell



Digital Input on pin 13

Reads in a digital input signal (anything below 2.5V converted to LOW, anything above 2.5V converted to HIGH). In this case, the value of switch.

Digital Output on pin 8

Writes out 0V or 5V using digitalWrite function. In this case, turning on and off an LED.

Analog Output on pin 3

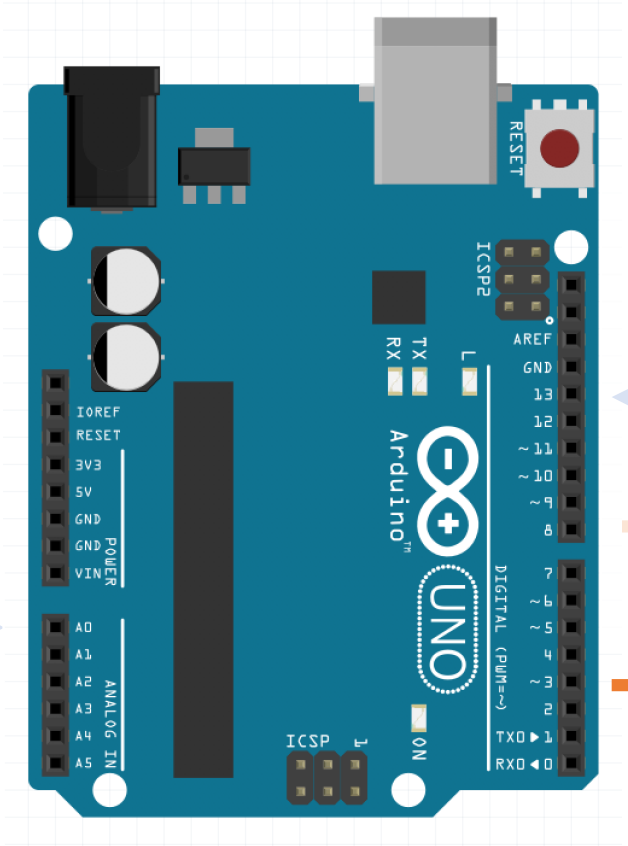
Writes out any value between 0V or 5V using analogWrite function. In this case, vibrating a motor (where strength of vibration proportional to voltage). Only pins with a tilde ~ can be analog outputs.

INPUT VS. OUTPUT

Input and output is with respect to the microcontroller board. Input is anything that the Arduino Uno reads in (*e.g.*, from a sensor) and output is anything that the Arduino Uno writes out (*e.g.*, to a light or motor)

Analog Input on A0

Reads in any value between 0V or 5V using the analogRead function. In this case, the value of a photocell



Digital Input on pin 13

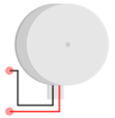
Reads in a digital input signal (anything below 2.5V converted to LOW, anything above 2.5V converted to HIGH). In this case, the value of switch.

Digital Output on pin 8

Writes out 0V or 5V using digitalWrite function. In this case, turning on and off an LED.

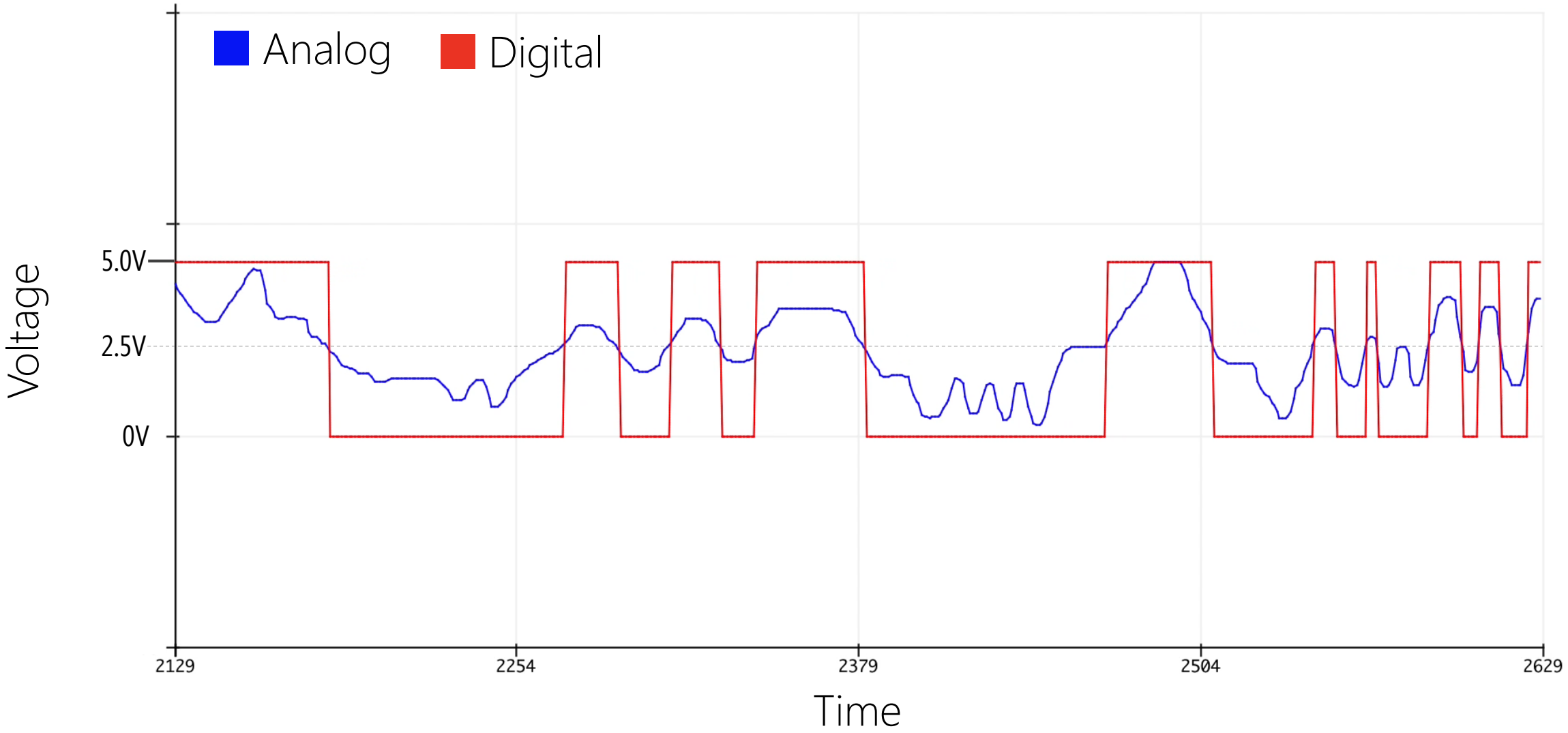
Analog Output on pin 3

Writes out any value between 0V or 5V using analogWrite function. In this case, vibrating a motor (where strength of vibration proportional to voltage). Only pins with a tilde ~ can be analog outputs.



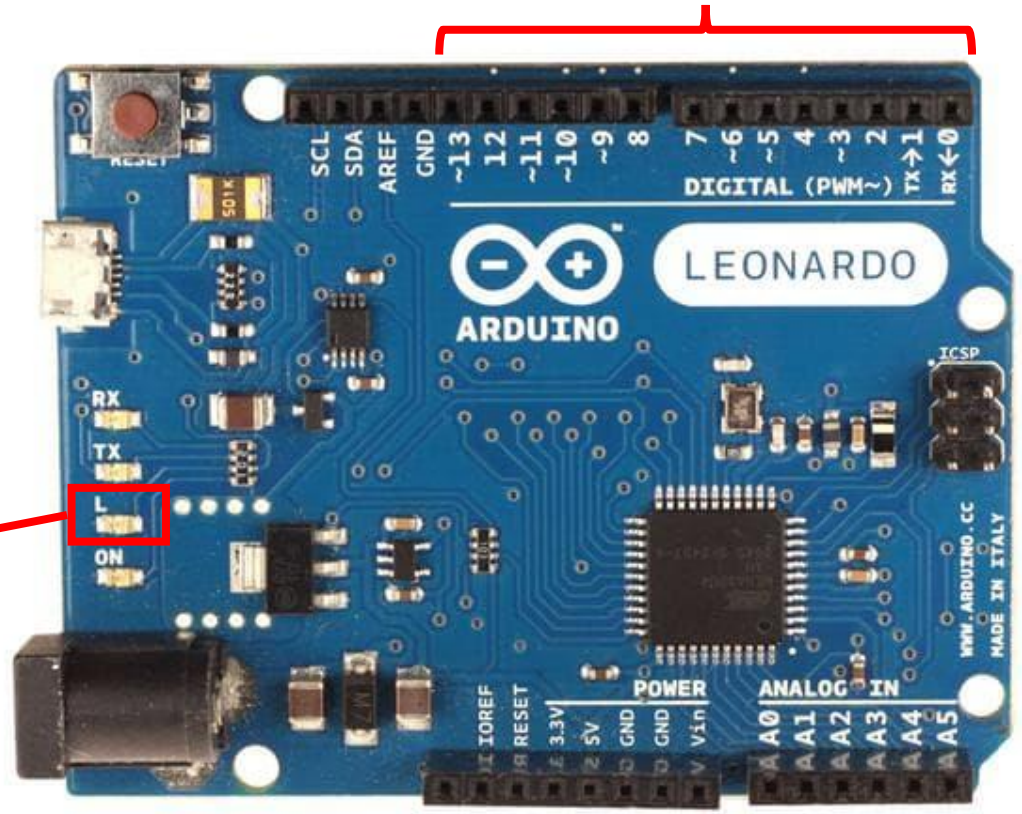
ANALOG VS. DIGITAL

digitalRead and digitalWrite can only be LOW or HIGH corresponding to 0V and 5V.



ARDUINO LEONARDO DIGITAL I/O

14 digital I/O pins: these pins can be used for either input or output (set the mode using the pinMode function). Max output current is 40mA (total across all pins is 200mA)

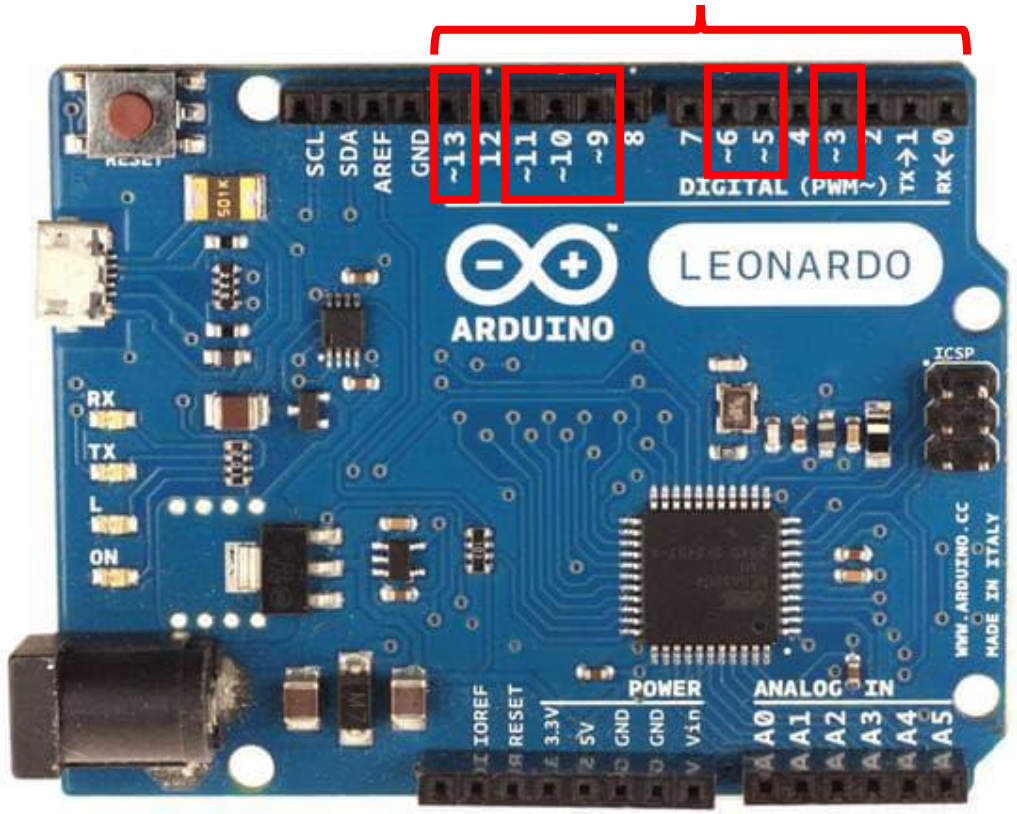


'L' LED: The 'L' LED lights up when pin D13 goes HIGH. This is useful for debugging.

Source: https://www.arduino.cc/en/Guide/ArduinoLeonardoMicro; https://store.arduino.cc/usa/Leonardo; https://learn.sparkfun.com/tutorials/arduino-comparison-guide/atmega32u4-boards

ARDUINO LEONARDO ANALOG OUTPUT

7 analog output pins: pins with the '~' symbol can be used to simulate analog output using 8-bit pulse-width modulation (PWM). Again, max current is 40mA per pin.



Source: https://www.arduino.cc/en/Guide/ArduinoLeonardoMicro; https://store.arduino.cc/usa/Leonardo; https://learn.sparkfun.com/tutorials/arduino-comparison-guide/atmega32u4-boards

ANALOG OUTPUT

analogWrite(pin, value)

Writes an analog value between 0 and 5V to a pin using pulse-width modulation (PWM).

Syntax

`analogWrite(pin, value)`

Parameters

`pin`: the pin to write to.

`value`: an integer value between 0 & 255 which roughly maps to 0 – 5V on the Arduino Uno.

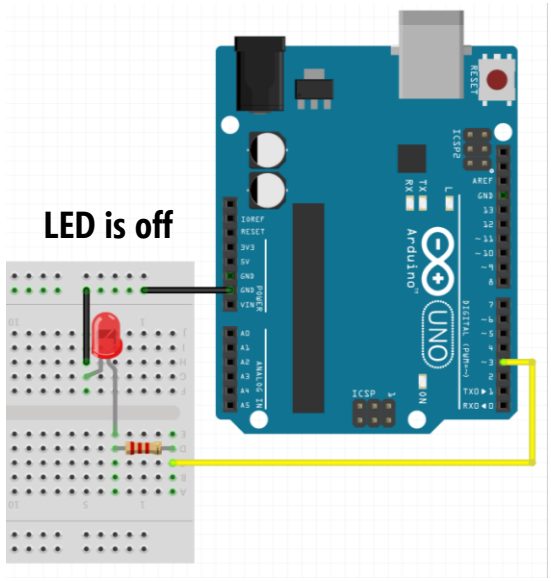
FadeOneDirection §

```
 * Adapted from http://www.arduino.cc/en/Tutorial/Fade
 *
 */

const int LED_OUTPUT_PIN = 3;
int _curBrightness = 0;   // how bright the LED is

// The setup function runs once when you press reset or power the board
void setup() {
  // set the LED pin to as an output
  pinMode(LED_OUTPUT_PIN, OUTPUT);
}

// The loop function runs over and over again forever
void loop() {

  // set the brightness of the LED pin:
  analogWrite(LED_OUTPUT_PIN, _curBrightness);

  // change the brightness for next time through the loop:
  _curBrightness = _curBrightness + 1;

  // the maximum value that we can write out to analogWrite is 255
  // so check to see if the current brightness value is greater than 255
  // and if it is, reset the brightness to 0 (which is off)
  if (_curBrightness > 255){
    _curBrightness = 0;
  }

  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/

ANALOGWRITE EXAMPLE

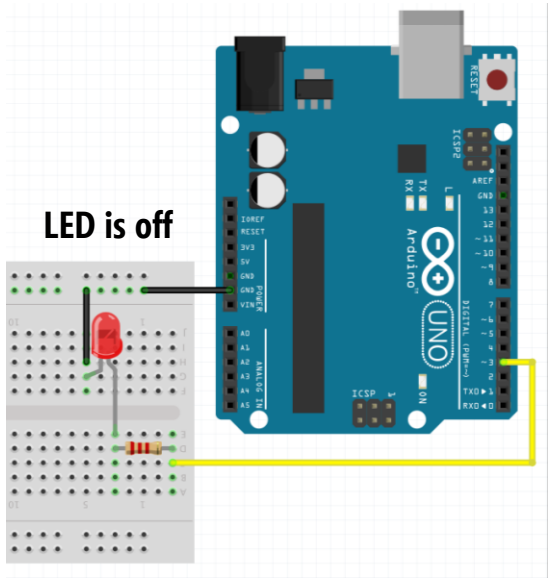
Recall that analogWrite(pin, value) where value is 0 – 255, which is linearly mapped to 0 – 5V




```
analogWrite(3, ?);
```

ANALOGWRITE EXAMPLE

Recall that analogWrite(pin, value) where value is 0 – 255, which is linearly mapped to 0 – 5V



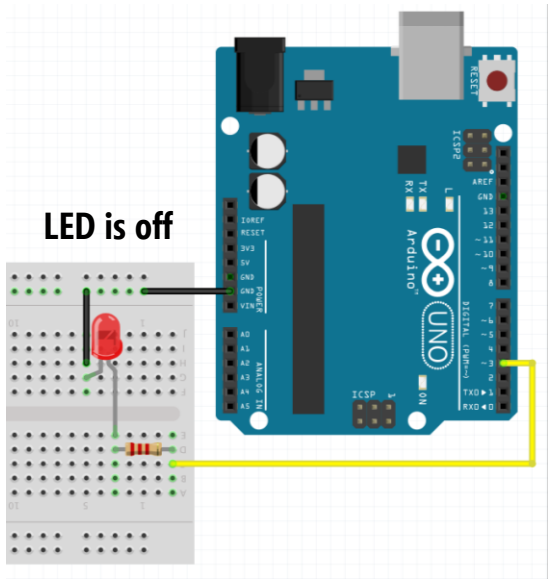

```
analogWrite(3, 0);
```

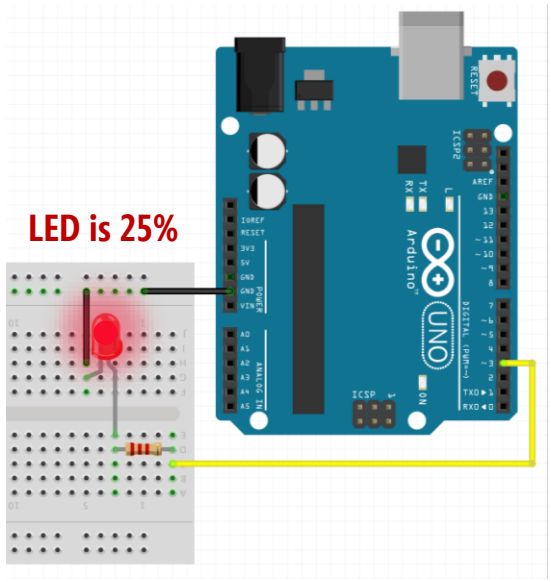
0% brightness (off)

(because 0/255 = 0)

ANALOGWRITE EXAMPLE

Recall that analogWrite(pin, value) where value is 0 – 255, which is linearly mapped to 0 – 5V






```
analogWrite(3, 0);
```

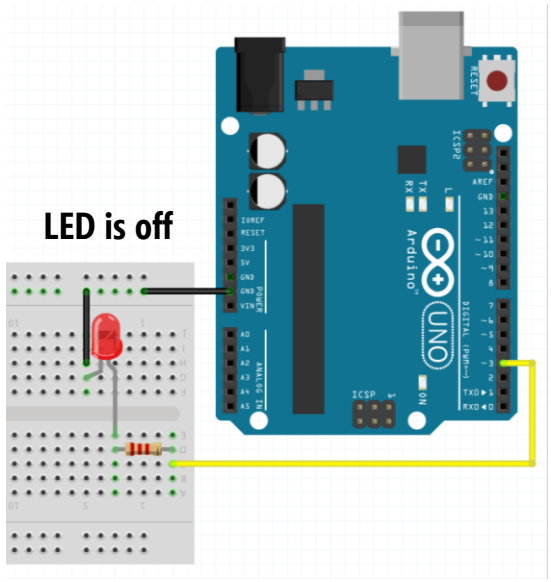
0% brightness (off)

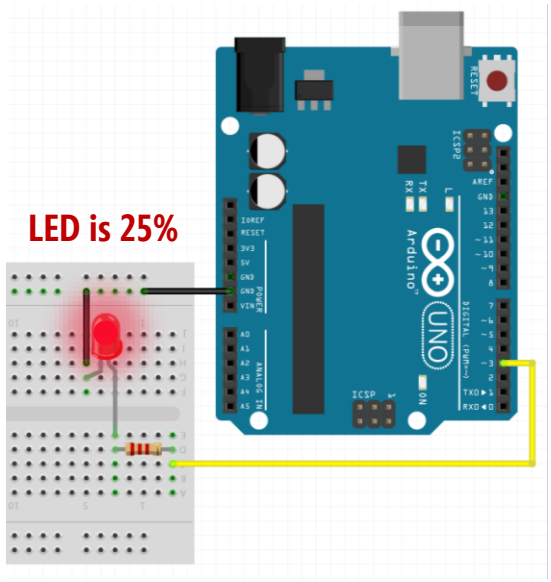
(because 0/255 = 0)

```
analogWrite(3, ?);
```

ANALOGWRITE EXAMPLE

Recall that analogWrite(pin, value) where value is 0 – 255, which is linearly mapped to 0 – 5V



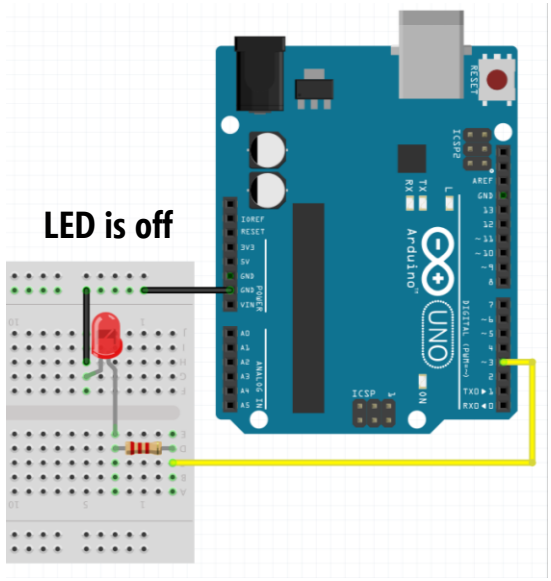


`analogWrite(3, 0);`
0% brightness (off)
(because 0/255 = 0)

`analogWrite(3, 64);`
25% brightness (1.25V)
(because 64/255 = 25%)

ANALOGWRITE EXAMPLE

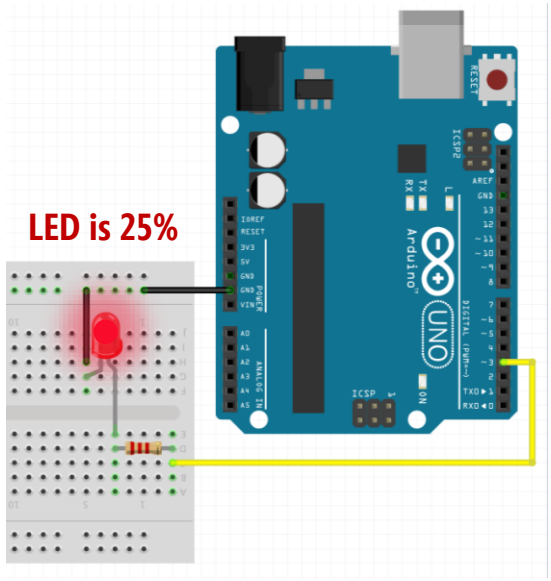
Recall that analogWrite(pin, value) where value is 0 – 255, which is linearly mapped to 0 – 5V



`analogWrite(3, 0);`

0% brightness (off)

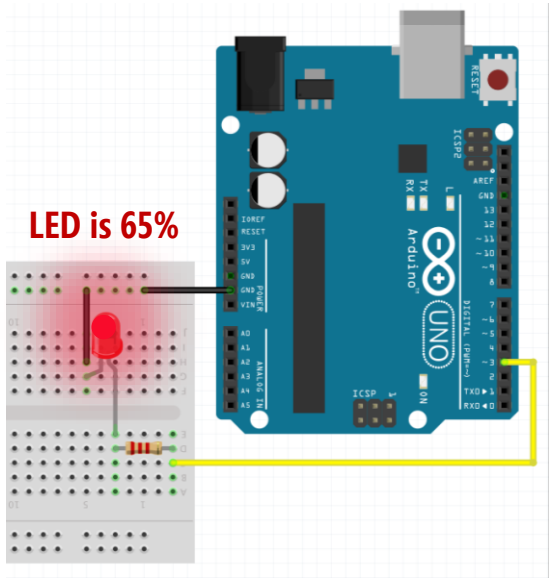
(because 0/255 = 0)



`analogWrite(3, 64);`

25% brightness (1.25V)

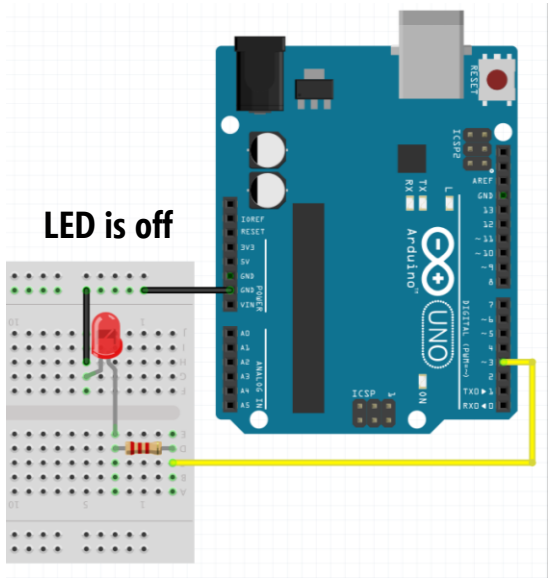
(because 64/255 = 25%)



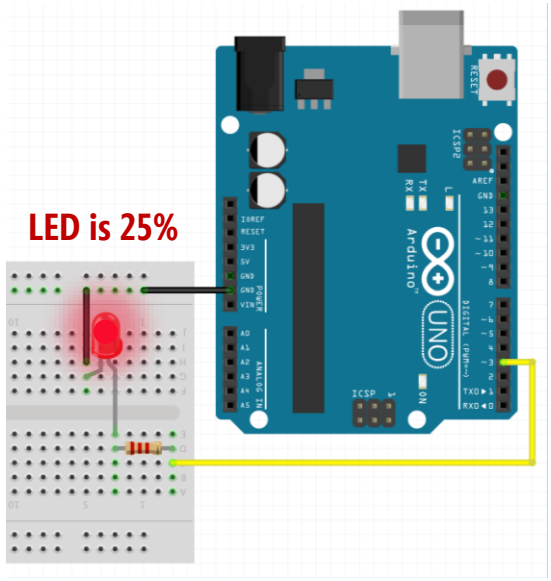
`analogWrite(3, ?);`

ANALOGWRITE EXAMPLE

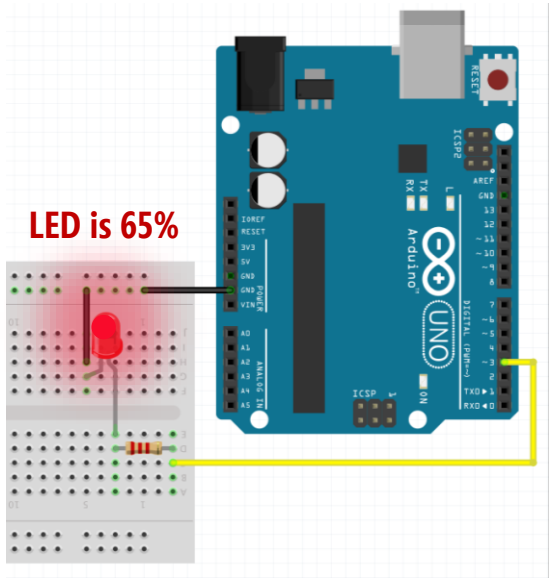
Recall that analogWrite(pin, value) where value is 0 – 255, which is linearly mapped to 0 – 5V



`analogWrite(3, 0);`
0% brightness (off)
(because 0/255 = 0)



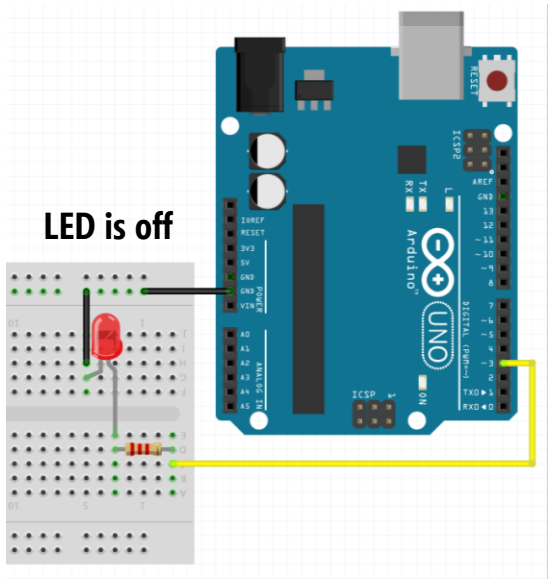
`analogWrite(3, 64);`
25% brightness (1.25V)
(because 64/255 = 25%)



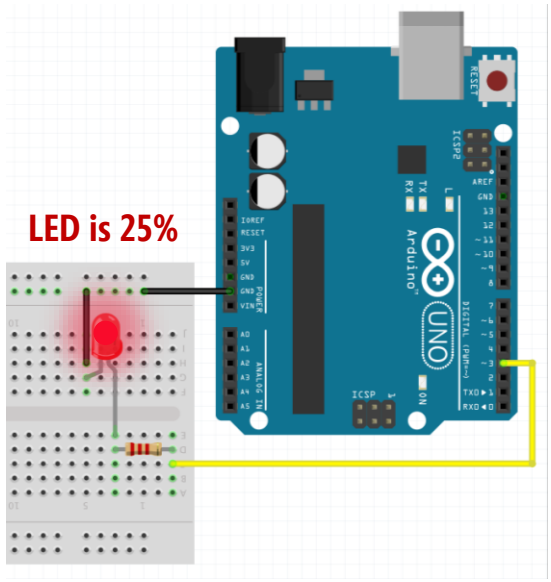
`analogWrite(3, 166);`
65% brightness (3.25V)
(because 166/255 = 65%)

ANALOGWRITE EXAMPLE

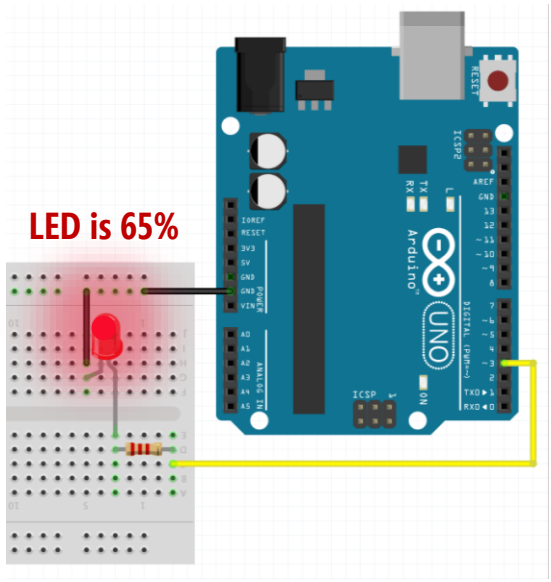
Recall that analogWrite(pin, value) where value is 0 – 255, which is linearly mapped to 0 – 5V



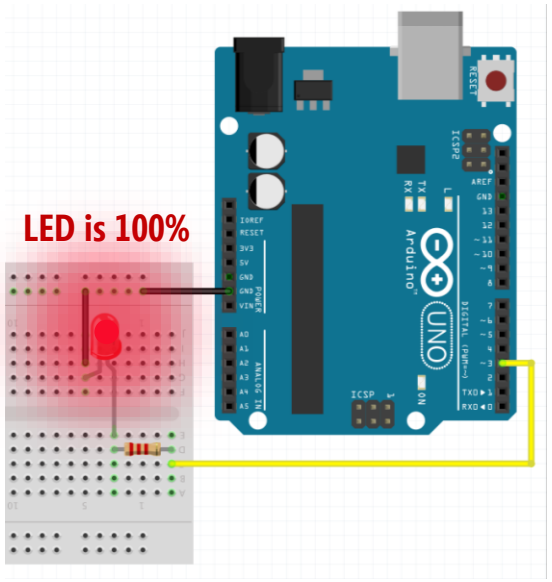
`analogWrite(3, 0);`
0% brightness (off)
(because 0/255 = 0)



`analogWrite(3, 64);`
25% brightness (1.25V)
(because 64/255 = 25%)



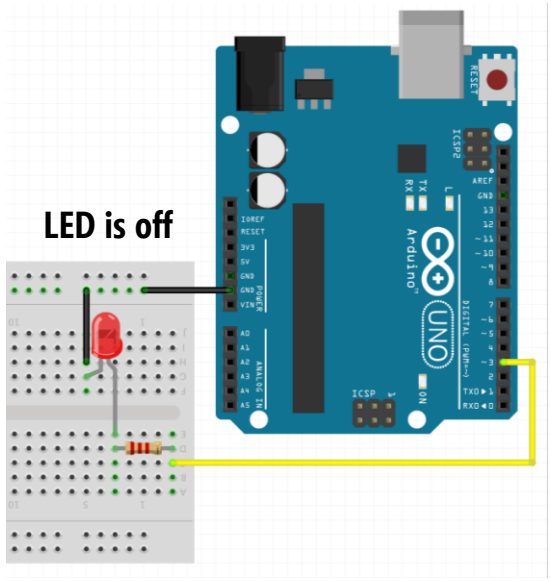
`analogWrite(3, 166);`
65% brightness (3.25V)
(because 166/255 = 65%)



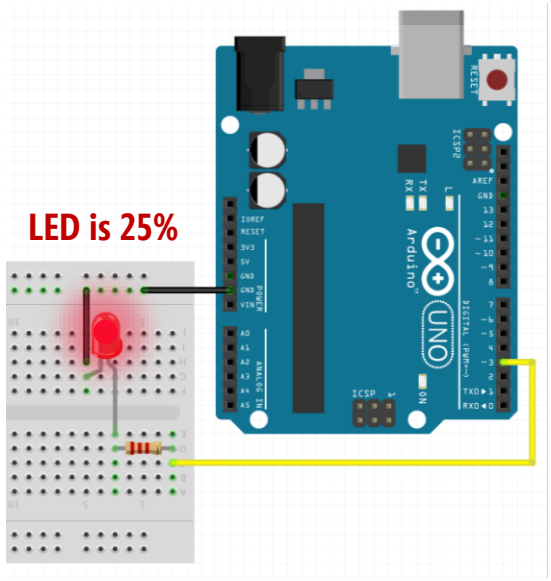
`analogWrite(3, ?);`

ANALOGWRITE EXAMPLE

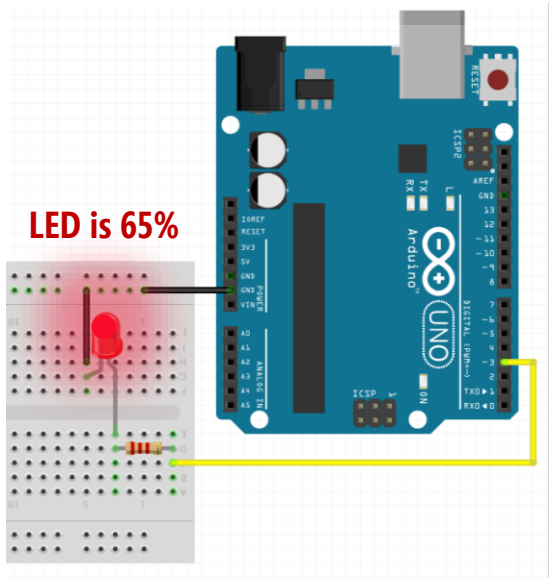
Recall that analogWrite(pin, value) where value is 0 – 255, which is linearly mapped to 0 – 5V



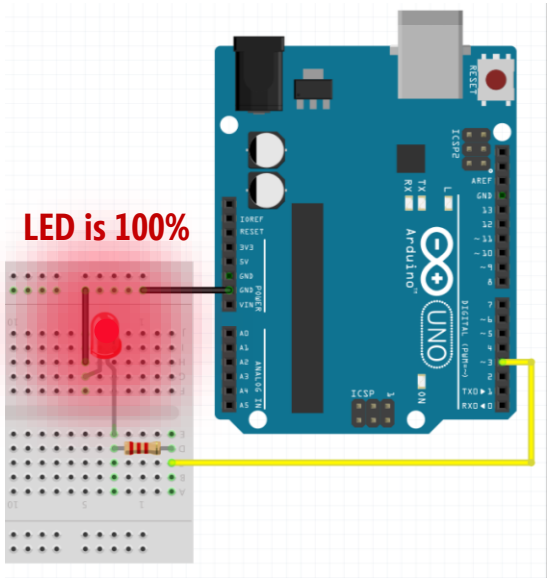
`analogWrite(3, 0);`
0% brightness (off)
(because 0/255 = 0)



`analogWrite(3, 64);`
25% brightness (1.25V)
(because 64/255 = 25%)



`analogWrite(3, 166);`
65% brightness (3.25V)
(because 166/255 = 65%)



`analogWrite(3, 255);`
100% brightness (5V)
(because 255/255 = 100%)

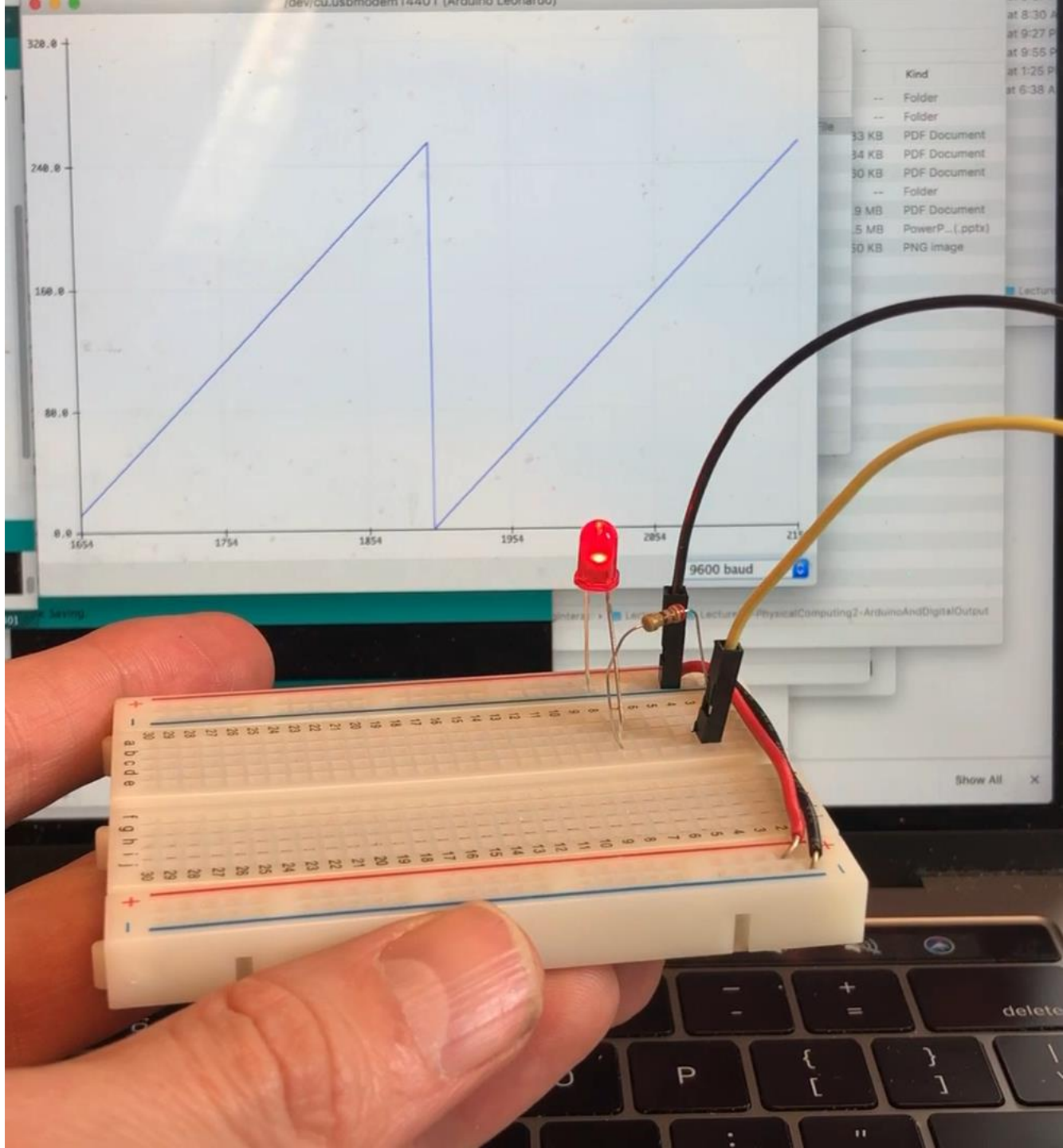
ANALOG OUTPUT

ACTIVITY: FADE LED ON

Fade the LED on using analogWrite

Remember, the **analog output** is limited to 8-bits (0-255)

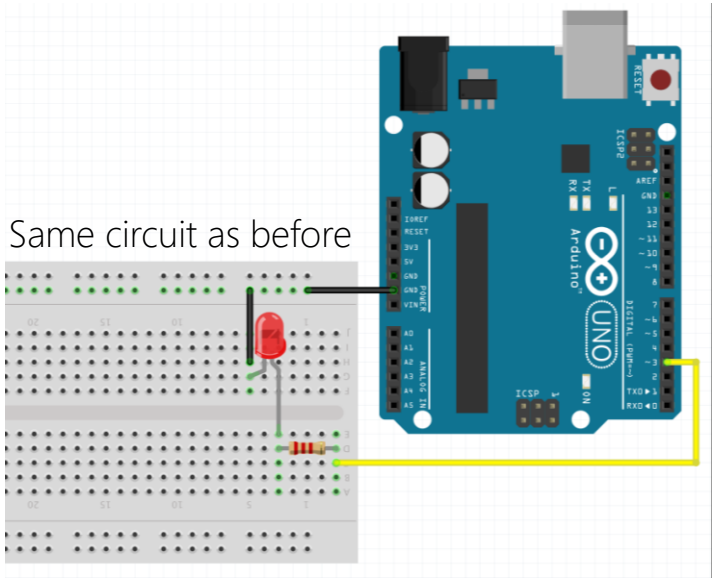
I’m not going to show you the circuit diagram or the code. Try it yourself!

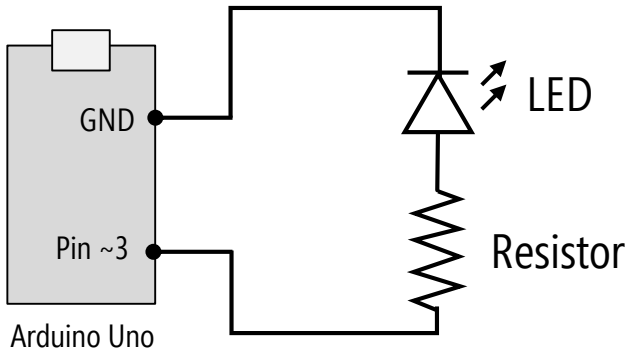


ACTIVITY: FADE LED ON

The circuit is the same as the flashing LED example but the code needs to change.

Circuit: Fade LED on via the pin 3





Code: Fade LED on via pin 3

FadeOn

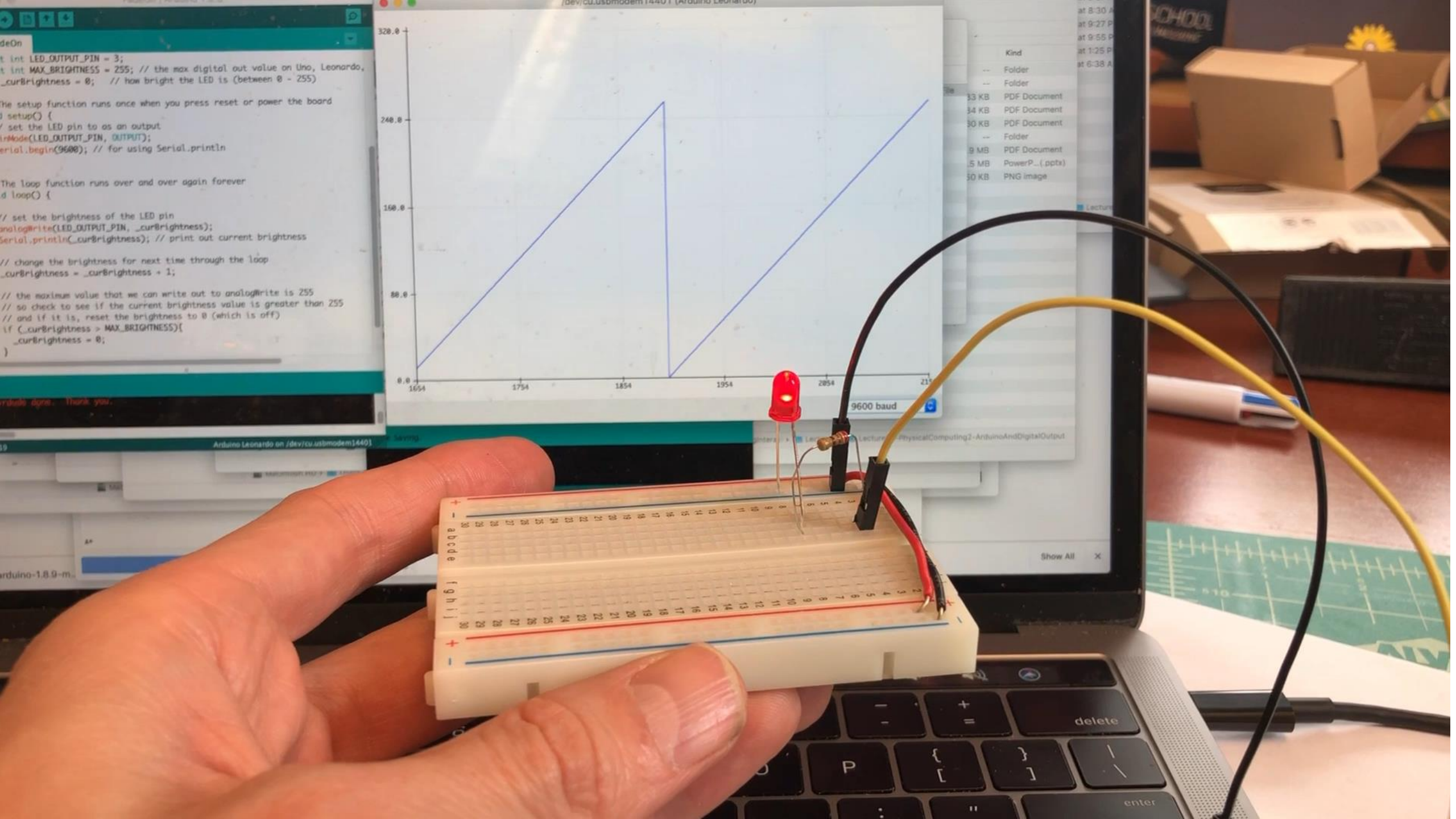
```
const int LED_OUTPUT_PIN = 3;
const int MAX_BRIGHTNESS = 255; // the max digital out value on Uno, Leonardo, etc. is 255 (8-bit max)
int _curBrightness = 0;   // how bright the LED is (between 0 - 255)

// The setup function runs once when you press reset or power the board
void setup() {
  // set the LED pin to as an output
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  Serial.begin(9600); // for using Serial.println
}

// The loop function runs over and over again forever
void loop() {

  // set the brightness of the LED pin
  analogWrite(LED_OUTPUT_PIN, _curBrightness);
  Serial.println(_curBrightness); // print out current brightness

  // change the brightness for next time through the loop
  _curBrightness = _curBrightness + 1;

  // the maximum value that we can write out to analogWrite is 255
  // so check to see if the current brightness value is greater than 255
  // and if it is, reset the brightness to 0 (which is off)
  if (_curBrightness > MAX_BRIGHTNESS){
    _curBrightness = 0;
  }

  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

DEBUGGING ARDUINO

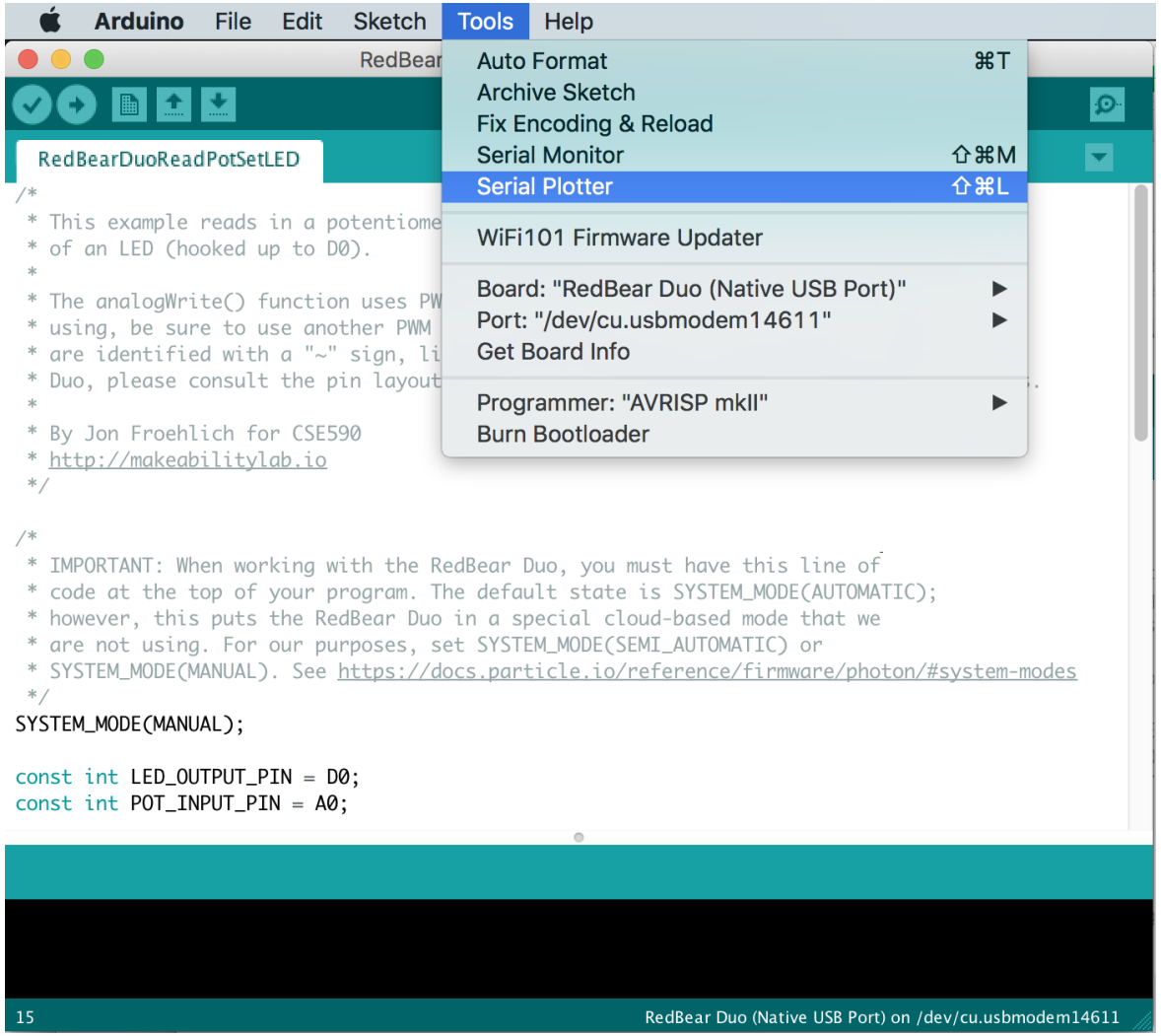
Modularize.

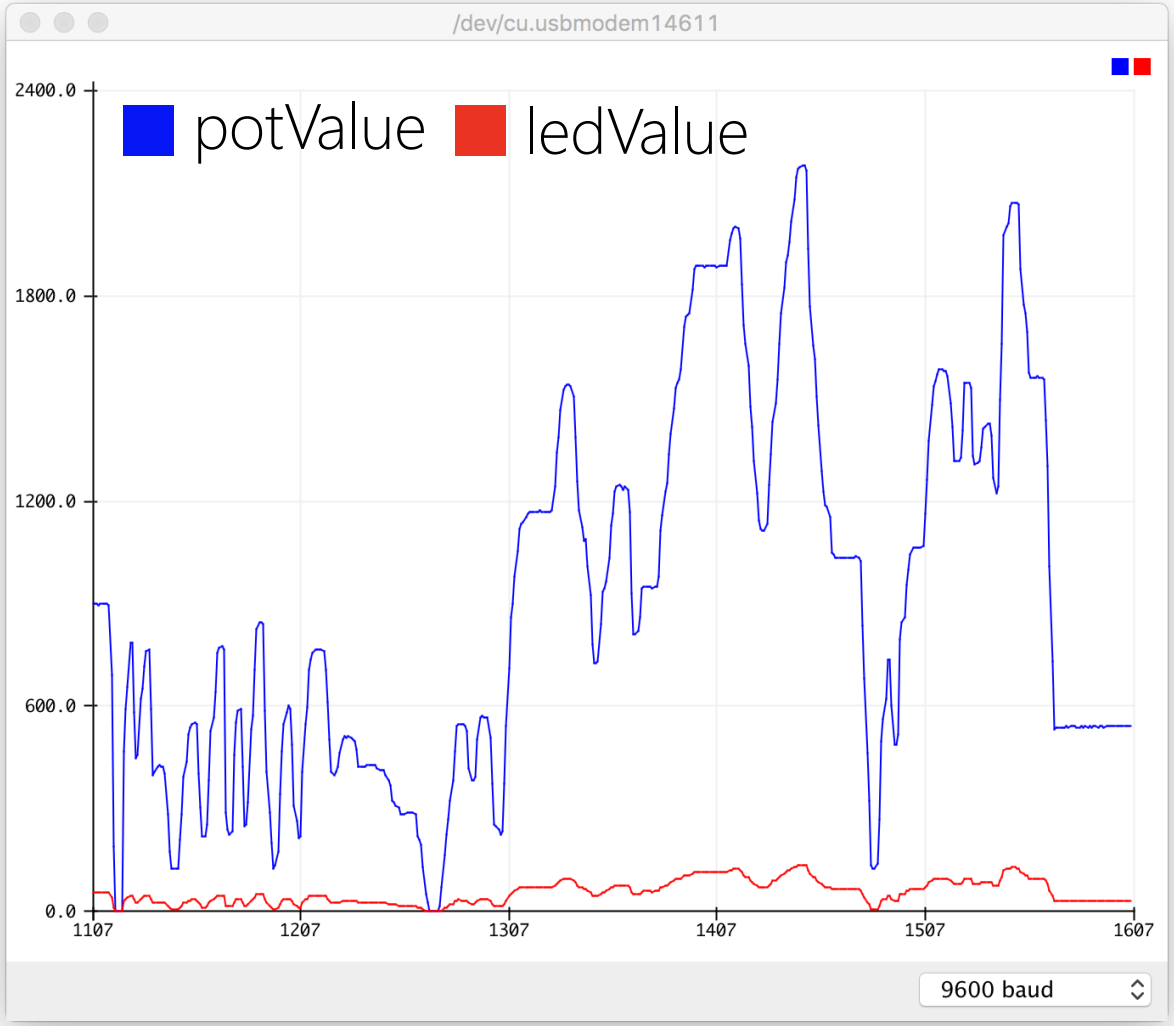
Use Serial.println -> Serial Monitor and Serial Plotter

Use a multimeter


int LED_OUTPUT_PIN = 3;
int MAX_BRIGHTNESS = 255; // the max digital out value on Uno, Leonardo,
_curBrightness = 0;   // how bright the LED is (between 0 - 255)
The setup function runs once when you press reset or power the board
setup() {
set the LED pin to as an output
inMode(LED_OUTPUT_PIN, OUTPUT);
erial.begin(9600); // for using Serial.println
The loop function runs over and over again forever
loop() {
// set the brightness of the LED pin
analogWrite(LED_OUTPUT_PIN, _curBrightness);
Serial.println(_curBrightness); // print out current brightness
// change the brightness for next time through the loop
_curBrightness = _curBrightness + 1;
// the maximum value that we can write out to analogWrite is 255
// so check to see if the current brightness value is greater than 255
// and if it is, reset the brightness to 0 (which is off)
if (_curBrightness > MAX_BRIGHTNESS){
_curBrightness = 0;
}
320.0
240.0
160.0
80.0
0.0
1654
1754
1854
1954
9600 baud
Kind
Folder
PDF Document
PNG image
Show All
delete
enter

USING THE SERIAL PLOTTER



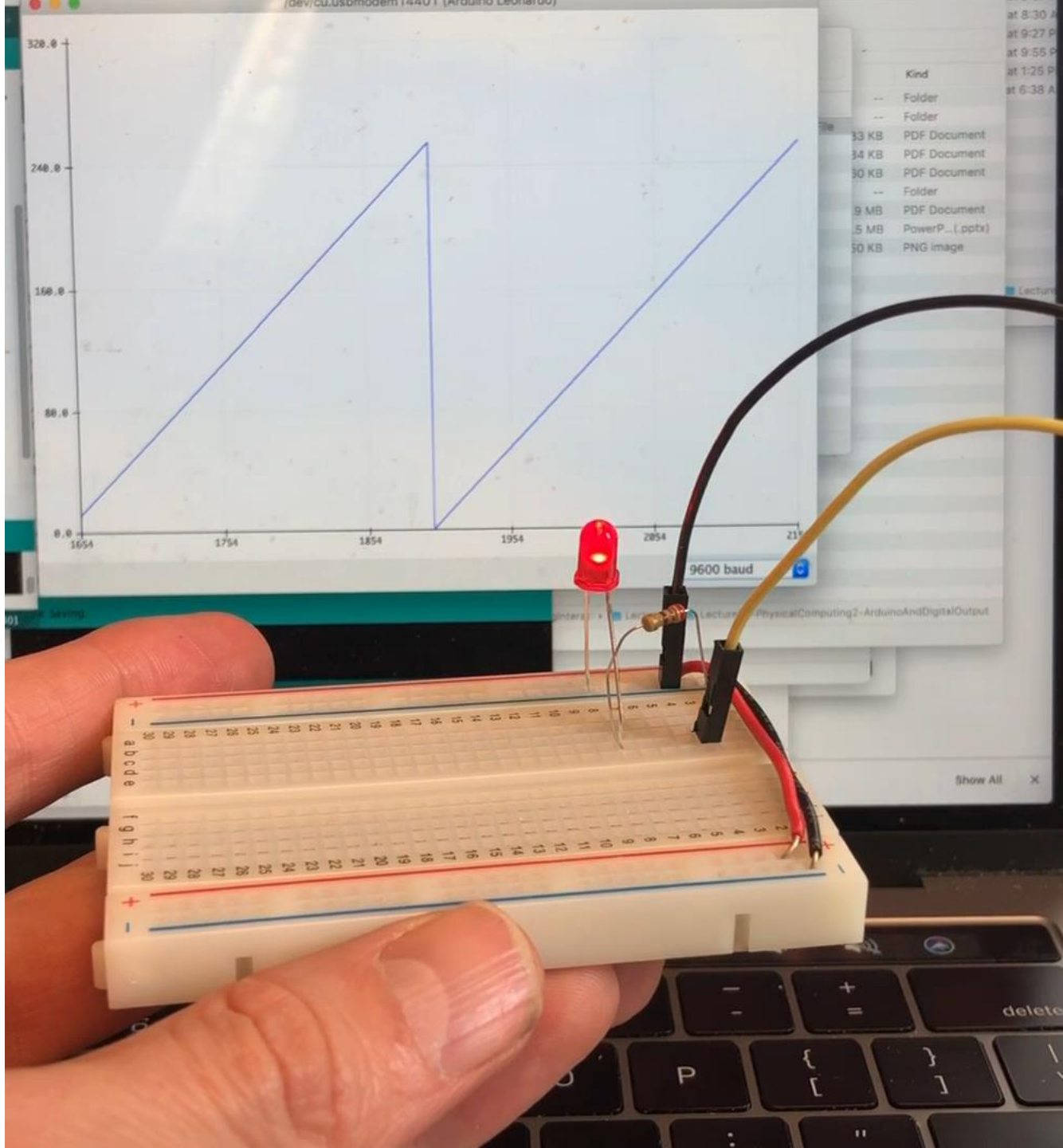


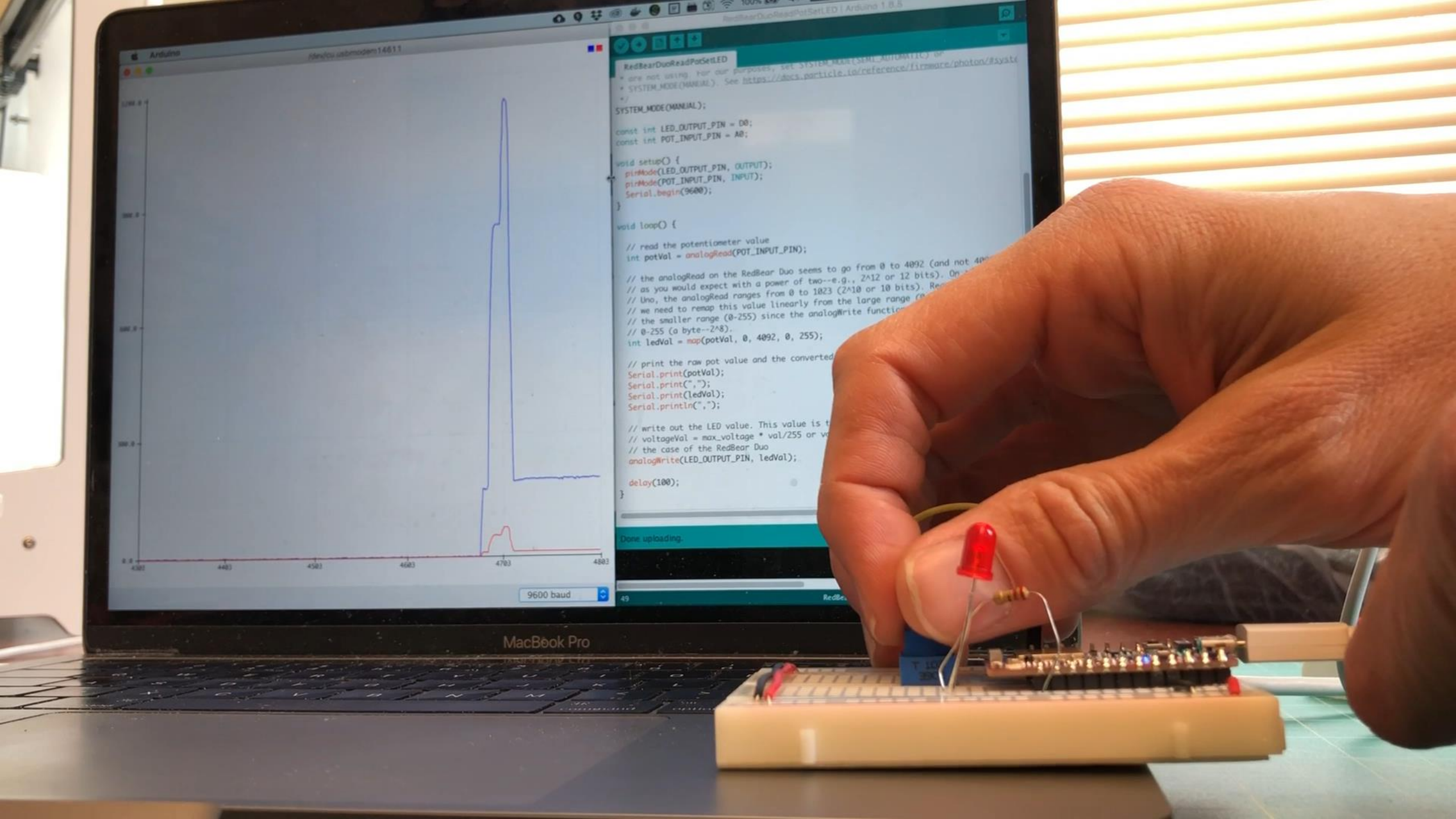
DEBUGGING

SERIAL PLOTTER

Visualizes numeric values off of Serial port

To **visualize multiple values**, use a **CSV** (comma separate value) format

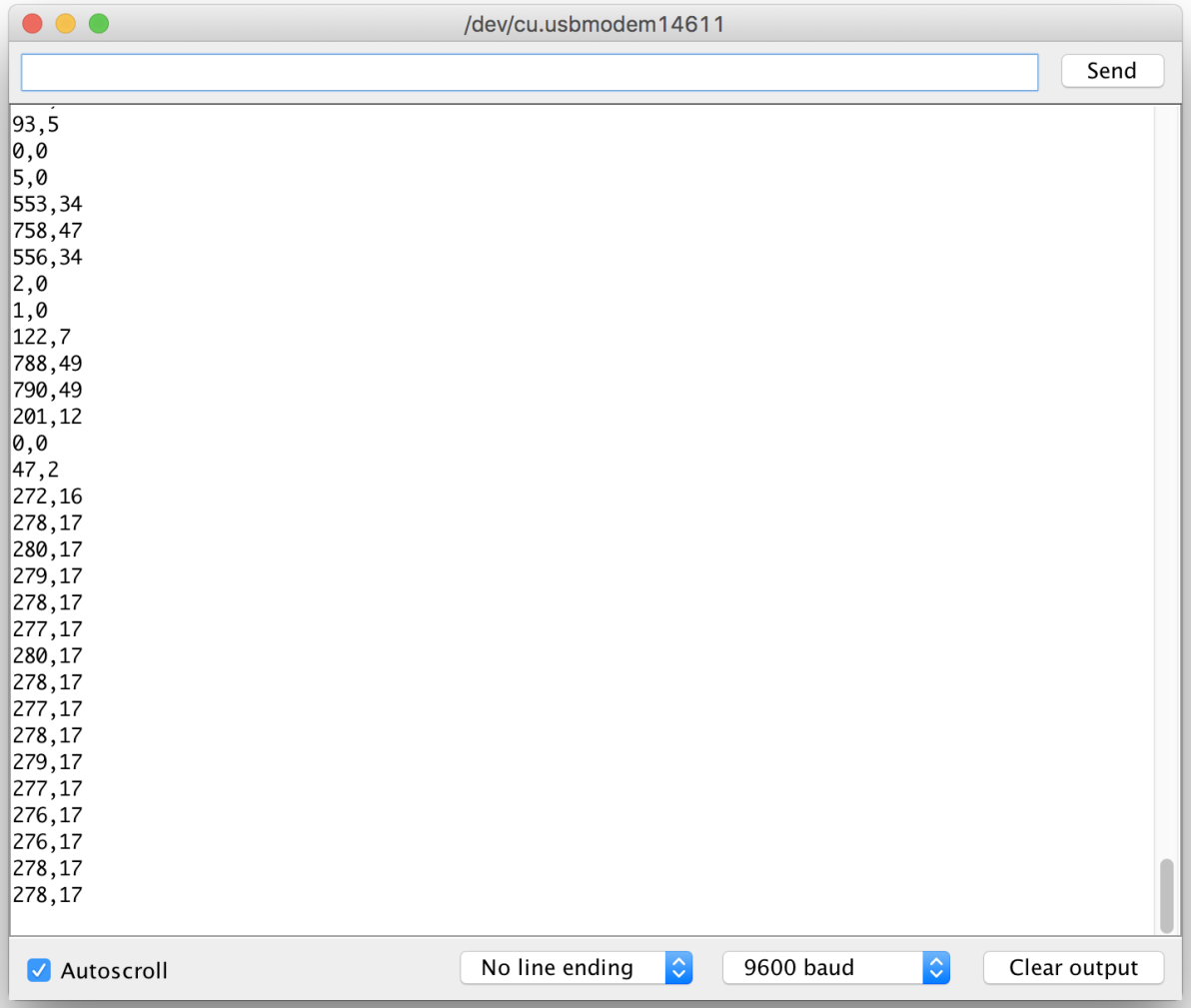


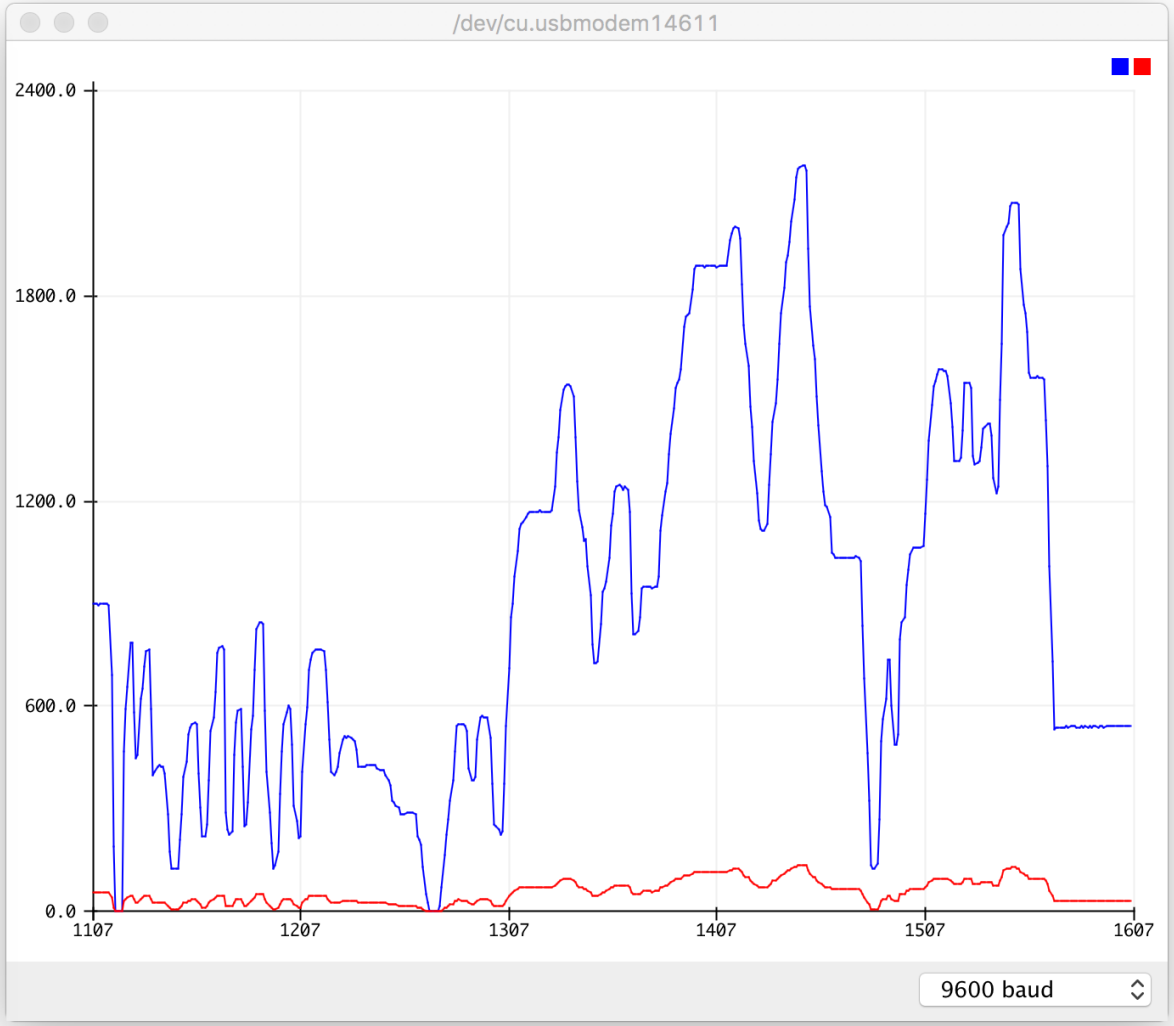

Arduino
/dev/cu.usbmodem14611
9600 baud
RedBearDuoReadPotSetLED
SYSTEM_MODE(MANUAL);
const int LED_OUTPUT_PIN = D0;
const int POT_INPUT_PIN = A0;
void setup() {
pinMode(LED_OUTPUT_PIN, OUTPUT);
pinMode(POT_INPUT_PIN, INPUT);
Serial.begin(9600);
}
void loop() {
// read the potentiometer value
int potVal = analogRead(POT_INPUT_PIN);
// print the raw pot value and the converted
Serial.print(potVal);
Serial.print(",");
Serial.print(ledVal);
Serial.println(",");
// write out the LED value. This value is t
// the case of the RedBear Duo
analogWrite(LED_OUTPUT_PIN, ledVal);
delay(100);
}
Done uploading.
MacBook Pro

SERIAL PLOTTER

SERIAL MONITOR VS. PLOTTER

You can only have one of these windows open at once.

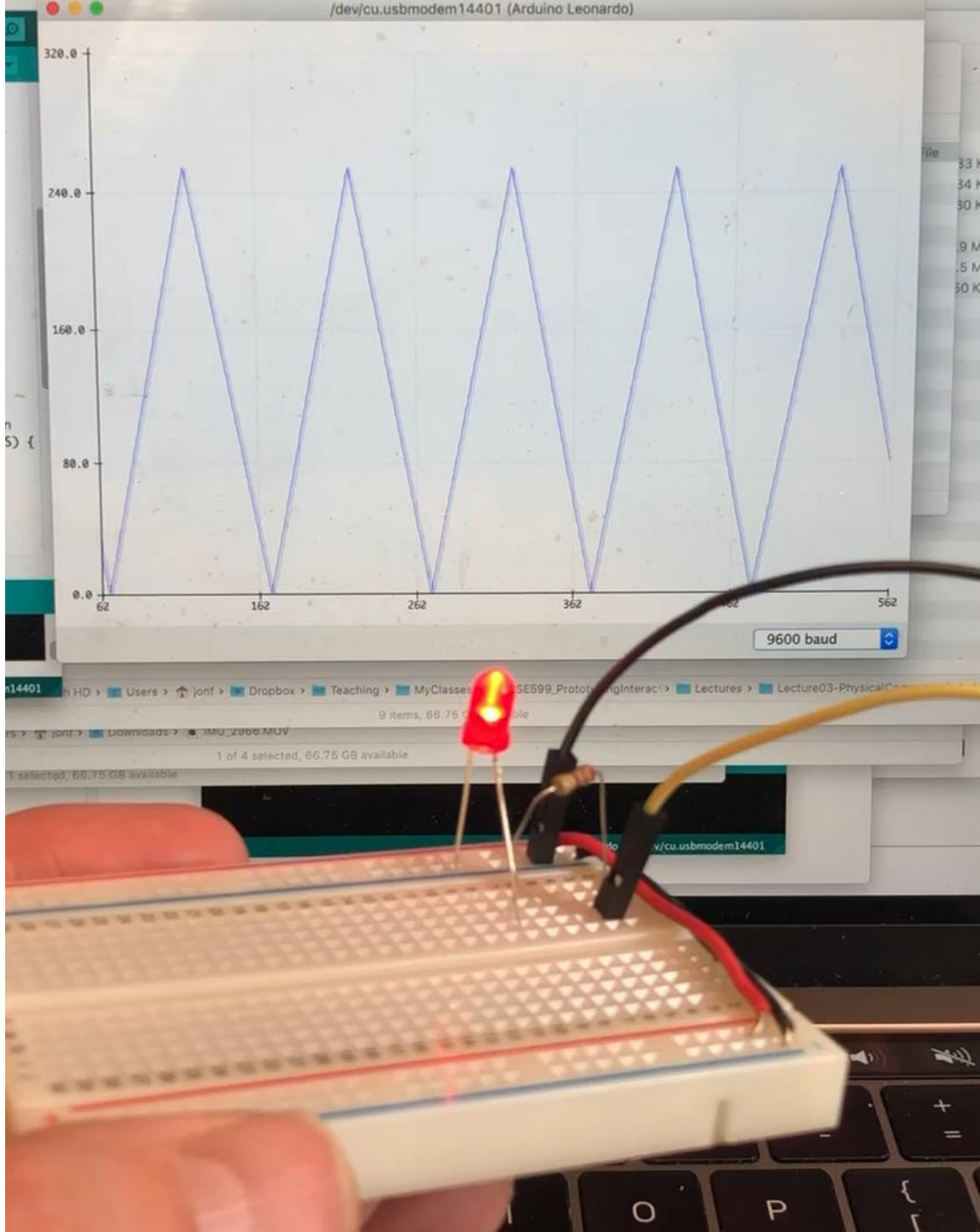




ANALOG OUTPUT

ACTIVITY FADE LED ON & OFF

Now fade the LED on **and** off

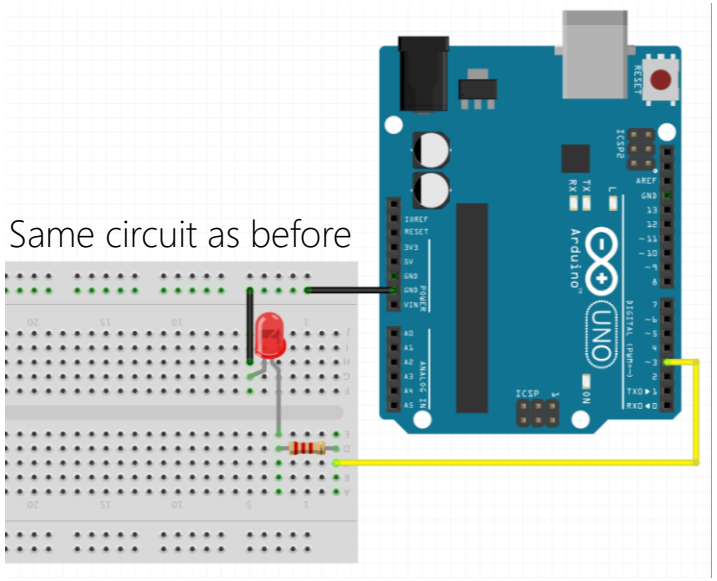


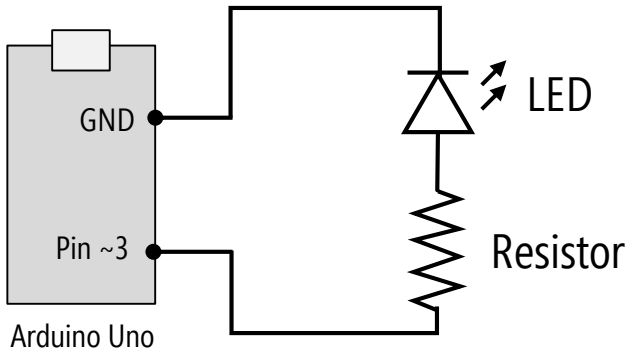
ANALOG OUTPUT

FADE LED ON AND FADE LED OFF

The same circuit as the one before but we need to update the code to fade on and off

Circuit: Fade LED on and off via the pin 3





Code: Fade LED on and off via pin 3

FadeOnAndOff

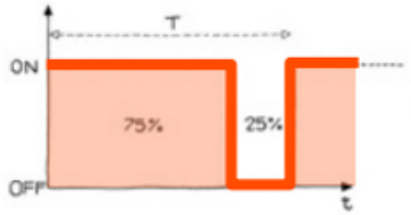
```
const int LED_OUTPUT_PIN = 3;
const int MIN_BRIGHTNESS = 0; // the min brightness
const int MAX_BRIGHTNESS = 255; // the max analog out value on Uno, Leonardo, etc. is 255 (8-bit max)

int _fadeAmount = 5;      // the amount to fade the LED by on each step
int _curBrightness = 0;   // how bright the LED is

// The setup function runs once when you press reset or power the board
void setup() {
  // set the LED pin to as an output
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  Serial.begin(9600); // for using Serial.println
}

// The loop function runs over and over again forever
void loop() {

  // set the brightness of the LED pin
  analogWrite(LED_OUTPUT_PIN, _curBrightness);
  Serial.println(_curBrightness); // print out current brightness

  // change the brightness for next time through the loop
  _curBrightness = _curBrightness + _fadeAmount;

  // reverse the direction of the fading at the end of each fade direction
  if (_curBrightness <= MIN_BRIGHTNESS || _curBrightness >= MAX_BRIGHTNESS) {
    _fadeAmount = -_fadeAmount; // reverses fade direction
  }

  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

HOW DOES ANALOG WRITE WORK?

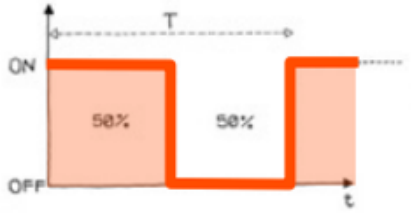
How does analogWrite work?

That is, how can the microcontroller control the voltage output?

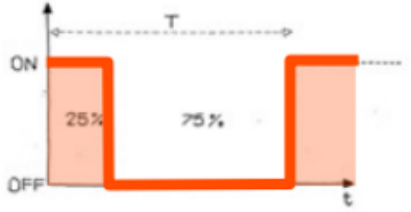
Using PWM!



analogWrite(11, 191) 75% brightness



analogWrite(11, 127) 50% brightness



analogWrite(11, 64) 25% brightness

analogWrite()

[Analog I/O]

Description

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to `analogWrite()`, the pin will generate a steady square wave of the specified duty cycle until the next call to `analogWrite()` (or a call to `digitalRead()` or `digitalWrite()`) on the same pin. The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz.

On most Arduino boards (those with the ATmega168 or ATmega328P), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 - 13 and 44 - 46. Older Arduino boards with an ATmega8 only support `analogWrite()` on pins 9, 10, and 11.
The Arduino DUE supports `analogWrite()` on pins 2 through 13, plus pins DAC0 and DAC1. Unlike the PWM pins, DAC0 and DAC1 are Digital to Analog converters, and act as true analog outputs.
You do not need to call `pinMode()` to set the pin as an output before calling `analogWrite()`.
The `analogWrite` function has nothing to do with the analog pins or the `analogRead` function.

Syntax

```
analogWrite(pin, value)
```

Parameters

`pin`: the pin to write to. Allowed data types: int.
`value`: the duty cycle: between 0 (always off) and 255 (always on). Allowed data types: int

[Analog I/O]

Description

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to `analogWrite()`, the pin will generate a steady square wave of the specified duty cycle until the next call to `analogWrite()` (or a call to `digitalRead()` or `digitalWrite()`) on the same pin. The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz.

On most Arduino boards (those with the ATmega168 or ATmega328P), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 - 13 and 44 - 46. Older Arduino boards with an ATmega8 only support `analogWrite()` on pins 9, 10, and 11.
The Arduino DUE supports `analogWrite()` on pins 2 through 13, plus pins DAC0 and DAC1. Unlike the PWM pins, DAC0 and DAC1 are Digital to Analog converters, and act as true analog outputs.
You do not need to call `pinMode()` to set the pin as an output before calling `analogWrite()`.
The `analogWrite` function has nothing to do with the analog pins or the `analogRead` function.

Syntax

```
analogWrite(pin, value)
```

[Analog I/O]

Description

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to `analogWrite()`, the pin will generate a steady square wave of the specified duty cycle until the next call to `analogWrite()` (or a call to `digitalRead()` or `digitalWrite()`) on the same pin. The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz.

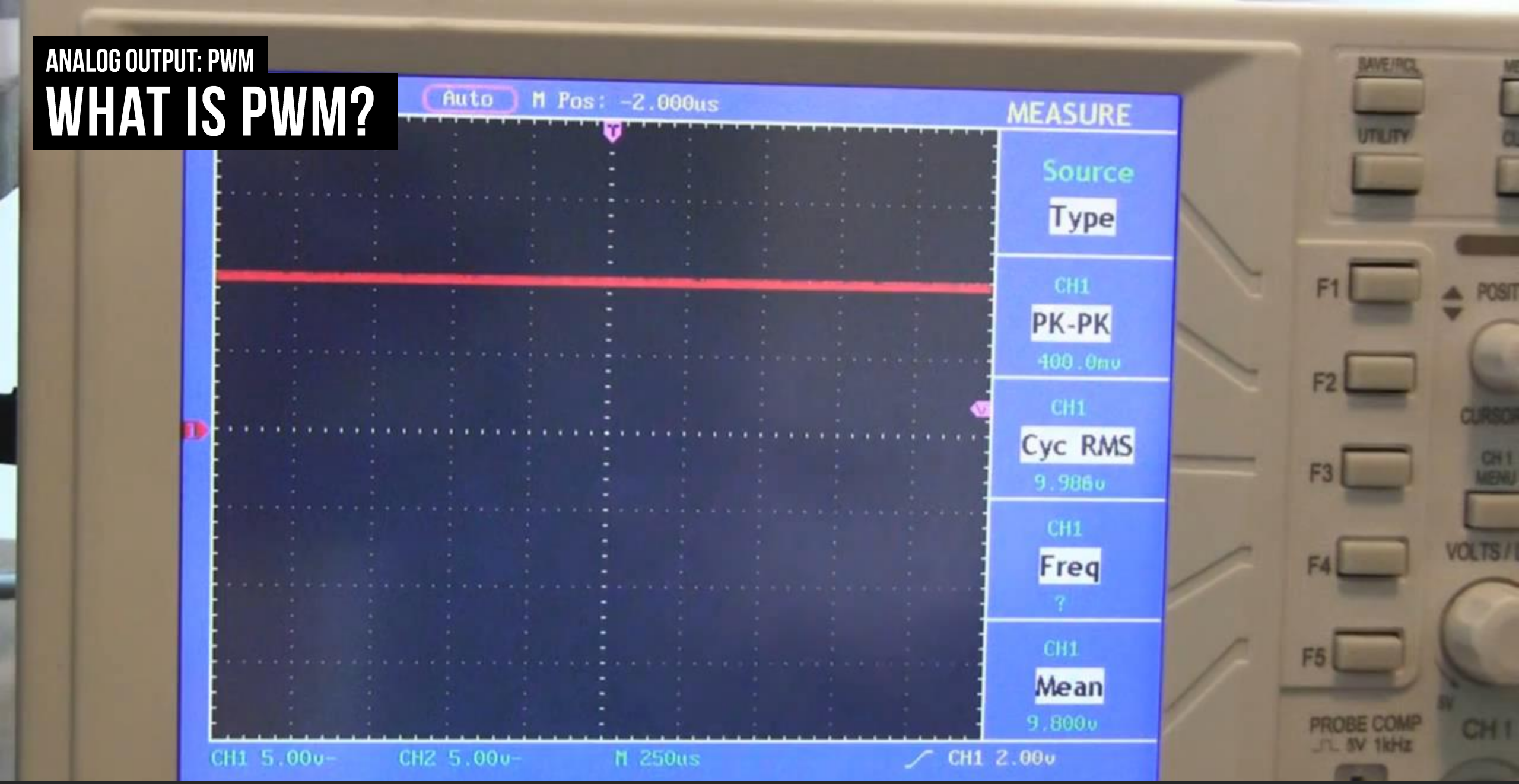
On most Arduino boards (those with the ATmega168 or ATmega328P), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 - 13 and 44 - 46. Older Arduino boards with an ATmega8 only support `analogWrite()` on pins 9, 10, and 11.
The Arduino DUE supports `analogWrite()` on pins 2 through 13, plus pins DAC0 and DAC1. Unlike the PWM pins, DAC0 and DAC1 are Digital to Analog converters, and act as true analog outputs.
You do not need to call `pinMode()` to set the pin as an output before calling `analogWrite()`.
The `analogWrite` function has nothing to do with the analog pins or the `analogRead` function.

Syntax

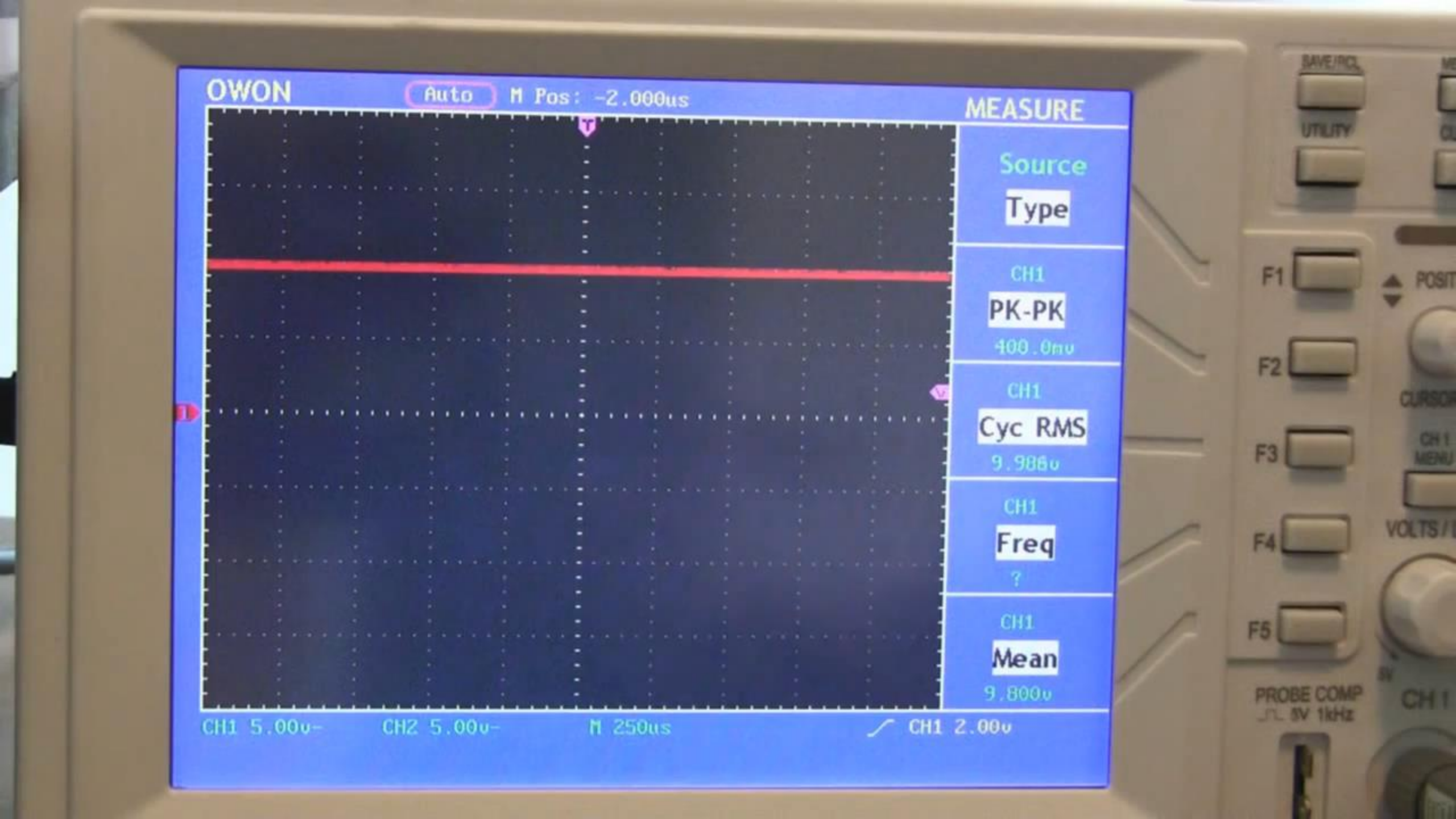
```
analogWrite(pin, value)
```

ANALOG OUTPUT: PWM

WHAT IS PWM?



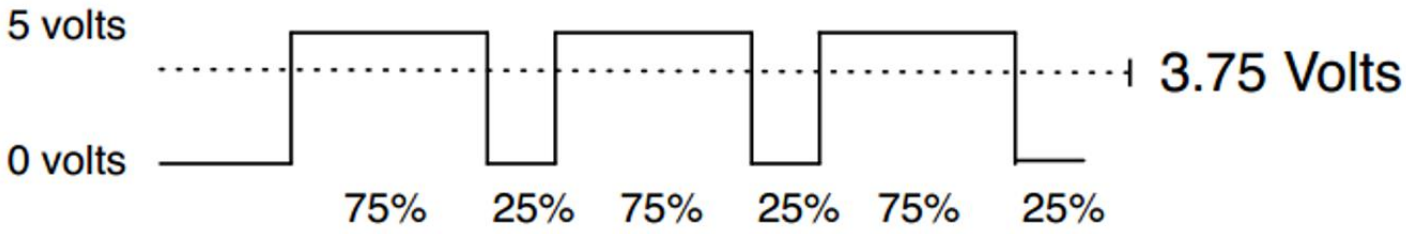
Source: Afrotechmods, https://youtu.be/YmPziPfaByw


OWON
Auto
M Pos: -2.000us
MEASURE
Source
Type
CH1
PK-PK
400.0mv
CH1
Cyc RMS
9.986v
CH1
Freq
?
CH1
Mean
9.800v
CH1 5.00v-
CH2 5.00v-
M 250us
CH1 2.00v
SAVE/RCL
UTILITY
F1
F2
F3
F4
F5
PROBE COMP
5V 1kHz
CH 1

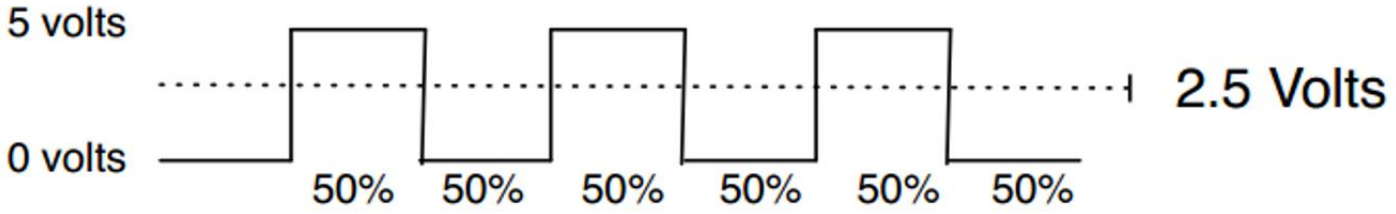
PULSE WIDTH MODULATION

The voltage is max_voltage * duty_cycle_val

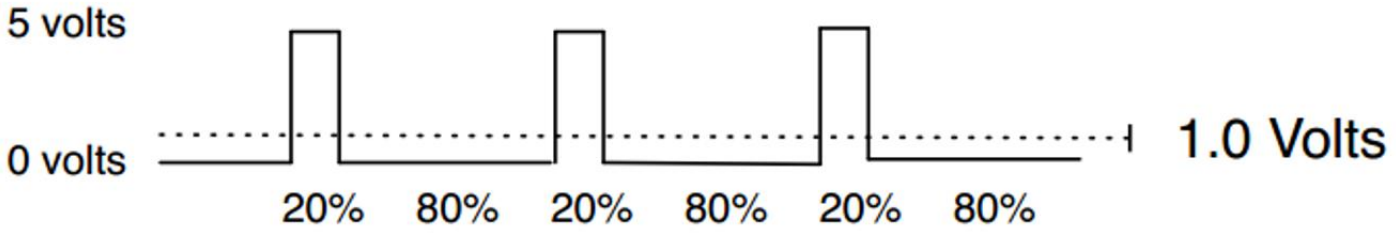
75% Duty Cycle
analogWrite(191)



50% Duty Cycle
analogWrite(127)

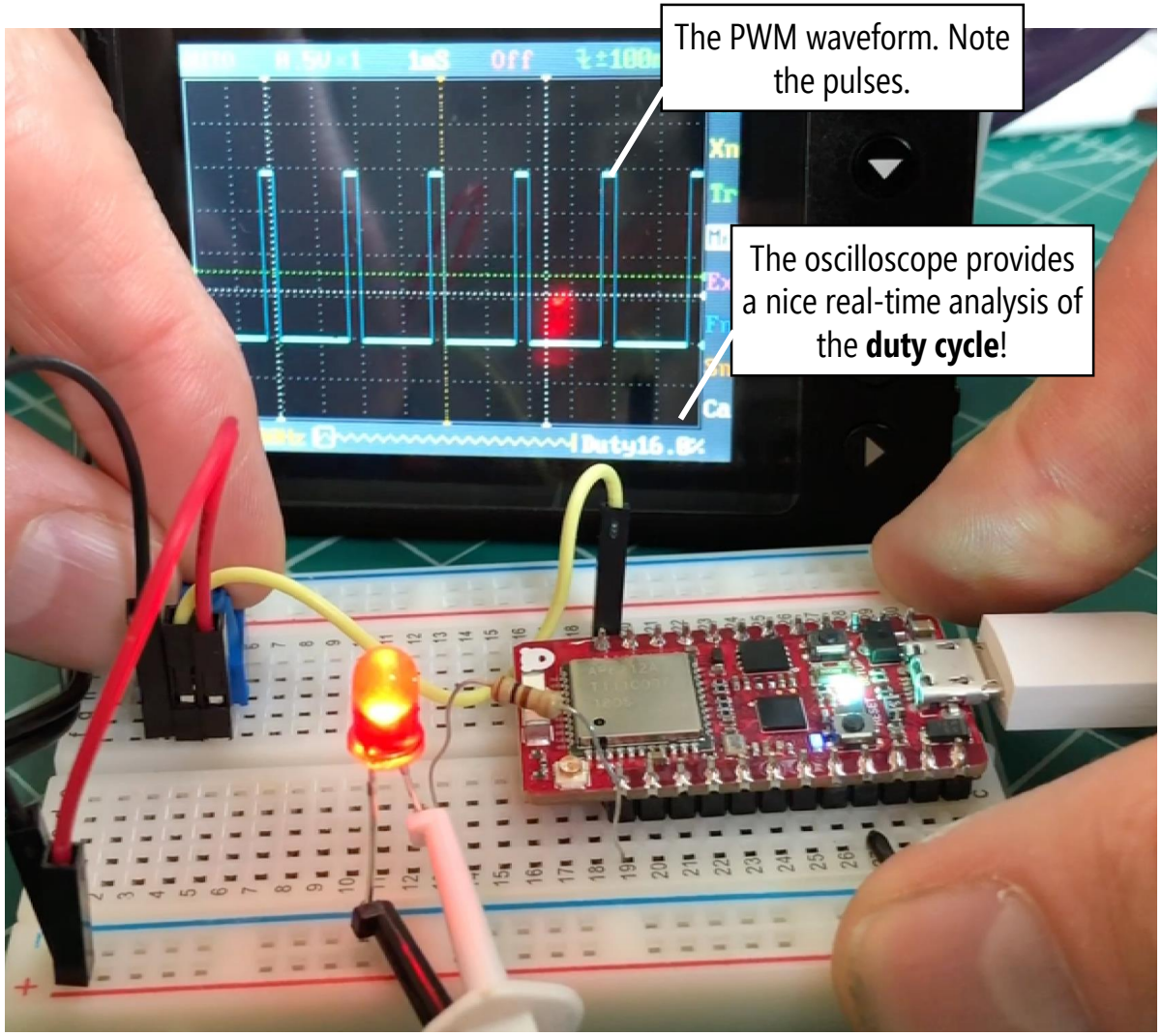


20% Duty Cycle
analogWrite(51)



Source: https://www.arduino.cc/en/Tutorial/PWM

VISUALIZING ARDUINO PWM USING AN OSCILLISCOPE

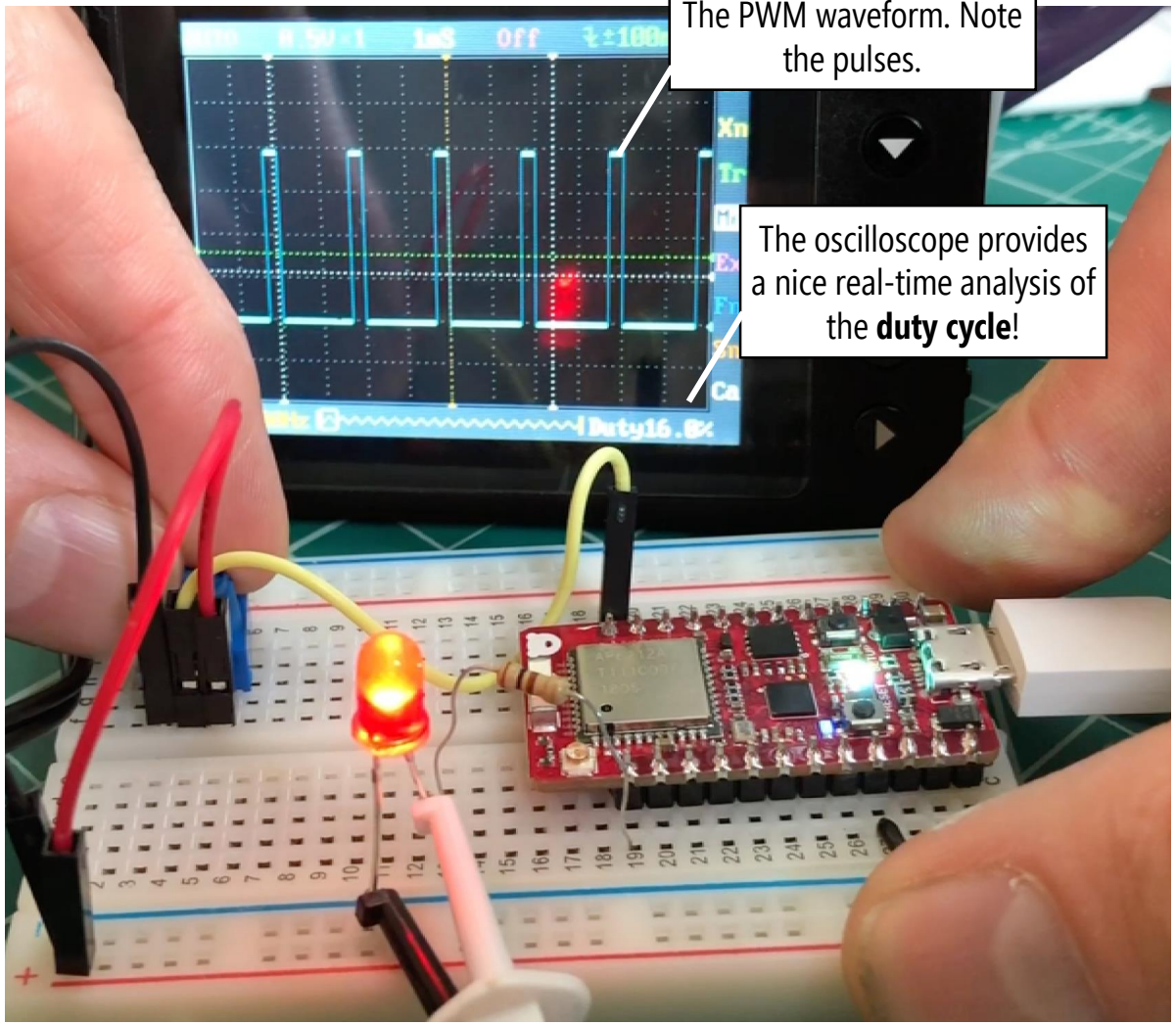


RedBearDuoReadPotSetLED | Arduino 1.8.5

RedBearDuoReadPotSetLED

```
SYSTEM_MODE(MANUAL);

const int LED_OUTPUT_PIN = D0;
const int POT_INPUT_PIN = A0;

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(POT_INPUT_PIN, INPUT);
}

void loop() {

  // read the potentiometer value
  int potVal = analogRead(POT_INPUT_PIN);

  // the analogRead on the RedBear Duo seems to go from 0 to 4092 (and not 4095
  // as you would expect with a power of two--e.g., 2^12 or 12 bits). On the Arduino
  // Uno, the analogRead ranges from 0 to 1023 (2^10 or 10 bits). Regardless,
  // we need to remap this value linearly from the large range (0-4092) to
  // the smaller range (0-255) since the analogWrite function can only write out
  // 0-255 (a byte--2^8).
  int ledVal = map(potVal, 0, 4092, 0, 255);

  // write out the LED value. This value is translated to voltage by:
  // voltageVal = max_voltage * val/255 or voltageVal = 3.3V * val/255 in
  // the case of the RedBear Duo
  analogWrite(LED_OUTPUT_PIN, ledVal);

  delay(100);
}
```

VISUALIZING ARDUINO PWM USING AN OSCILLISCOPE



RedBearDuoReadPotSetLED | Arduino 1.8.5

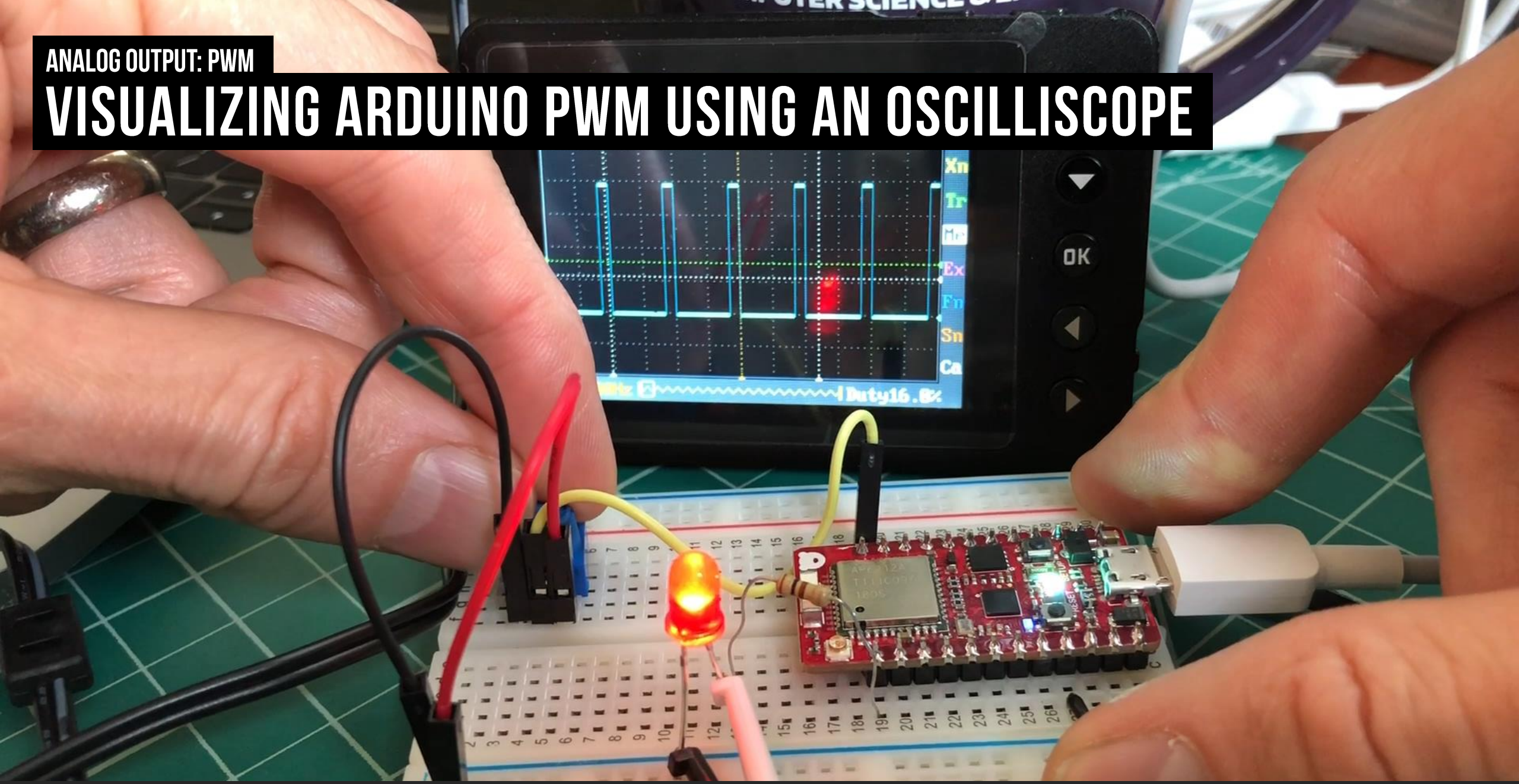
RedBearDuoReadPotSetLED

```
SYSTEM_MODE(MANUAL);

const int LED_OUTPUT_PIN = D0;
const int POT_INPUT_PIN = A0;

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(POT_INPUT_PIN, INPUT);
}

void loop() {

  // read the potentiometer value
  int potVal = analogRead(POT_INPUT_PIN);

  // the analogRead on the RedBear Duo seems to go from 0 to 4092 (and not 4095
  // as you would expect with a power of two--e.g., 2^12 or 12 bits). On the Arduino
  // Uno, the analogRead ranges from 0 to 1023 (2^10 or 10 bits). Regardless,
  // we need t                                                    092) to
  // the small                                                    ly write out
  // 0-255 (a
  int ledVal =

  // write ou                                                     :
  // voltageVal = max_voltage * val/255 or voltageVal = 3.3V * val/255 in
  // the case of the RedBear Duo
  analogWrite(LED_OUTPUT_PIN, ledVal);

  delay(100);
}
```

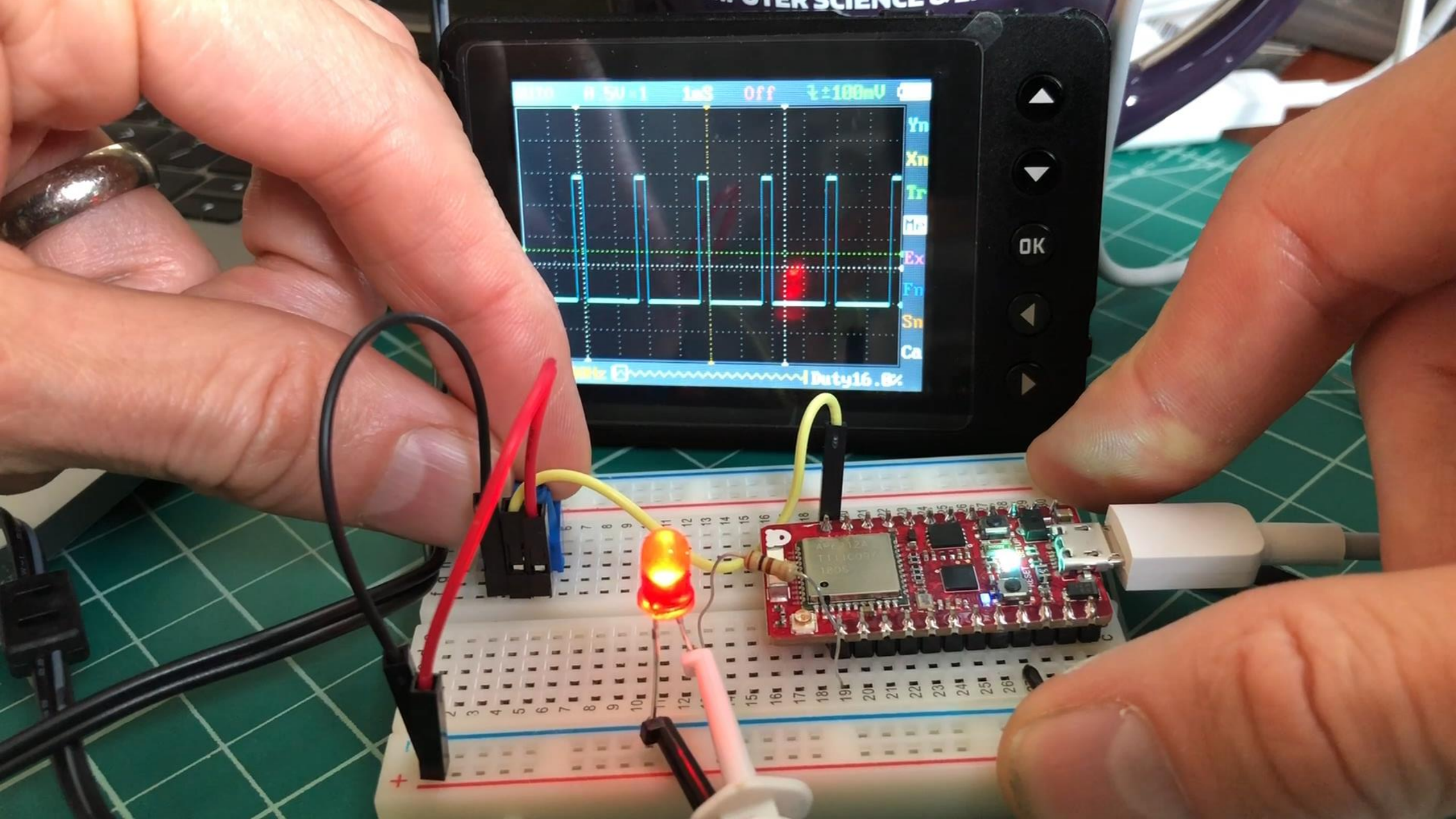
The **analogWrite call sets the duty cycle** of the PWM waveform. Note: you cannot set the PWM frequency. See the video on the next slide

ANALOG OUTPUT: PWM

VISUALIZING ARDUINO PWM USING AN OSCILLISCOPE



Source: video by Jon Froehlich


OK

PULSE WIDTH MODULATION

ANOTHER GREAT VIDEO ON PWM, DUTY CYCLES, & FREQUENCY

Source: https://youtu.be/rBQVfCUuhfs

ANALOG OUTPUT

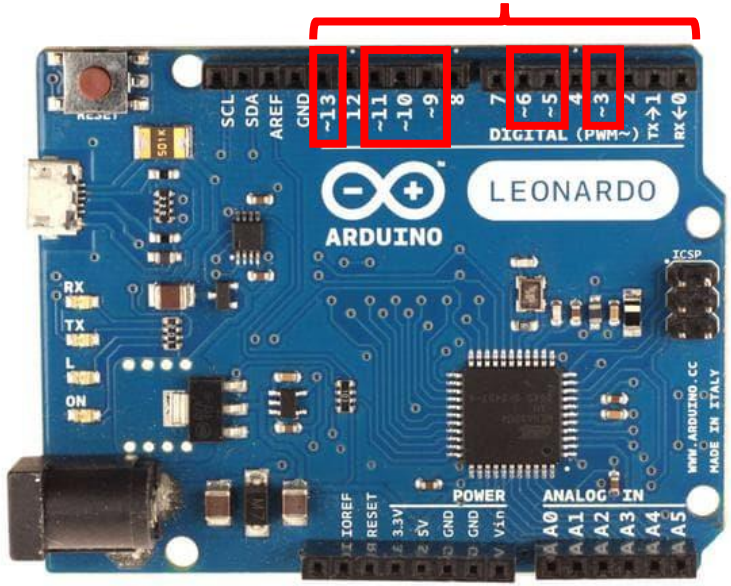
SUMMARY

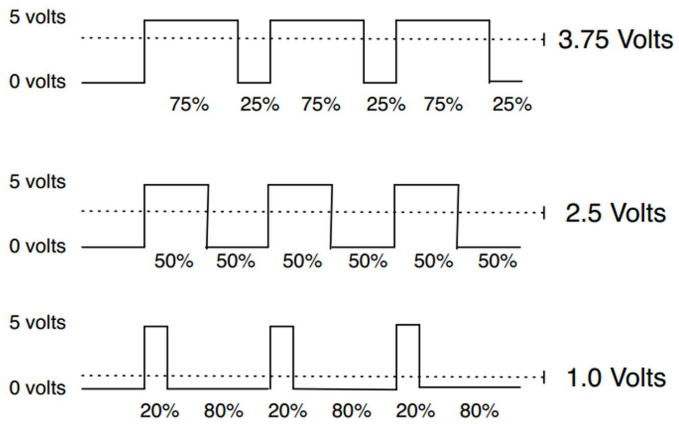
The **microcontroller** uses **PWM** to **vary** the **output voltage**

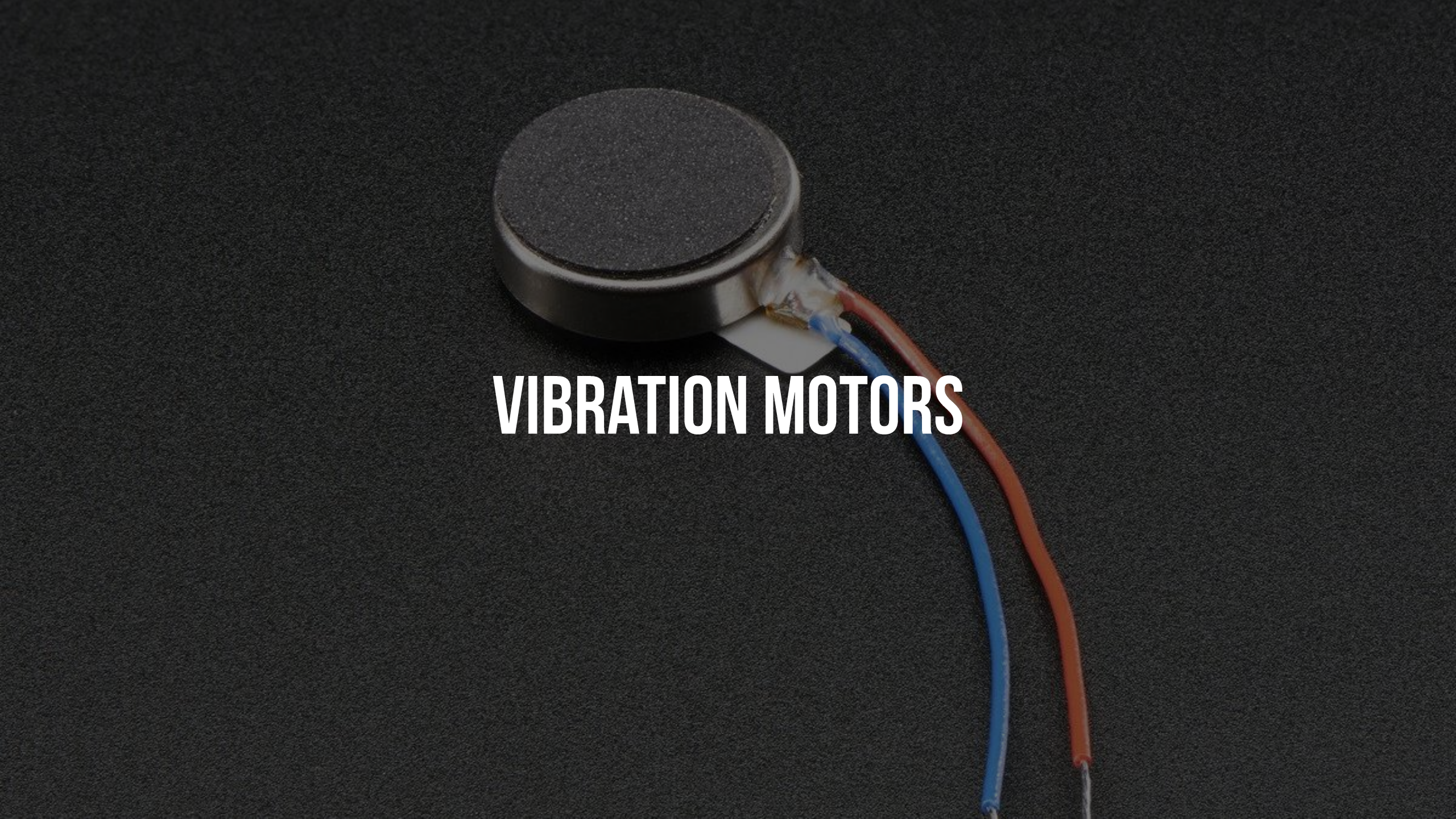
analogWrite changes the **duty cycle** but the waveform frequency is fixed at 490Hz or 980Hz depending on the pin

The analog **output voltage** is equal to **max_voltage * duty_cycle_val**

7 analog output pins: pins with the '~' symbol can be used to simulate analog output using 8-bit pulse-width modulation (PWM). Pins 3, 9, 10, 11, & 13 have PWM frequency of ~490Hz, Pins 5 & 6 are ~980Hz. Max current is 40mA per pin

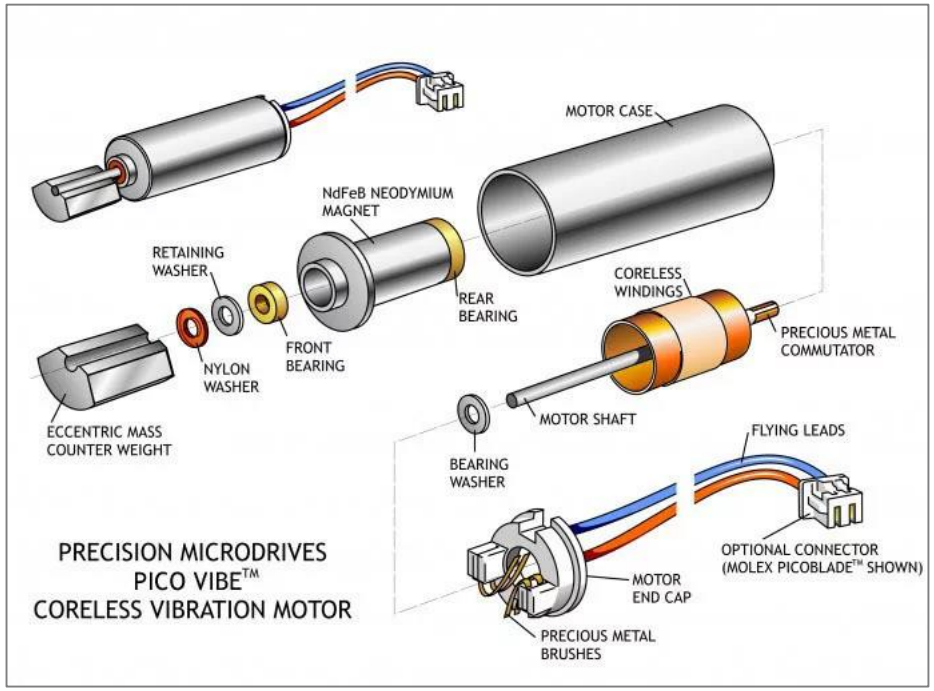





VIBRATION MOTORS

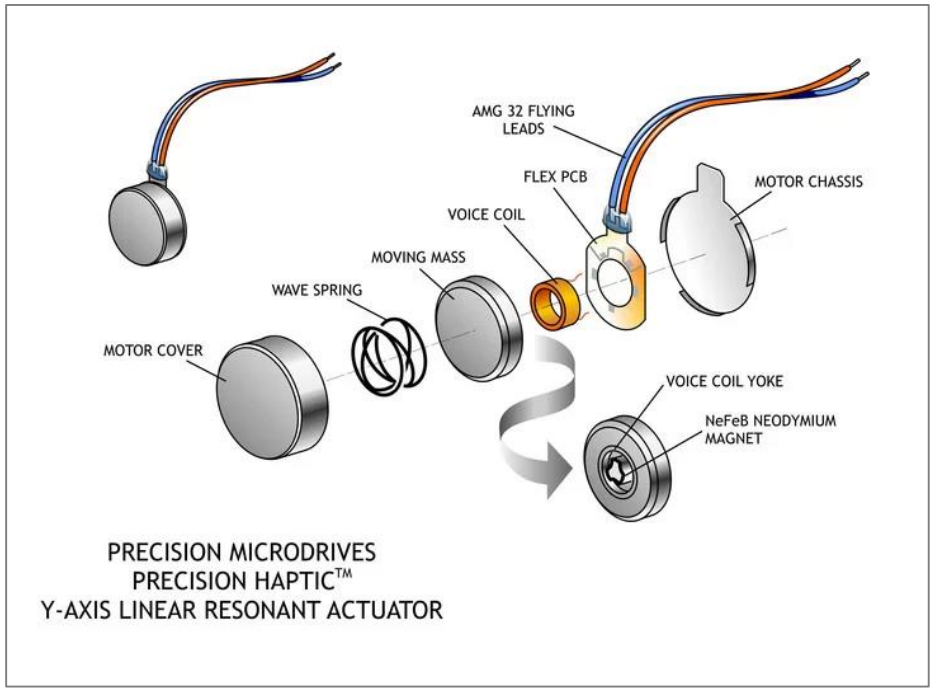
VIBRATION MOTORS

TWO PRIMARY TYPES: ERM AND LRA



Eccentric Rotating Mass (ERM) Motors

Based on simple DC motor
Cheap and easy to manufacture
Frequency and amplitude **coupled** together. Increased voltage leads to increase in both vibration frequency and amplitude



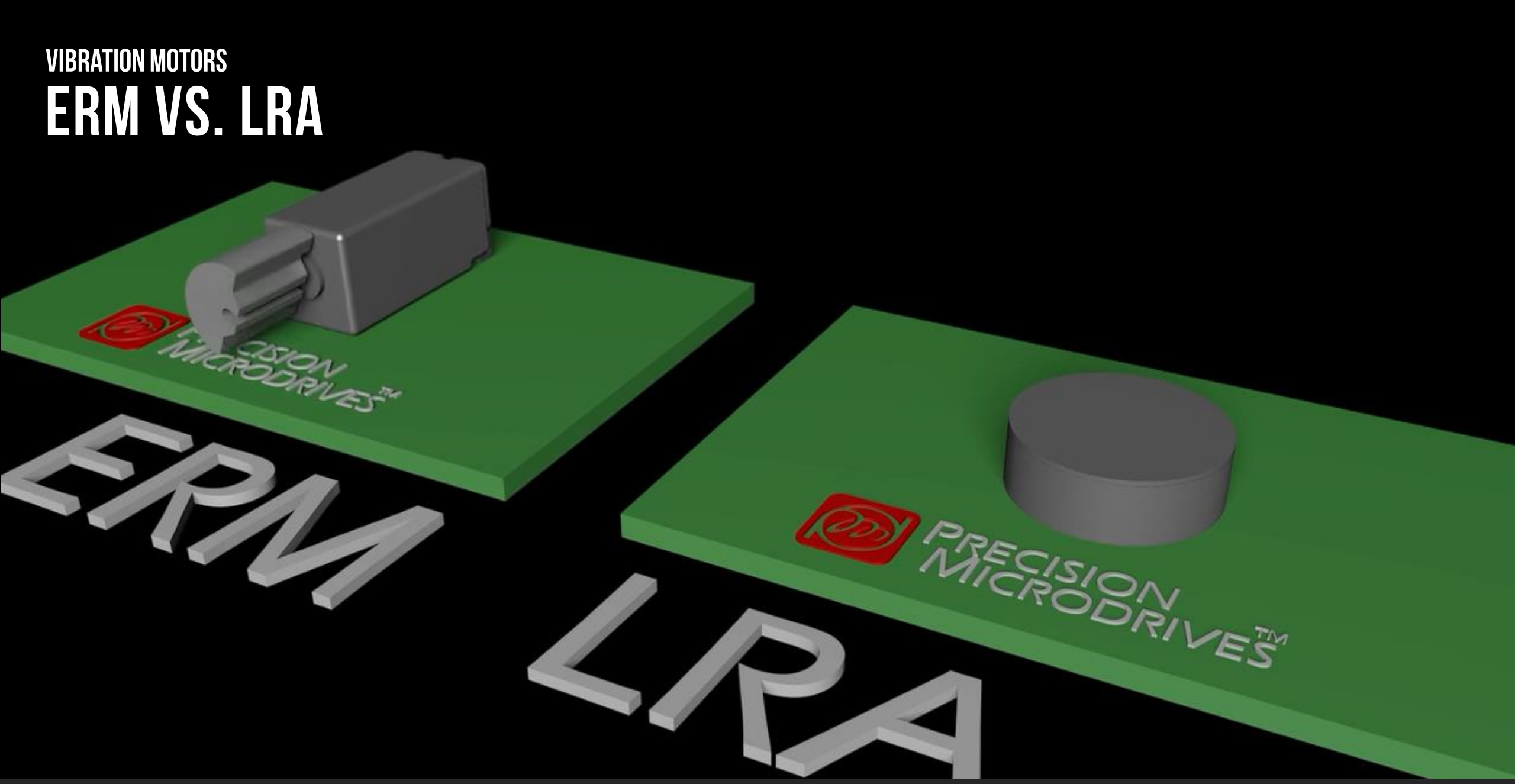
Linear Resonant Actuator (LRA) Motors

Requires AC signal (like a loudspeaker)
Uses moving mass with a spring and voice coil
Frequency and amplitude of vibration are **decoupled**

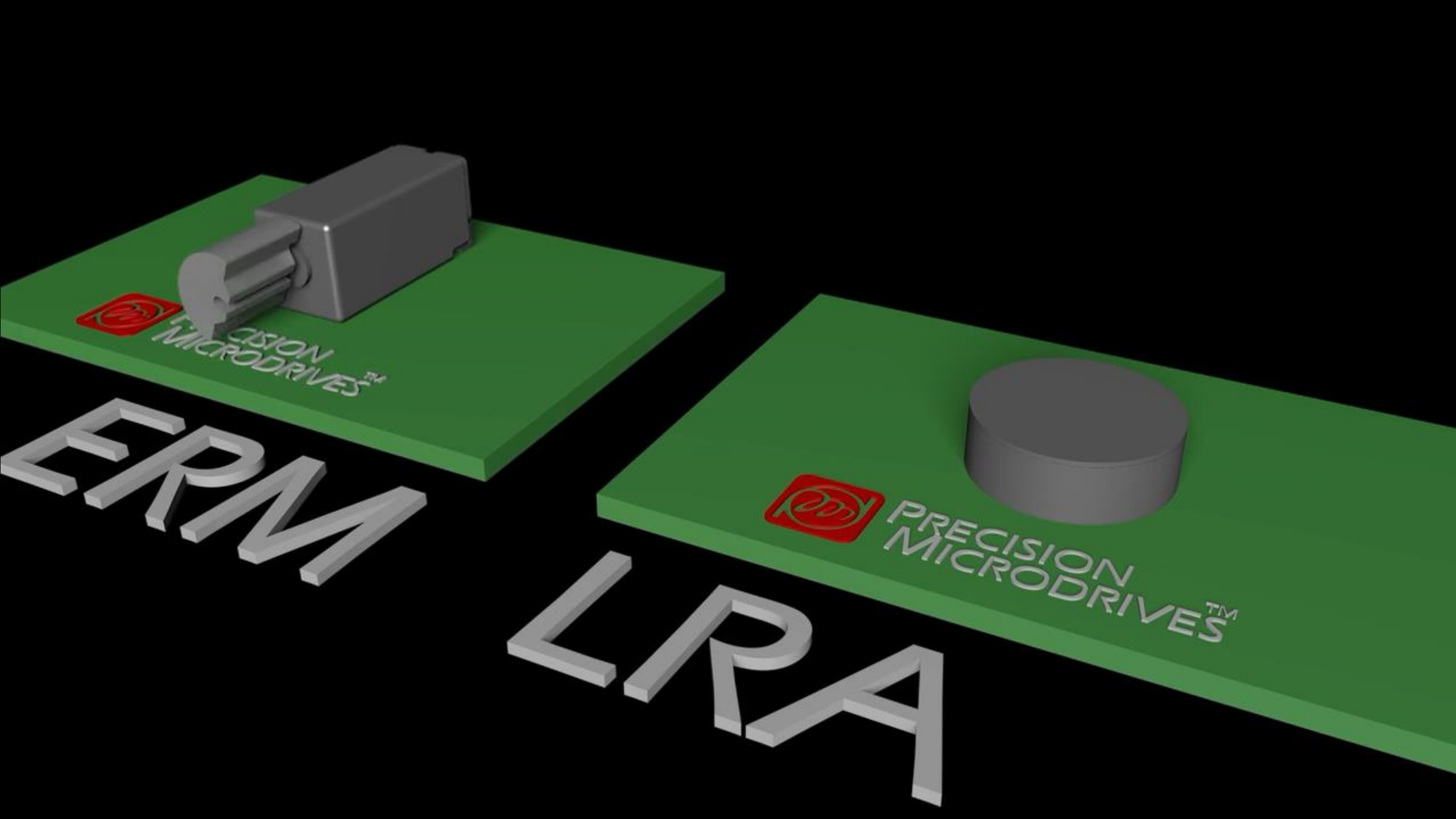
https://www.precisionmicrodrives.com/vibration-motors/vibration-motors-erms-and-lras/

VIBRATION MOTORS

ERM VS. LRA

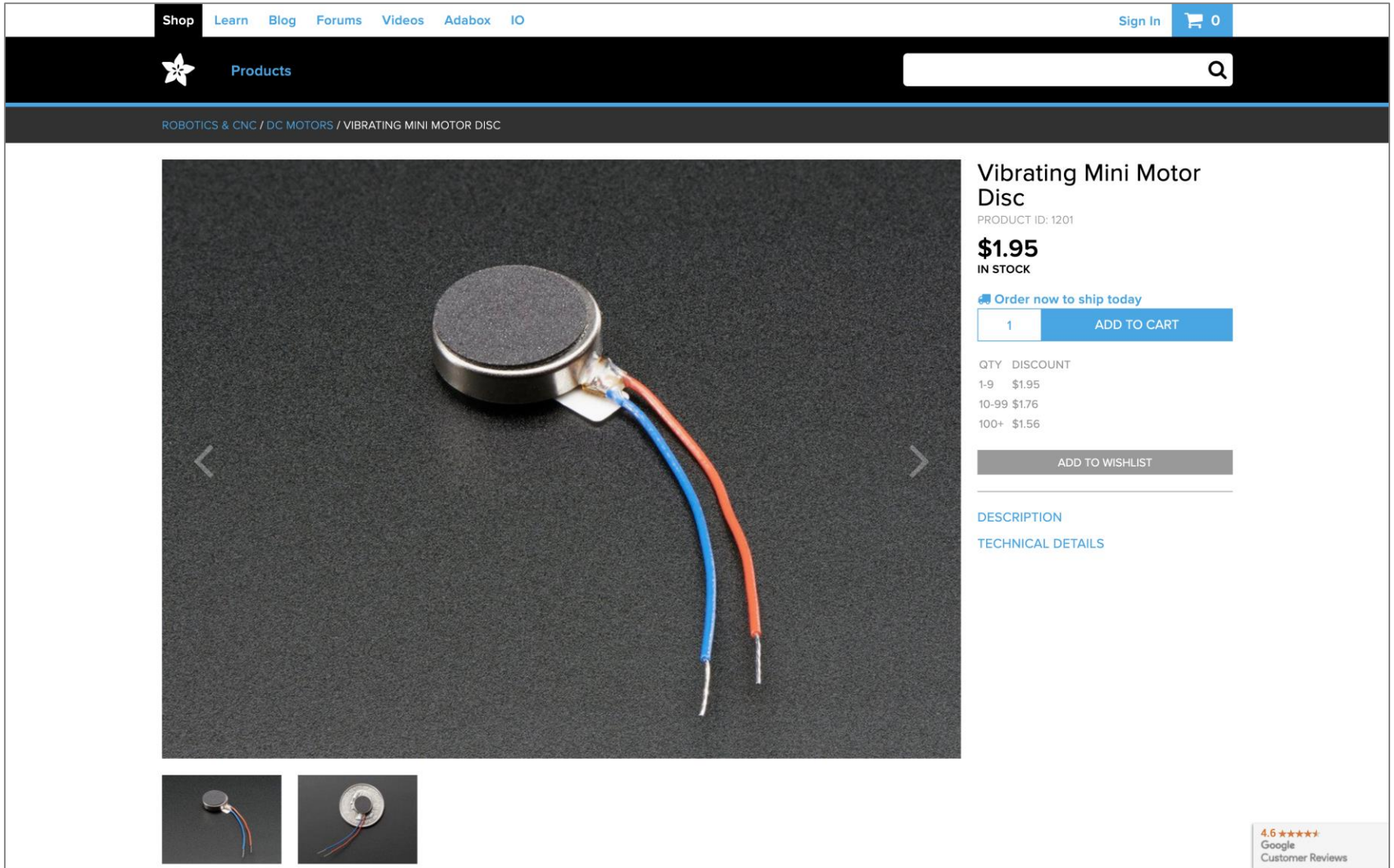


Source: https://www.precisionmicrodrives.com/vibration-motors/linear-resonant-actuators-lras/


PRECISION MICRODRIVES™
ERM
PRECISION MICRODRIVES™
LRA

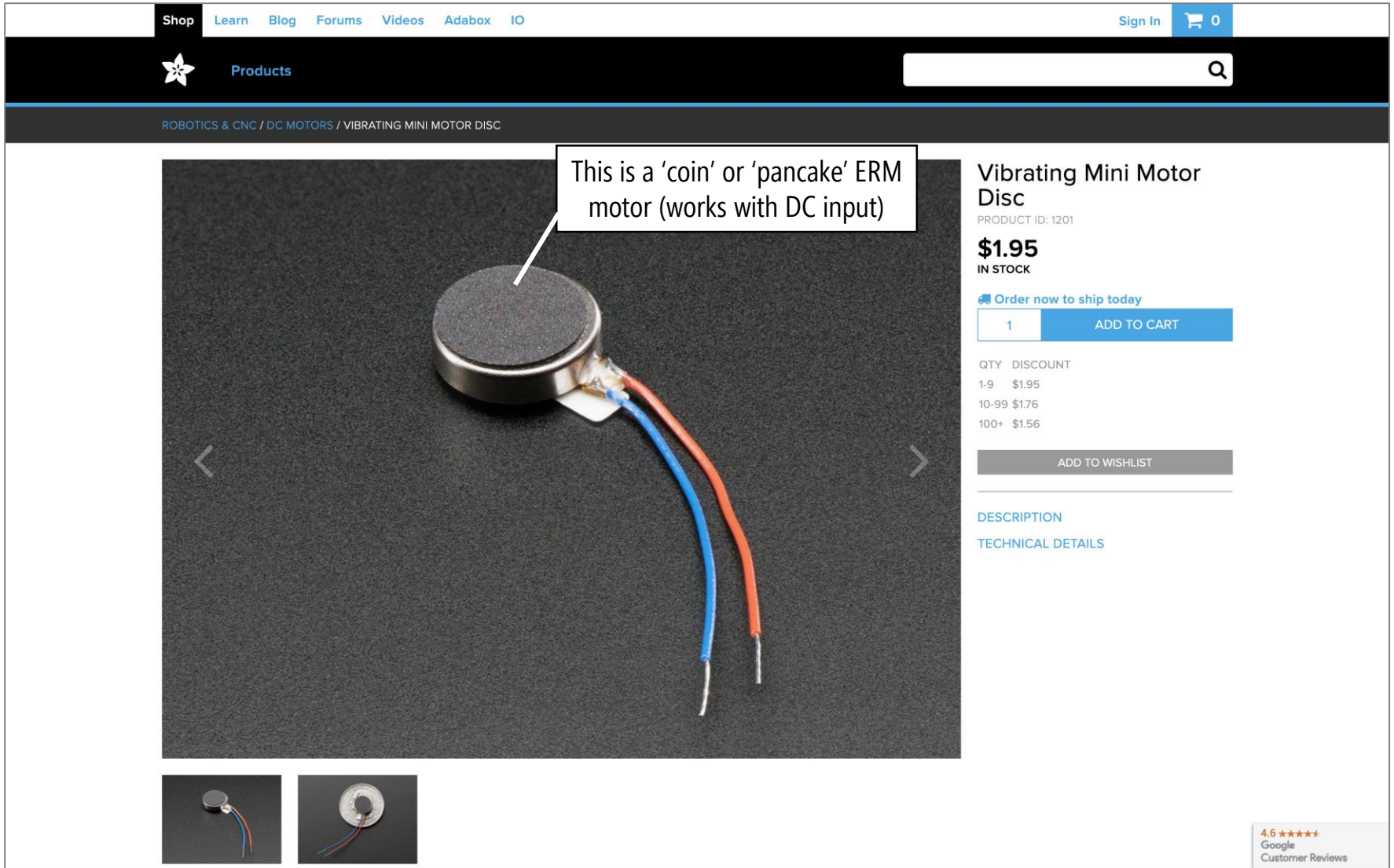
VIBRATION MOTORS

WHAT KIND OF MOTOR DO YOU HAVE IN YOUR KITS?



VIBRATION MOTORS

WHAT KIND OF MOTOR DO YOU HAVE IN YOUR KITS?



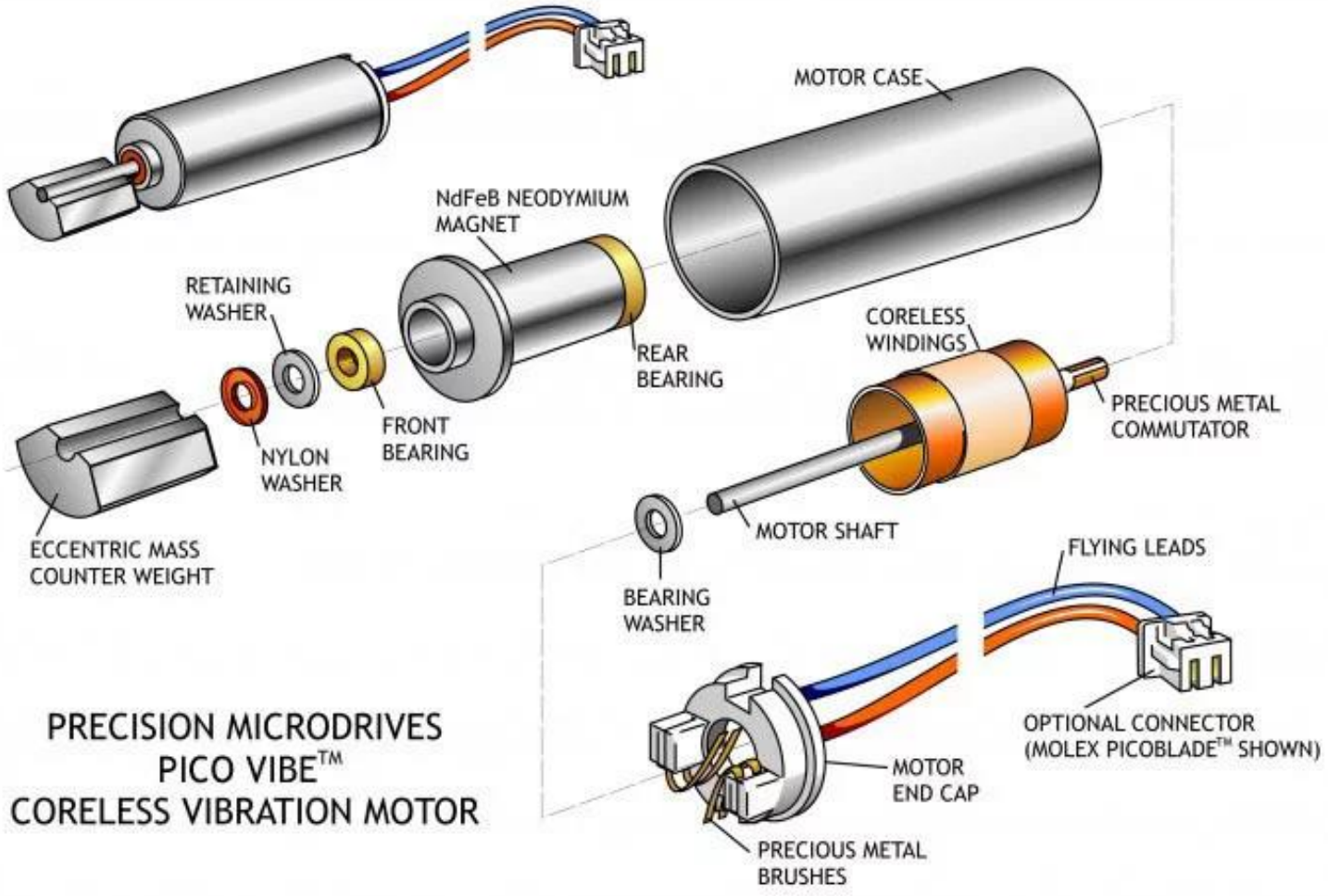
VIBRATION MOTORS

ERM FORM FACTORS

Cylindrical or 'Pager' Motor

Most popular form factor

Lots of DC motors available in a cylindrical form



https://www.precisionmicrodrives.com/vibration-motors/vibration-motors-erms-and-lras/

VIBRATION MOTORS

ERM FORM FACTORS

Cylindrical or ‘Pager’ Motor

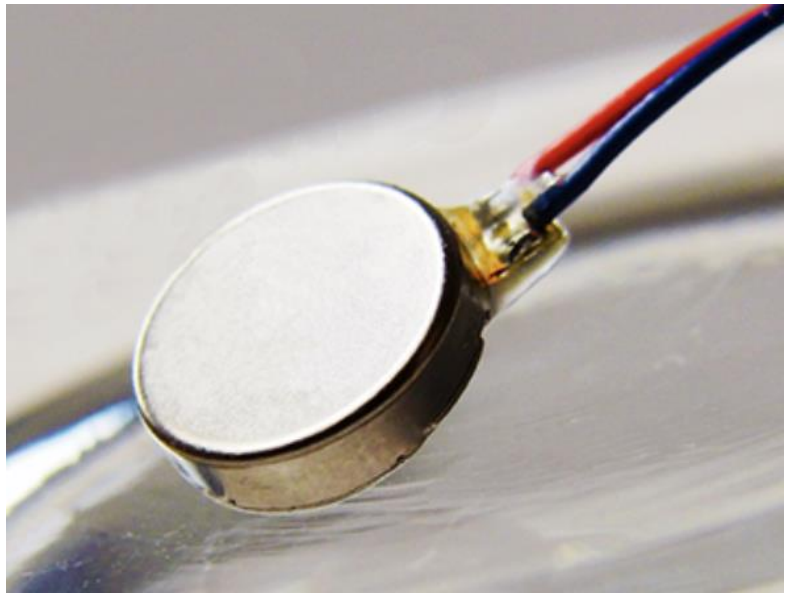
Most popular form factor

Lots of DC motors available in a cylindrical form

Encapsulated Cylindrical

Seals the ERM in plastic or composite housing

Important for waterproofing applications



Coin or ‘Pancake’ Motor

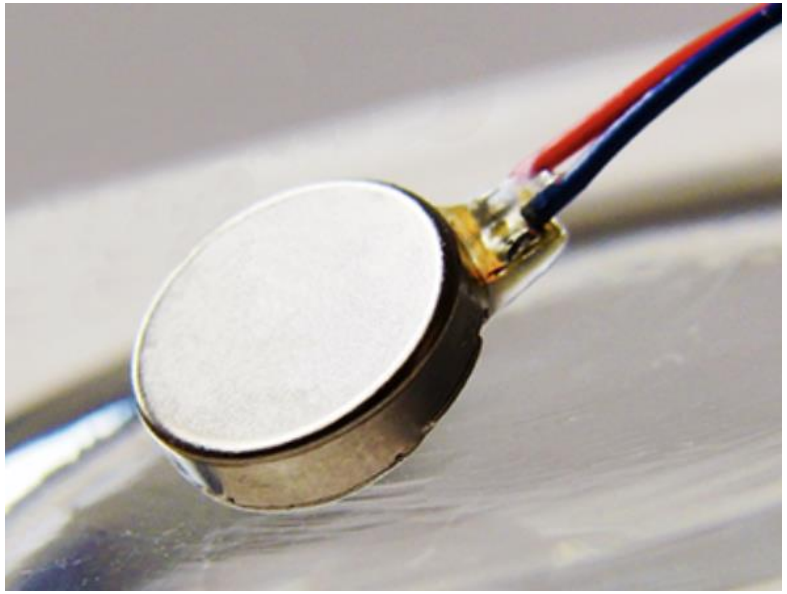
Same operating principle as the cylindrical

But eccentric mass is kept inside small ‘coin’ body

https://www.precisionmicrodrives.com/vibration-motors/vibration-motors-erms-and-lras/

VIBRATION MOTORS

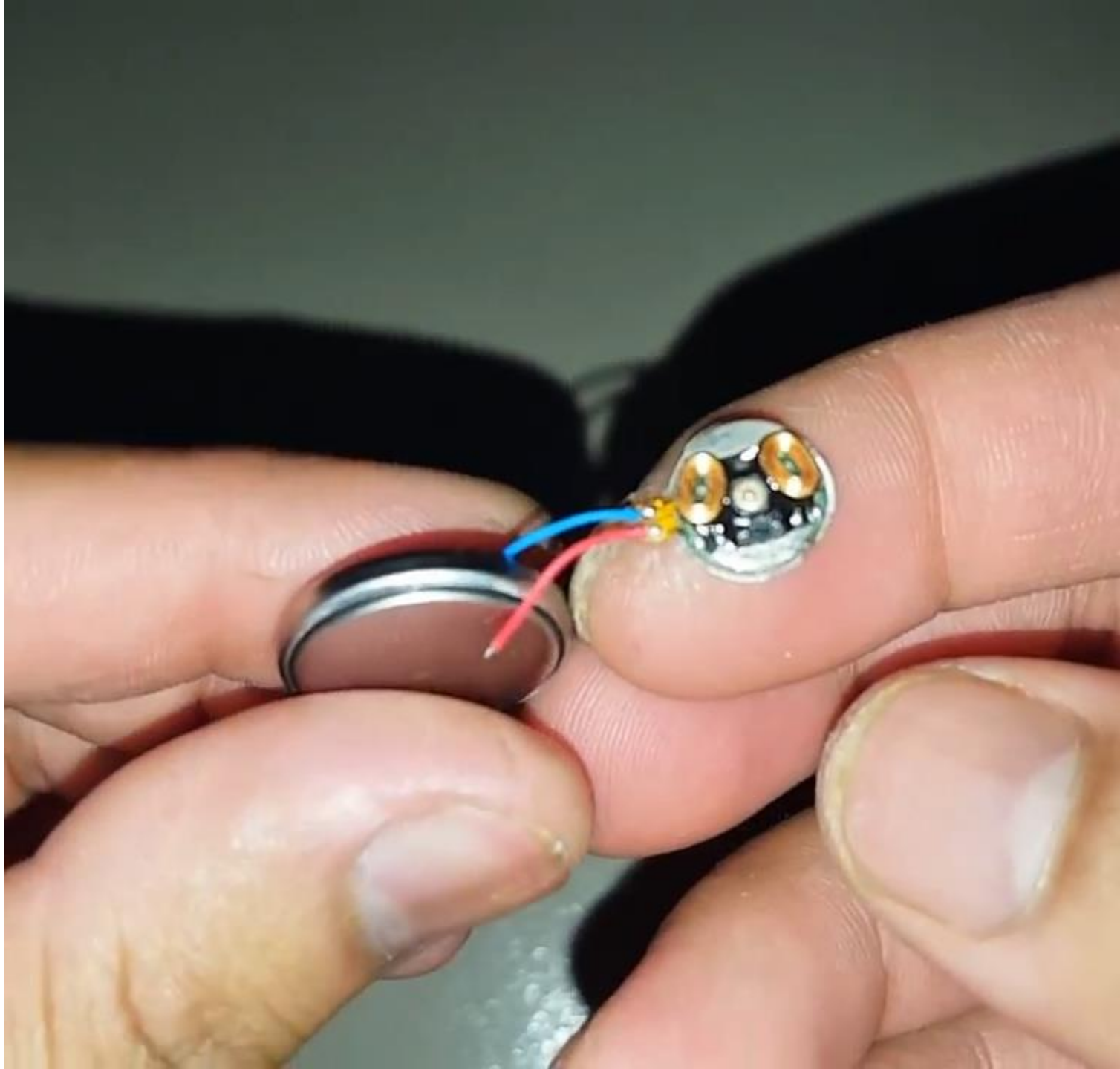
ERM COIN MOTOR



Coin or ‘Pancake’ Motor

Same operating principle as the cylindrical
But eccentric mass is kept inside small ‘coin’ body

Video source: https://youtu.be/iwEGqBpYaqc



APPLICATIONS

Notifications (*e.g.*, smartphone)

Video game controller haptics

Sensory substitution

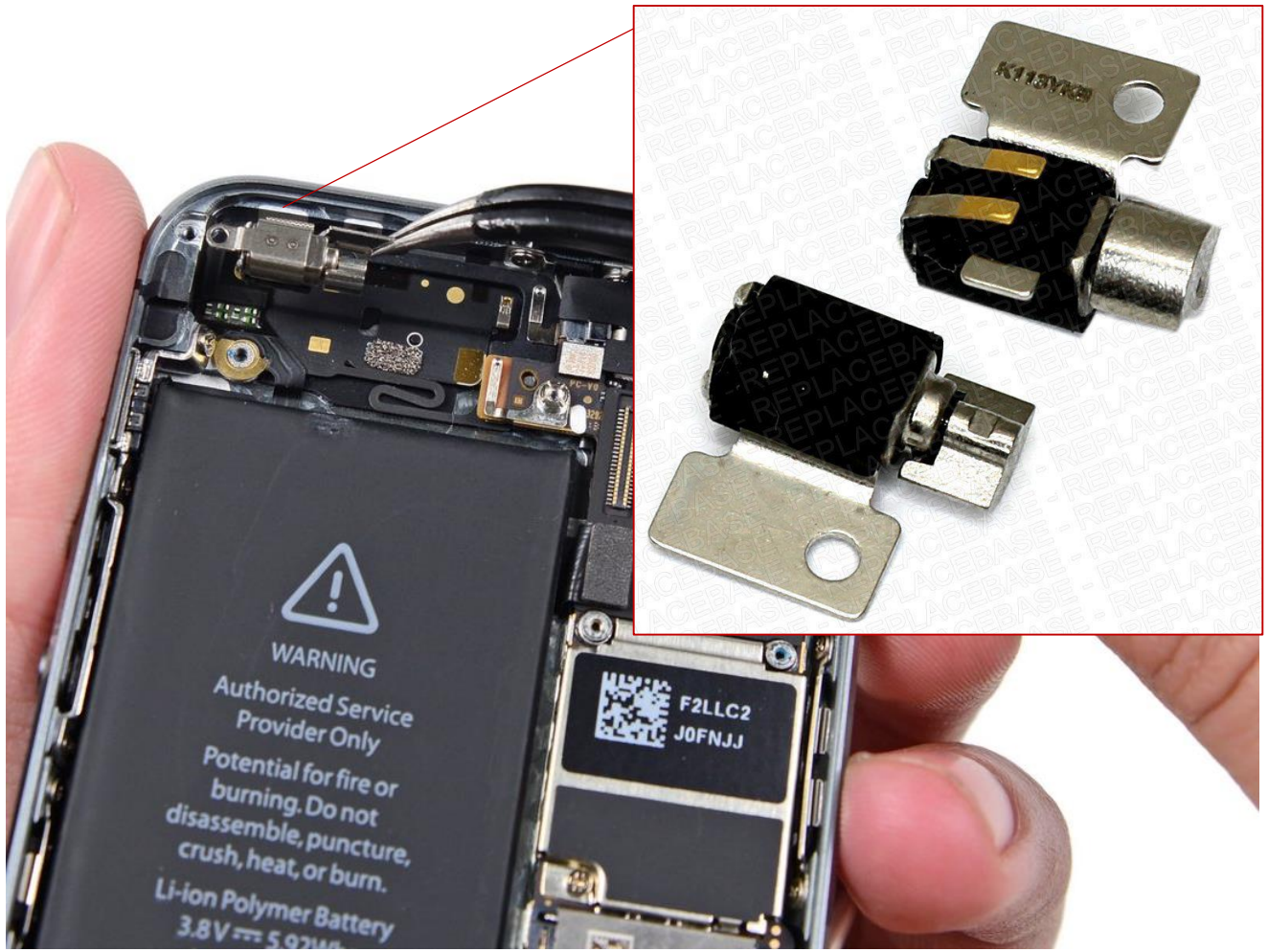
Robotics

Toothbrushes

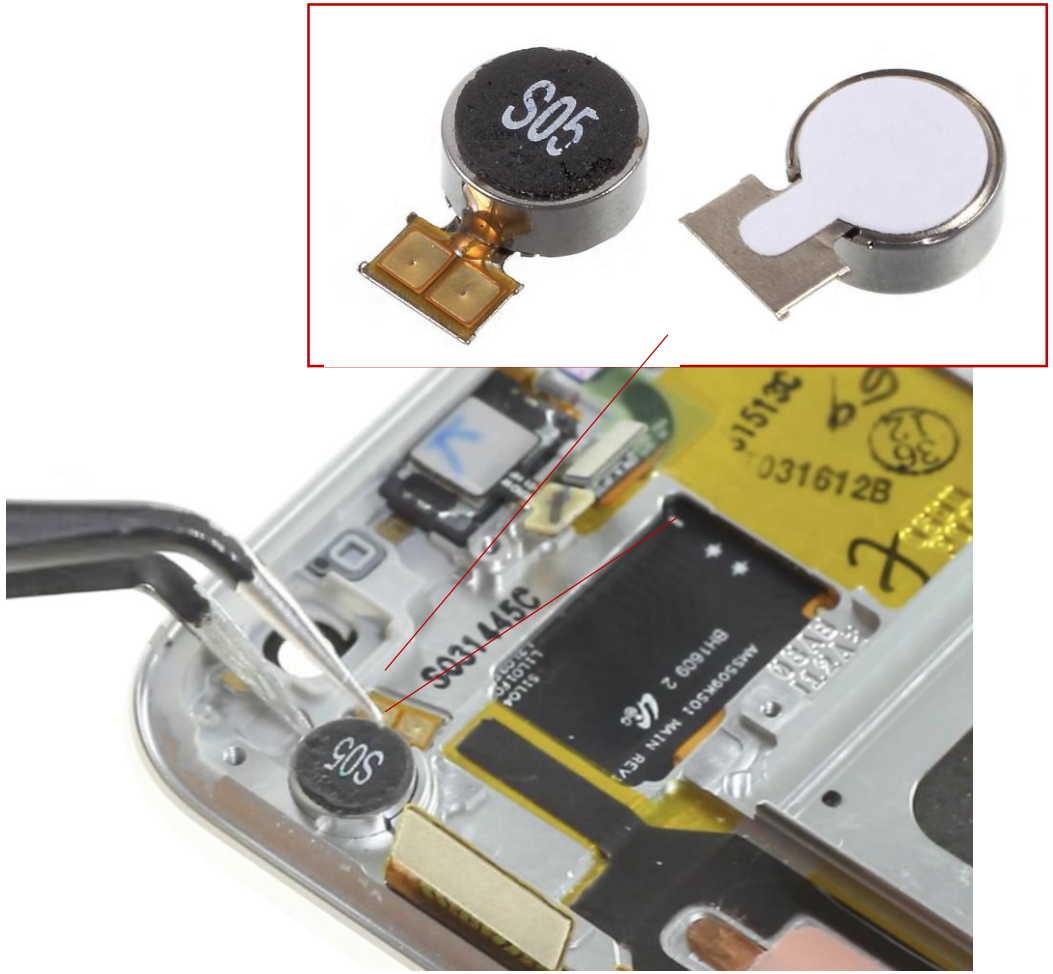
...

VIBRATION MOTORS

APPLICATIONS: SMARTPHONE HAPTICS



iPhone 5

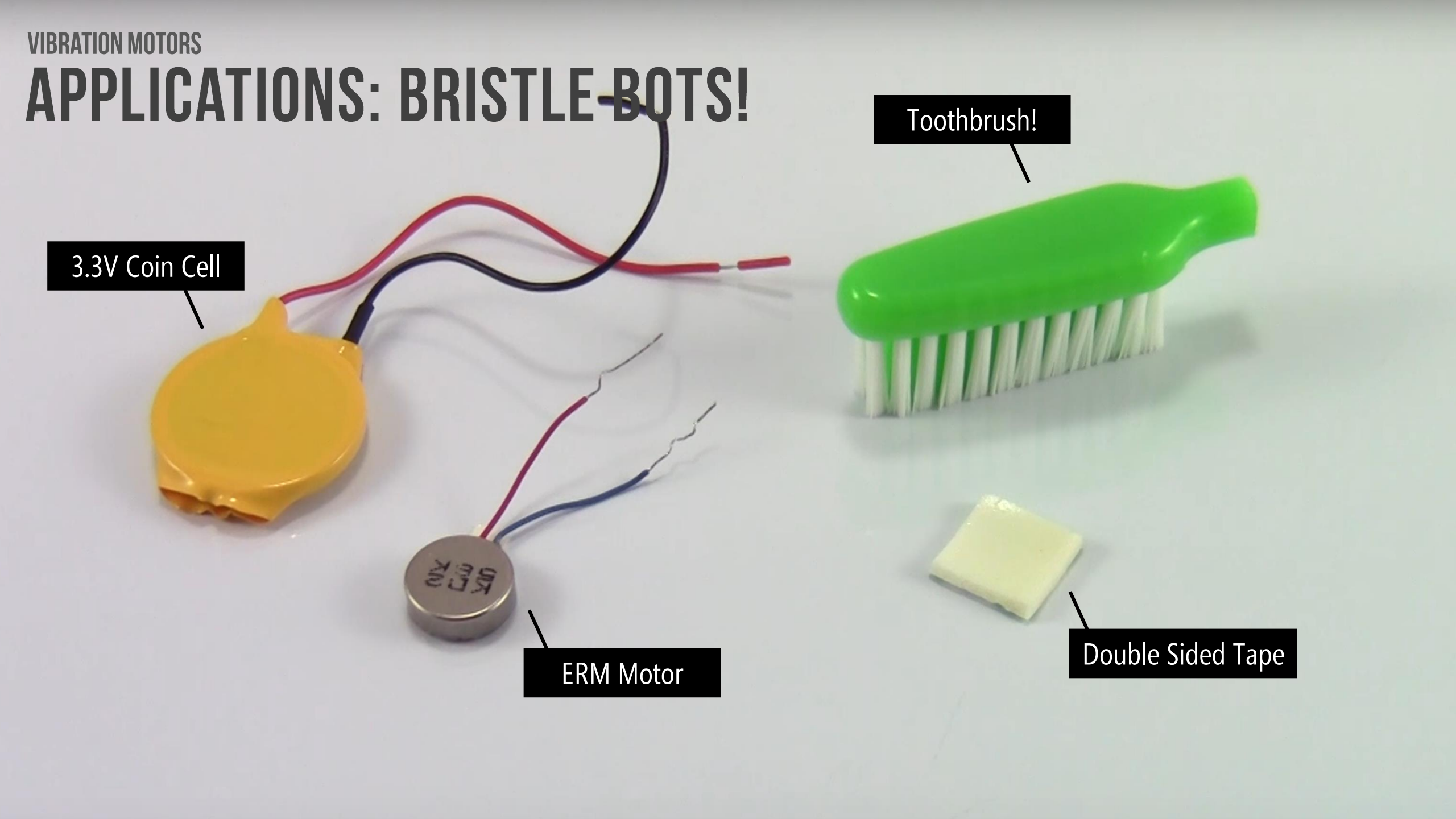


Samsung Galaxy 7


Samsung Galaxy S3
iPhone 4S
Sony Xperia Z
iPhone 5
HTC One X
iPhone 3GS

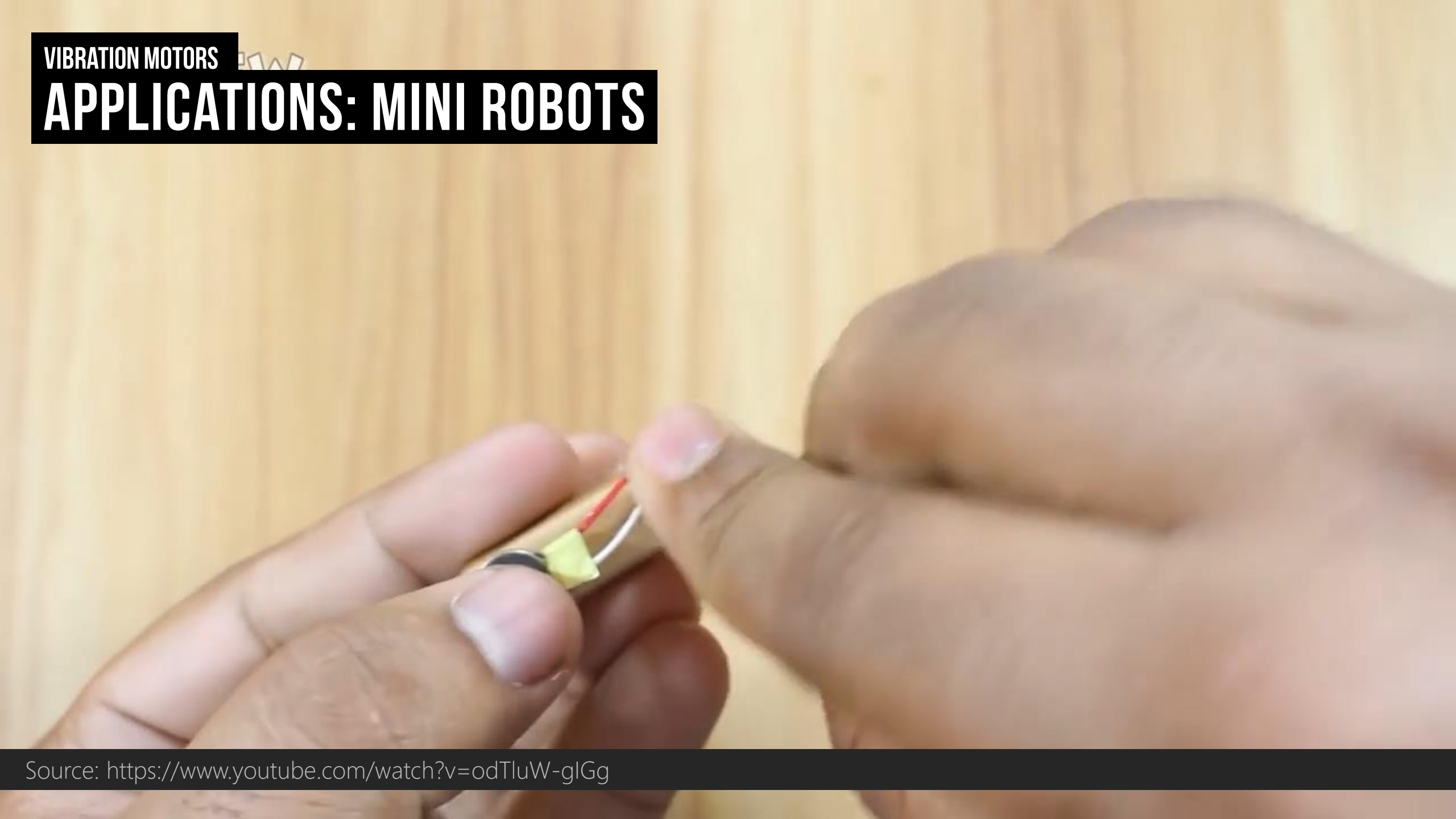
VIBRATION MOTORS

APPLICATIONS: BRISTLE BOTS!



VIBRATION MOTORS

APPLICATIONS: MINI ROBOTS



Source: https://www.youtube.com/watch?v=odTluW-gIGg

APPLICATIONS: FINGER/HAND GUIDANCE

Evaluating Haptic and Auditory Directional Guidance to Assist Blind People in Reading Printed Text Using Finger-Mounted Cameras

LEE STEARNS, University of Maryland, College Park
RUOFEI DU, University of Maryland, College Park
URAN OH, University of Maryland, College Park
CATHERINE JOU, University of Maryland, College Park
LEAH FINDLATER, University of Maryland, College Park
DAVID A. ROSS, Atlanta VA R&D Center of Excellence in Vision and Neurocognitive Rehabilitation (CVNR)
JON E. FROEHLICH, University of Maryland, College Park

The recent miniaturization of cameras has enabled finger-based reading approaches that provide blind and visually impaired readers with access to printed materials. Compared to handheld text scanners such as mobile phone applications, mounting a tiny camera on the user's own finger has the potential to mitigate camera framing issues, enable a blind reader to better understand the spatial layout of a document, and provide better control over reading pace. A finger-based approach, however, also introduces the need to guide the reader in physically navigating a document, such as tracing along lines of text. While previous work has proposed audio and haptic directional finger guidance for this purpose, user studies of finger-based reading have not provided an in-depth performance analysis of the finger-based reading process. To further investigate the effectiveness of finger-based sensing and feedback for reading printed text, we conducted a controlled lab experiment with 19 blind participants, comparing audio and haptic directional finger guidance within an iPad-based testbed. As a small follow-up, we later asked 4 of those participants to return and provide feedback on a preliminary wearable prototype called HandSight. Findings from the controlled experiment show similar performance between haptic and audio directional guidance, although audio may offer an accuracy advantage for tracing lines of text. Subjective feedback also highlights tradeoffs between the two types of guidance, such as the interference of audio guidance with speech output and the potential for desensitization to haptic guidance. While several participants appreciated the direct access to layout information provided by finger-based exploration, important concerns also arose about ease of use and the amount of concentration required. We close with a discussion on the effectiveness of finger-based reading for blind users and potential design improvements to the HandSight prototype.

General Terms: Human Factors

Additional Key Words and Phrases: Accessibility, Visual Impairments, Wearables, Real-Time OCR

1. INTRODUCTION

Despite the increased availability of digital information and screen reader software, reading printed text materials remains an important but challenging task for people who are blind or visually impaired. The inability to read menus, receipts and handouts, bills and other mail can negatively impact the daily activities of those living with visual impairment (*e.g.*, [Brady et al. 2013; Haymes et al. 2002]). Although braille has long provided a promising alternative, fewer than 10% of the approximately 2 million adults with severe visual impairment in the United States are braille literate [National Federation of the Blind Jernigan Institute 2009; National Center for Health Statistics 2012], and many materials are not available in braille format.

Although many devices and mobile applications—such as *SARA CE*[1], *KNFB Reader iOS*[2], and *OrCam*[3]—attempt to provide access to printed materials through camera capture and optical character recognition (OCR), open questions remain. One

[1] SARA CE: http://www.freedomscientific.com/Products/LowVision/SARA
[2] KNFB Reader: http://knfbreader.com/
[3] OrCam: http://www.orcam.com/

Evaluating Haptic and Auditory Directional Guidance to Assist Blind People In Reading Printed Text Using Finger-mounted Cameras
Stearns *et al.,* TACCESS'16

Evaluating Angular Accuracy of Wrist-based Haptic Directional Guidance for Hand Movement

Jonggi Hong*, Lee Stearns*, Tony Cheng*, Jon E. Froehlich*, David Ross†, Leah Findlater‡
*Department of Computer Science, University of Maryland, College Park
†Atlanta VA R&D Center of Excellence in Vision and Neurocognitive Rehabilitation (CVNR)
‡College of Information Studies, University of Maryland, College Park

ABSTRACT

Haptic guidance for the hand can offer an alternative to visual or audio feedback when those information channels are overloaded or inaccessible due to environmental factors, vision impairments, or hearing loss. We report on a controlled lab experiment to evaluate the impact of directional wrist-based vibro-motor feedback on hand movement, comparing lower-fidelity (4-motor) and higher-fidelity (8-motor) wristbands. Twenty blindfolded participants completed a series of trials, which consisted of interpreting a haptic stimulus and executing a 2D directional movement on a touchscreen. We compare the two conditions in terms of movement error and trial speed, but also analyze the impact of specific directions on performance. Our results show that doubling the number of haptic motors reduces directional movement error but not to the extent expected. We also empirically derive an apparent lower bound in accuracy of ~25° in interpreting and executing on the directional haptic signal.

Keywords: Wearables, haptics, non-visual directional guidance.

Index Terms: H.5.m. Information interfaces and presentation (*e.g.*, HCI)

1 INTRODUCTION

Haptic guidance for the hand can be useful for a range of applications, including virtual reality [26, 33], teaching new motor skills [11, 20], and accessibility for blind users [13, 30]. Such guidance can offer an alternative to visual or audio feedback when those information channels are overloaded or inaccessible due to environmental factors, vision impairments, or hearing loss. For example, a driver may need to interact non-visually with their car's console [4], a firefighter may require fine-grained guidance in a smoky room with no visibility [9], or a blind user may want to accurately trace a line of printed text with a wearable system while listening to text-to-speech output of that text [13, 30].

While researchers have looked at both finger-worn [13, 30] and wrist-worn haptic directional guidance [11, 26, 28, 33], the wrist offers a balance of proximity to the finger, sensitivity [7], surface area, and social acceptability. The larger surface area, for example, allows for a greater number of vibration sources than is possible on the finger. Moreover, emerging smartwatches offer a compelling and timely opportunity to embed haptic feedback in the watchband itself to support a range of non-visual interactions.

Wrist-based haptic feedback, however, has primarily been studied for notifications, such as providing pulse patterns with a single motor (*e.g.*, [24]) or localizing one motor in a grid of motors (*e.g.*, [5, 22]). Fine-grained directional guidance for the hand has received less attention. Weber *et al.* [33] and Sergi *et al.* [28], for example, studied wrist-based haptic guidance for 3D hand movements and a relatively small set of target directions (4 or 6). However, the accuracy with which users can interpret an arbitrary target direction (*e.g.*, our study evaluates 32 directions) and execute a corresponding movement is not yet known—basic information that would impact the design of a range of interactions, from quickly finding a target to tracing a path.

In this paper, we report on a controlled lab experiment to evaluate the impact of wrist-based vibro-motor feedback on hand movement, comparing lower-fidelity (4-motor) and higher-fidelity (8-motor) wristbands. Twenty participants completed a series of trials consisting of interpreting a haptic stimuli and executing a 2D directional movement on a touchscreen. For the results to apply to non-visual interaction scenarios such as situational impairments, participants were blindfolded. We assess movement error and trial speed, but also analyze the impact of specific directions on performance. The contributions include: (i) empirical evidence that doubling the number of haptic motors reduces directional movement error, and that movement error is greater to the upper-left than in other directions; (ii) identification of an empirically derived maximum threshold of ~25° accuracy using our approach; (iii) design considerations for incorporating directional haptic guidance into a smartwatch band—in particular, for most applications, a four-motor wristband may be sufficient.

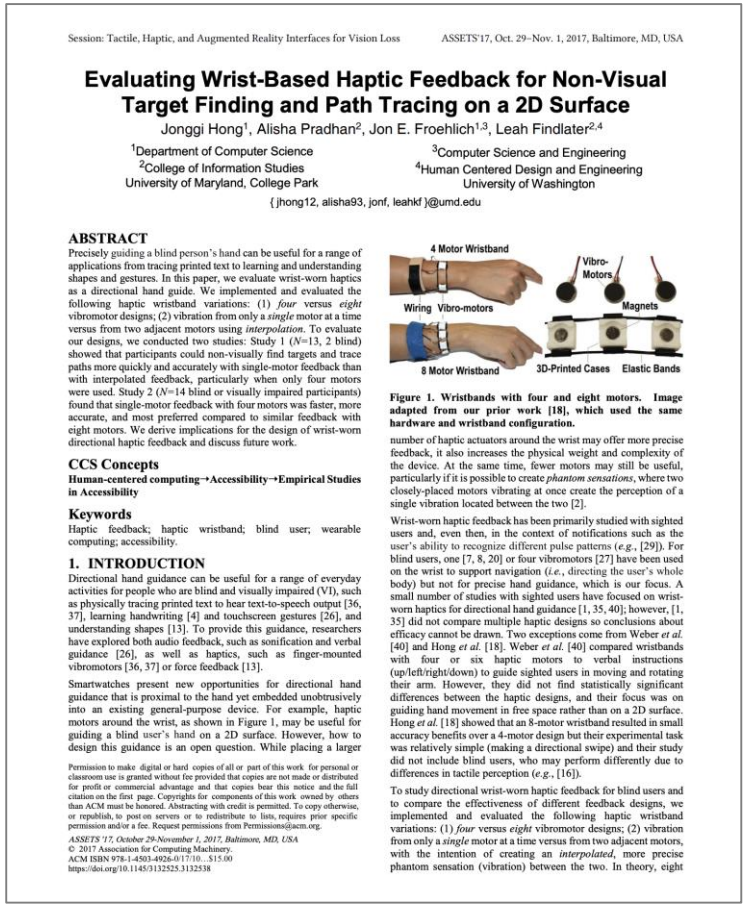
2 RELATED WORK

Designing effective haptic feedback requires a consideration of factors such as human perception thresholds and the ability to discriminate different haptic patterns; see [6] for a survey. While our focus is on fine-grained directional guidance for hand movements, many projects have looked at directional feedback for directing the user's whole body (*e.g.*, for navigation). These include a few examples with wrist-worn devices with one [2, 3, 18] or four [23] vibro-motors, but more commonly other parts of the body are used, such as the waist or back [8, 17]. In terms of general feedback on the wrist or forearm, Cholewiak *et al.* [7] showed that localization is most precise when the stimulus is close to an anatomical point of reference (*e.g.*, wrist or elbow).

Most closely related to our study is work on haptic directional guidance for the hand or arm. Schätzle *et al.* [26] studied different fidelities of wrist-based translation and rotation feedback, but the study had critical validity issues including no counterbalancing. In follow-up work, Weber *et al.* [33] compared wristband conditions with four or six motors, assessing the user's ability to perceive a signal and rotate or move their arm in one of four or six directions. Providing a verbal direction ("up", "down", "left", or "right") resulted in better performance than the haptic feedback for this relatively simple task. In contrast, we evaluate movement in 32 directions, for which discrete verbal feedback would not be feasible. Sergi *et al.* [28] compared the impact of visual and haptic feedback on reaching accuracy in a virtual reality (visual) environment with a small set of targets. Stanley and

*email: {jhong12, lstearns, jonf}@umd.edu
†email: ross0128@bellsouth.net
‡email: leahkf@umd.edu

Evaluating Angular Accuracy of Wrist-based Haptic Directional Guidance for Hand Movement
Hong *et al.,* Graphics Interface'16



Evaluating Wrist-Based Haptic Feedback for Non-Visual Target Finding and Path Tracing on a 2D Surface

Jonggi Hong[1], Alisha Pradhan[2], Jon E. Froehlich[1,3], Leah Findlater[2,4]
[1]Department of Computer Science, [2]College of Information Studies, University of Maryland, College Park
[3]Computer Science and Engineering, [4]Human Centered Design and Engineering, University of Washington
{ jhong12, alisha93, jonf, leahkf }@umd.edu

ABSTRACT

Precisely guiding a blind person's hand can be useful for a range of applications from tracing printed text to learning and understanding shapes and gestures. In this paper, we evaluate wrist-worn haptics as a directional hand guide. We implemented and evaluated the following haptic wristband variations: (1) *four* versus *eight* vibromotor designs; (2) vibration from only a *single* motor at a time versus from two adjacent motors using *interpolation*. To evaluate our designs, we conducted two studies: Study 1 (N=13, 2 blind) showed that participants could non-visually find targets and trace paths more quickly and accurately with single-motor feedback than with interpolated feedback, particularly when only four motors were used. Study 2 (N=14 blind or visually impaired participants) found that single-motor feedback with four motors was faster, more accurate, and most preferred compared to similar feedback with eight motors. We derive implications for the design of wrist-worn directional haptic feedback and discuss future work.

CCS Concepts
Human-centered computing→Accessibility→Empirical Studies in Accessibility

Keywords
Haptic feedback; haptic wristband; blind user; wearable computing; accessibility.

1. INTRODUCTION

Directional hand guidance can be useful for a range of everyday activities for people who are blind and visually impaired (VI), such as physically tracing printed text to hear text-to-speech output [36, 37], learning handwriting [4] and touchscreen gestures [26], and understanding shapes [13]. To provide this guidance, researchers have explored both audio feedback, such as sonification and verbal guidance [26], as well as haptics, such as finger-mounted vibromotors [36, 37] or force feedback [13].

Smartwatches present new opportunities for directional hand guidance that is proximal to the hand yet embedded unobtrusively into an existing general-purpose device. For example, haptic motors around the wrist, as shown in Figure 1, may be useful for guiding a blind user's hand on a 2D surface. However, how to design this guidance is an open question. While placing a larger number of haptic actuators around the wrist may offer more precise feedback, it also increases the physical weight and complexity of the device. At the same time, fewer motors may still be useful, particularly if it is possible to create *phantom sensations*, where two closely-placed motors vibrating at once create the perception of a single vibration located between the two [2].

ASSETS '17, October 29-November 1, 2017, Baltimore, MD, USA

ACM ISBN 978-1-4503-4926-0/17/10…$15.00
https://doi.org/10.1145/3132525.3132538

Figure 1. Wristbands with four and eight motors. Image adapted from our prior work [18], which used the same hardware and wristband configuration.

Wrist-worn haptic feedback has been primarily studied with sighted users and, even then, in the context of notifications such as the user's ability to recognize different pulse patterns (*e.g.*, [29]). For blind users, one [7, 8, 20] or four vibromotors [27] have been used on the wrist to support navigation (*i.e.*, directing the user's whole body) but not for precise hand guidance, which is our focus. A small number of studies with sighted users have focused on wrist-worn haptics for directional hand guidance [1, 35, 40]; however, [1, 35] did not compare multiple haptic designs so conclusions about efficacy cannot be drawn. Two exceptions come from Weber *et al.* [40] and Hong *et al.* [18]. Weber *et al.* [40] compared wristbands with four or six haptic motors to verbal instructions (up/left/right/down) to guide sighted users in moving and rotating their arm. However, they did not find statistically significant differences between the haptic designs, and their focus was on guiding hand movement in free space rather than on a 2D surface. Hong *et al.* [18] showed that an 8-motor wristband resulted in small accuracy benefits over a 4-motor design but their experimental task was relatively simple (making a directional swipe) and their study did not include blind users, who may perform differently due to differences in tactile perception (*e.g.*, [16]).

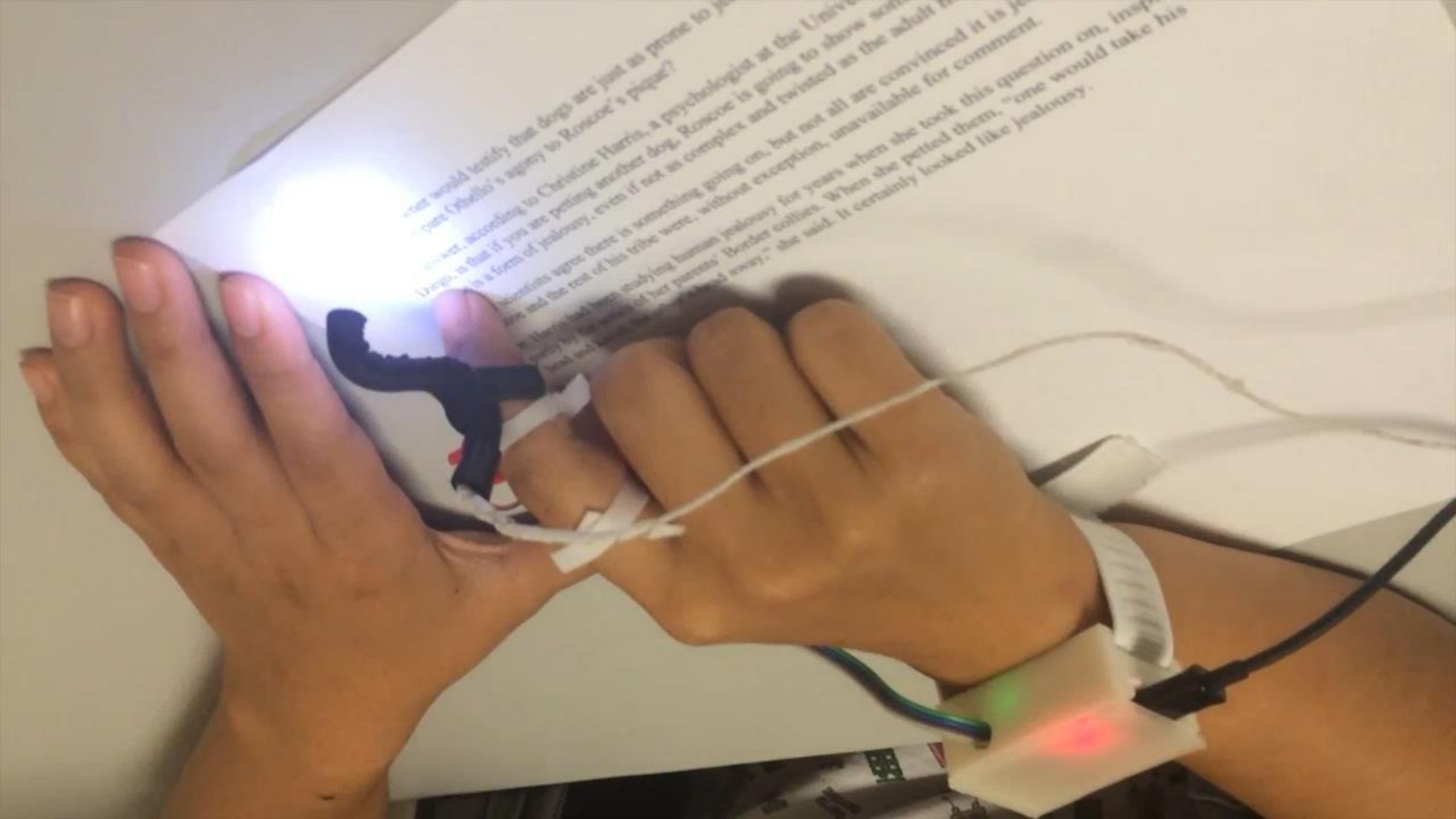
To study directional wrist-worn haptic feedback for blind users and to compare the effectiveness of different feedback designs, we implemented and evaluated the following haptic wristband variations: (1) *four* versus *eight* vibromotor designs; (2) vibration from only a *single* motor at a time versus from two adjacent motors, with the intention of creating an *interpolated*, more precise phantom sensation (vibration) between the two. In theory, eight

Evaluating Wrist-Based Haptic Feedback for Non-Visual Target Finding and Path Tracing on a 2D Surface
Hong *et al.,* ASSETS'17

HANDSIGHT

READING SUPPORT FOR BLIND

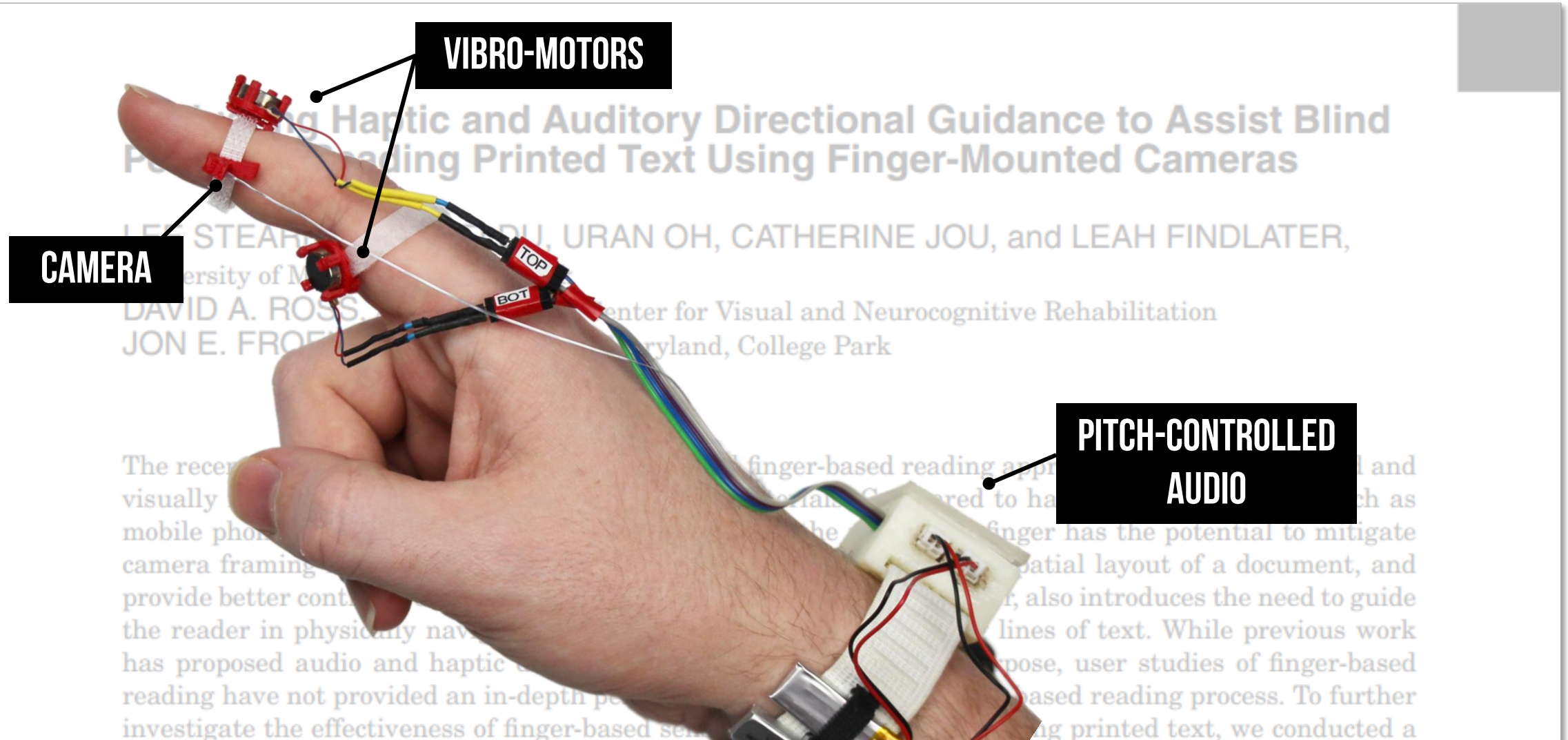
Steam *et al.*, ACVR'14



HANDSIGHT

HAPTIC VS AUDIO LINE GUIDANCE

TACCESS'16



HANDSIGHT

HAPTIC VS AUDIO LINE GUIDANCE

TACCESS'16

g Haptic and Auditory Directional Guidance to Assist Blind
eading Printed Text Using Finger-Mounted Cameras

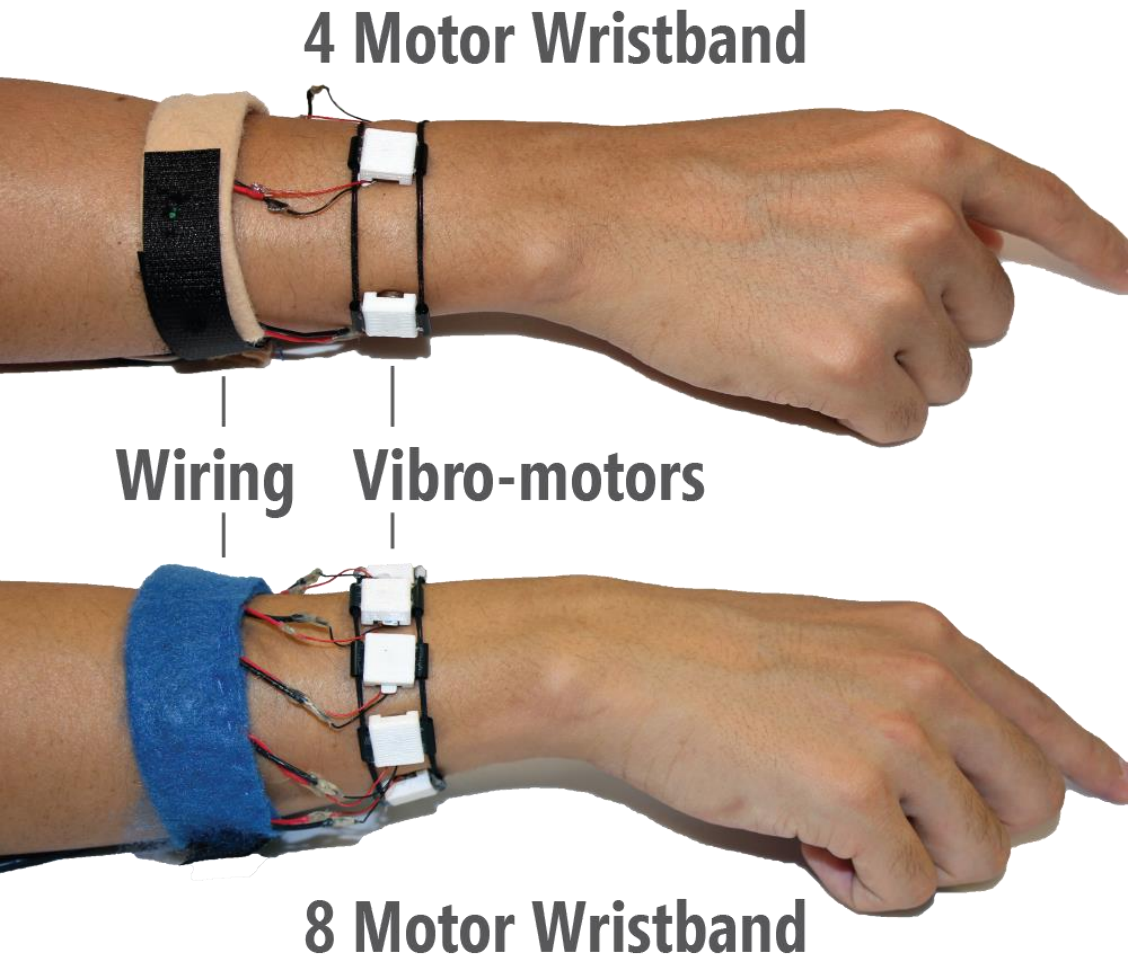
LEE STEAD DU, URAN OH, CATHERINE JOU, and LEAH FINDLATER,
University of M
DAVID A. ROSS, enter for Visual and Neurocognitive Rehabilitation
JON E. FROE ryland, College Park

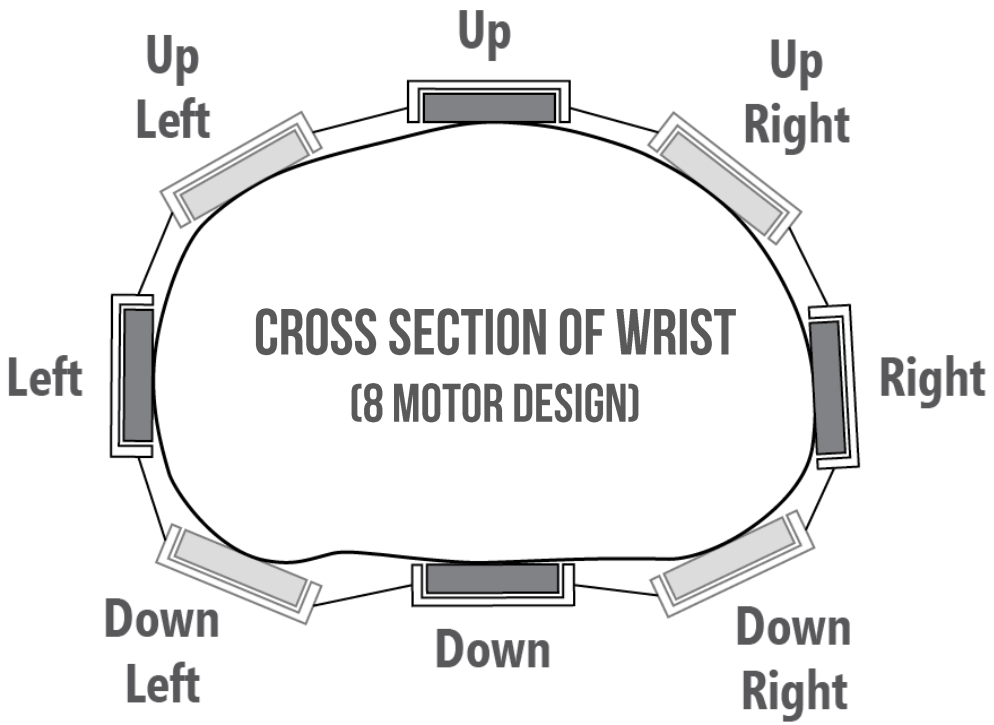
The recent finger-based reading approaches that provide blind and visually i ared to handheld text scanners such as mobile pho finger has the potential to mitigate camera framing atial layout of a document, and provide better cont , also introduces the need to guide the reader in physic lines of text. While previous work has proposed audio and haptic pose, user studies of finger-based reading have not provided an in-depth ased reading process. To further investigate the effectiveness of finger-based g printed text, we conducted a

HANDSIGHT

WRIST HAPTICS

GI'16



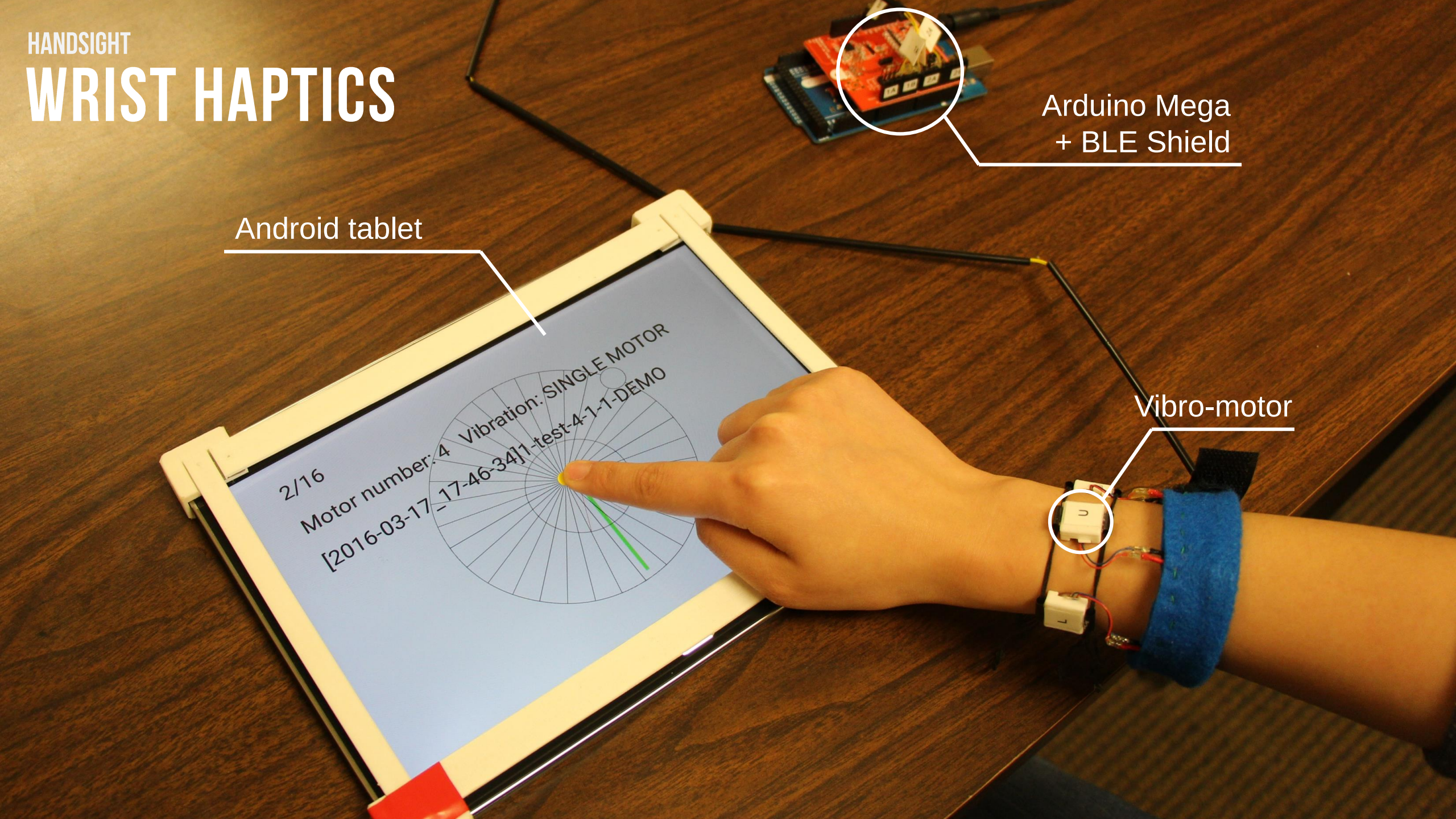


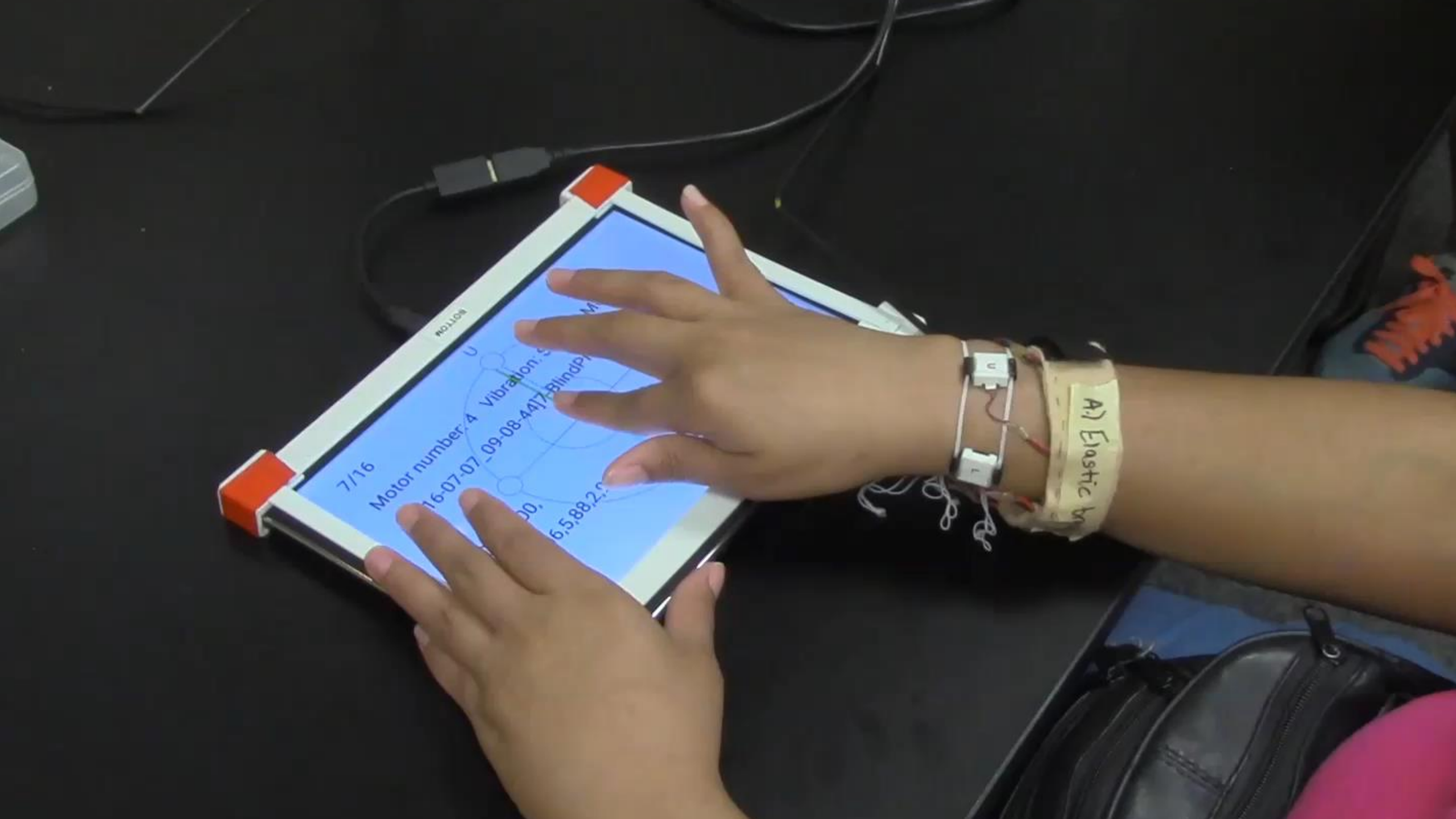
TESTED 32 MOVEMENT DIRECTIONS

(11.25° INTERVALS)

HANDSIGHT

WRIST HAPTICS

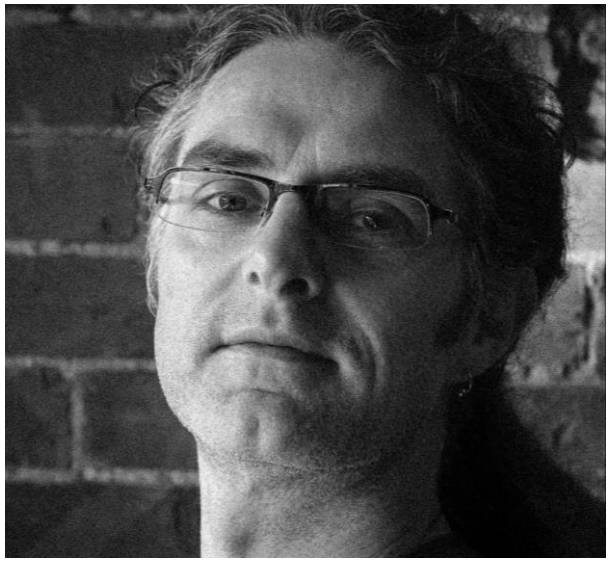



7/16
Motor number 4 Vibration
A) Elastic

HAPTIC RESEARCHERS IN HCI: A SMALL SAMPLE!

Professor Hong Tan
Haptic Interface Research Lab
Electrical & Computer Engineering
Purdue University

Professor Karon MacLean
Sensory Perception & Interaction Group
Computer Science
University of British Columbia



Professor Stephen Brewster
Multimodal Interaction Group
Computer Science
University of Glasgow

Why I work on **haptic interfaces**? Of the five major human senses of vision, audition, taction (touch and proprioception), olfaction and gustation, only the first three have been engaged in most human-machine interface research. Of these three, a disproportional majority of work has been conducted on visual and auditory systems.

Professor Hong Tan
Haptic Interface Research Lab
Electrical & Computer Engineering
Purdue University

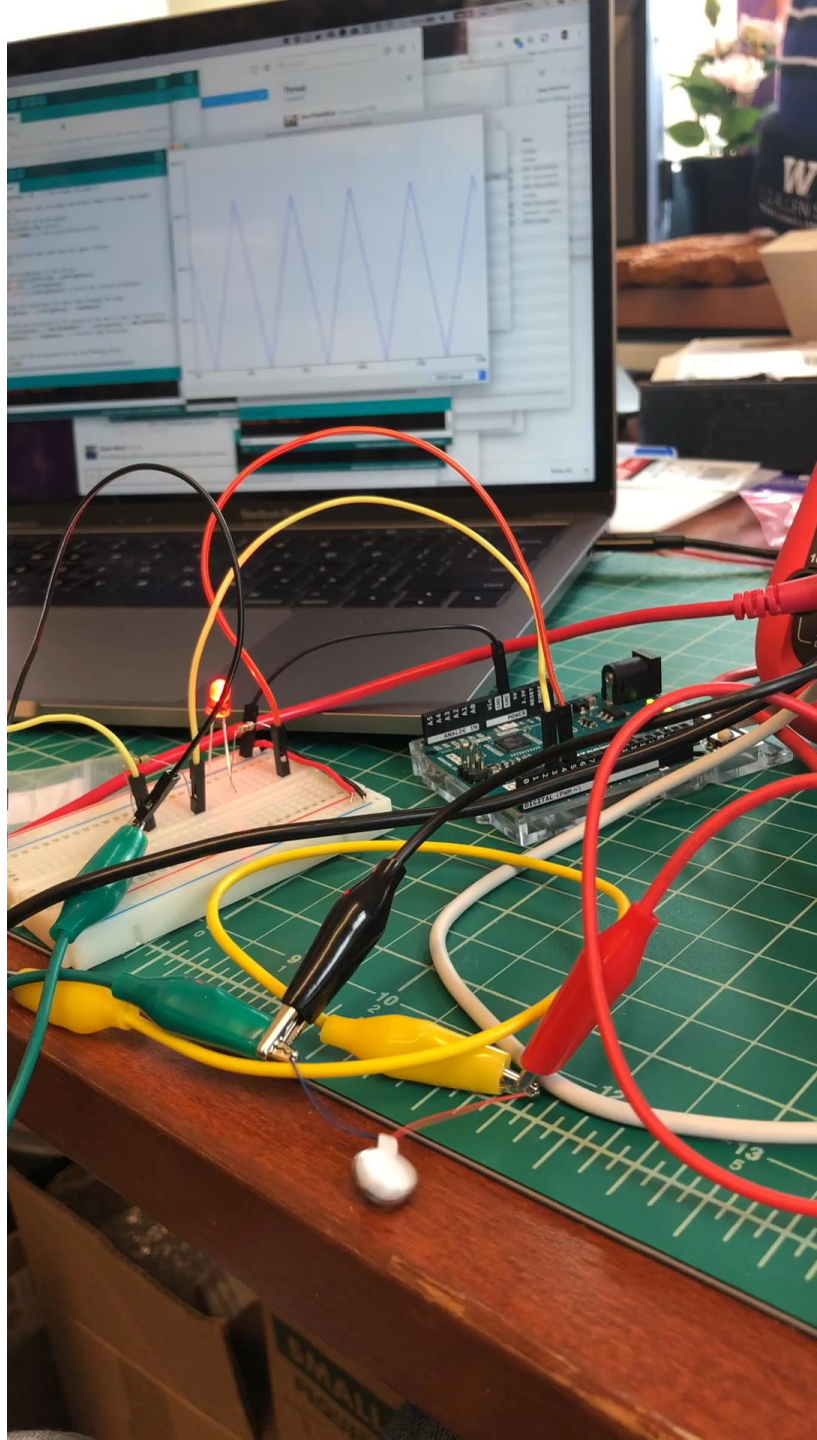
Source: https://engineering.purdue.edu/~hongtan/

VIBRATION MOTOR

ACTIVITY: USE VIBRO-MOTOR

Now fade the **haptic motor** on and off

What's the required **input voltage** and **current**?



Two wires are used to control/power the vibe. Simply provide power from a battery or microcontroller pin (red is positive, blue is negative) and it will buzz away. Works from 2V up to 5V, higher voltages result in more current draw but also a stronger vibration.

If you want to reduce the current draw/strength (for example, to control it directly from an Arduino pin) try putting a resistor (100 to 1000 ohms) in series. For full power control, a small PN2222 transistor can control a motor easily, some experimentation may be required!

TECHNICAL DETAILS

- Dimension: 10mm diameter, 2.7mm thick
- Voltage: 2V - 5V
- 5V current draw: 100mA, 4V current draw: 80mA, 3V current draw: 60mA, 2V current draw: 40mA
- 11000 RPM at 5V
- Weight: 0.9 gram

VIBRATION MOTOR

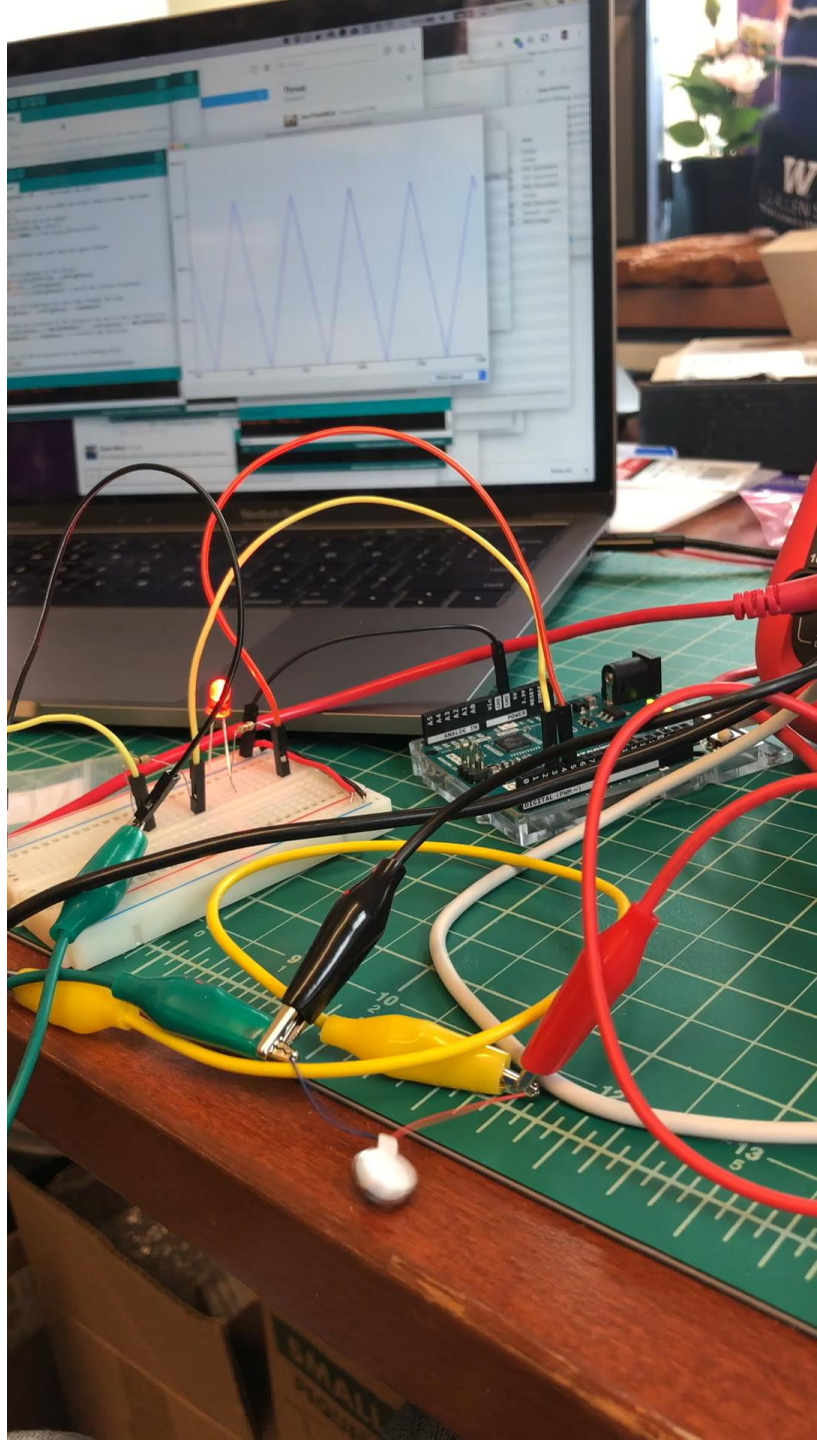
ACTIVITY: USE VIBRO-MOTOR

Now fade the **haptic motor** on and off

What's the required **input voltage** and **current**?

From my experimenting, need at least ~**1.5-2V** and ~**20-25mA**.

Use a series resistor of **50Ω**

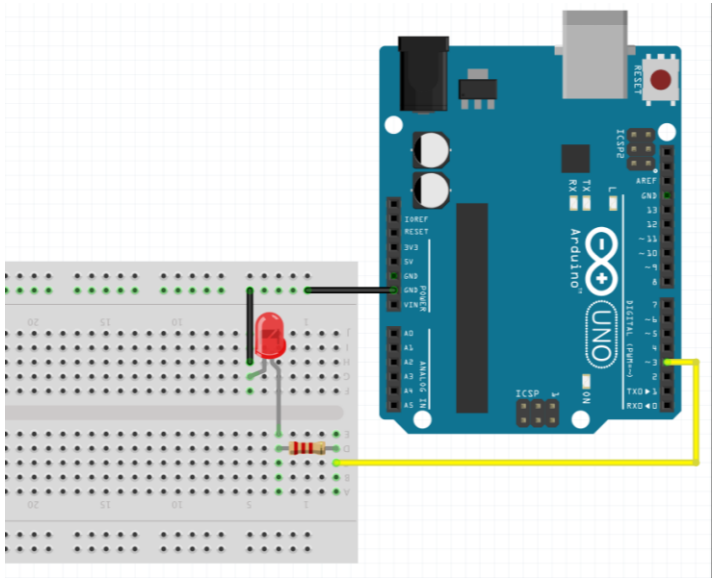


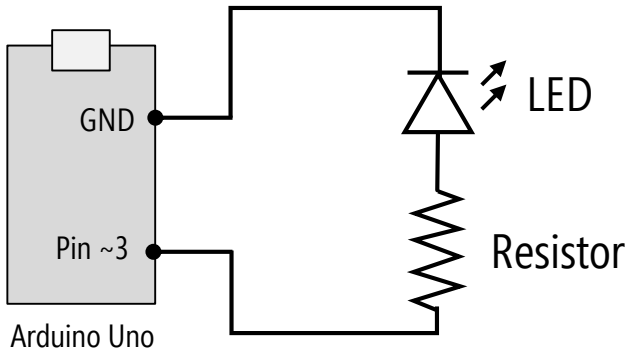
ANALOG OUTPUT

FADE VIBRO-MOTOR ON AND FADE LED OFF

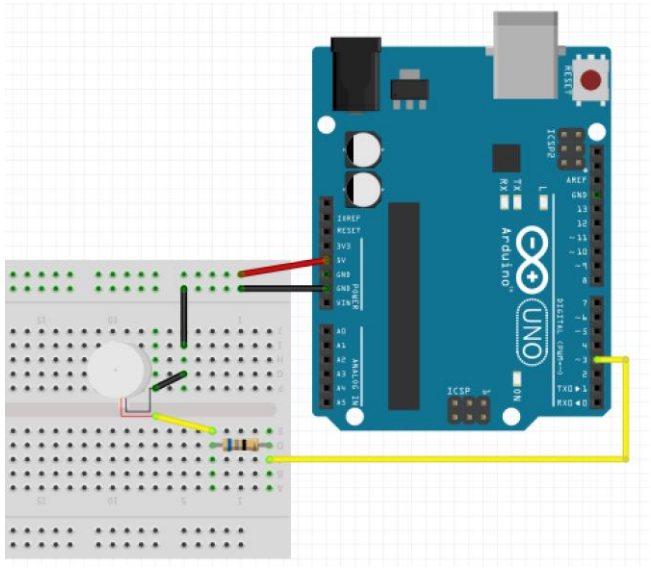
Similar circuit as before but we need to swap the LED and vibro-motor

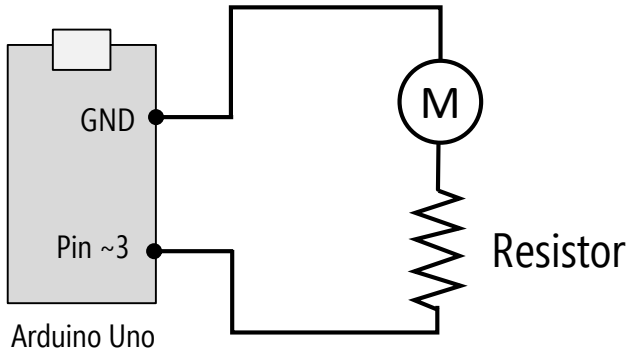
Circuit: Fade LED on and off via the pin 3





Circuit: Fade vibro-motor on and off via the pin 3



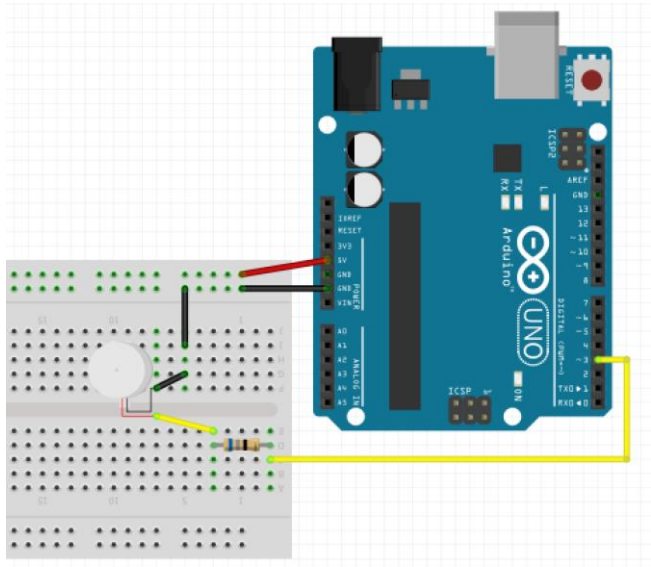


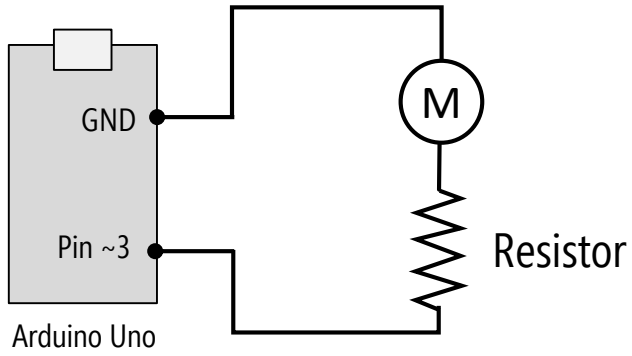
ANALOG OUTPUT

FADE VIBRO-MOTOR ON AND FADE LED OFF

Similar circuit as before but we need to swap the LED and vibro-motor

Circuit: Fade vibro-motor on and off via the pin 3





Code: Can use same code as the LED fade on and off

FadeOnAndOff

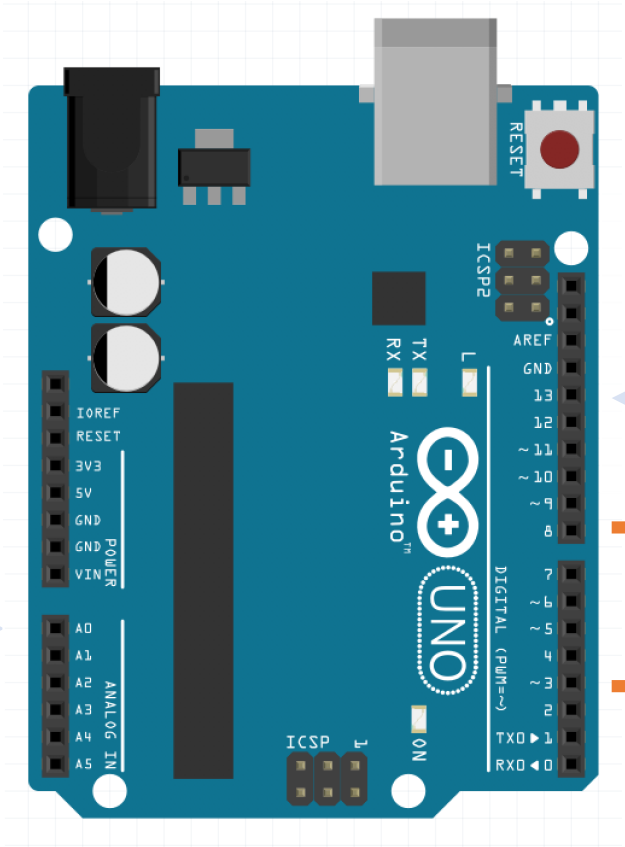
```
const int LED_OUTPUT_PIN = 3;
const int MIN_BRIGHTNESS = 0; // the min brightness
const int MAX_BRIGHTNESS = 255; // the max analog out value on Uno, Leonardo, etc. is 255 (8-bit max)

int _fadeAmount = 5;      // the amount to fade the LED by on each step
int _curBrightness = 0;   // how bright the LED is

// The setup function runs once when you press reset or power the board
void setup() {
  // set the LED pin to as an output
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  Serial.begin(9600); // for using Serial.println
}

// The loop function runs over and over again forever
void loop() {

  // set the brightness of the LED pin
  analogWrite(LED_OUTPUT_PIN, _curBrightness);
  Serial.println(_curBrightness); // print out current brightness

  // change the brightness for next time through the loop
  _curBrightness = _curBrightness + _fadeAmount;

  // reverse the direction of the fading at the end of each fade direction
  if (_curBrightness <= MIN_BRIGHTNESS || _curBrightness >= MAX_BRIGHTNESS) {
    _fadeAmount = -_fadeAmount; // reverses fade direction
  }

  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

DIGITAL AND ANALOG OUTPUT

This completes our quick tour of digital and analog output on the Arduino

Analog Input on A0

Reads in any value between 0V or 5V using the analogRead function. In this case, the value of a photocell



Digital Input on pin 13

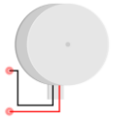
Reads in a digital input signal (anything below 2.5V converted to LOW, anything above 2.5V converted to HIGH). In this case, the value of switch.

Digital Output on pin 8

Writes out 0V or 5V using digitalWrite function. In this case, turning on and off an LED.

Analog Output on pin 3

Writes out any value between 0V or 5V using analogWrite function. In this case, vibrating a motor (where strength of vibration proportional to voltage). Only pins with a tilde ~ can be analog outputs.

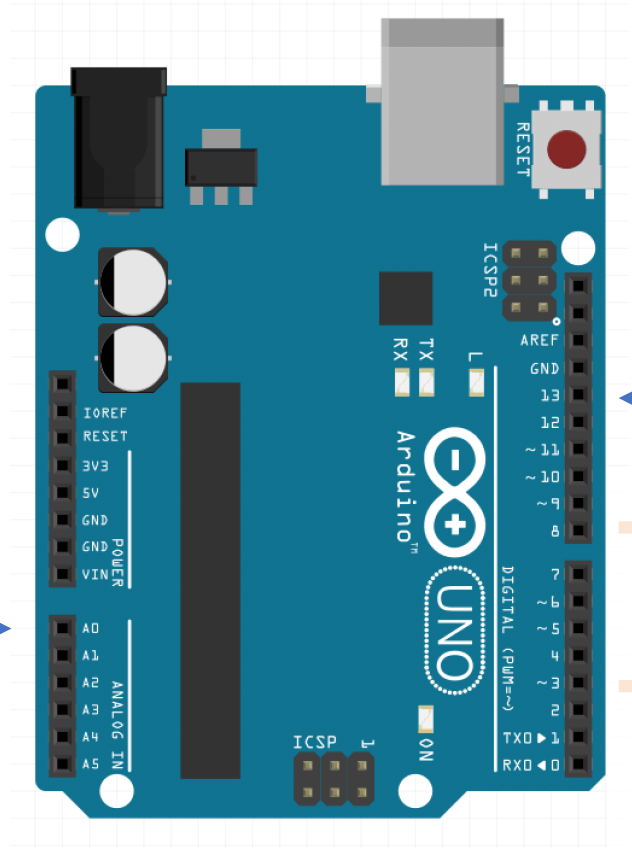


DIGITAL AND ANALOG OUTPUT

Now we are going to switch from talking about output to talking about input

Analog Input on A0

Reads in any value between 0V or 5V using the analogRead function. In this case, the value of a photocell



Digital Input on pin 13

Reads in a digital input signal (anything below 2.5V converted to LOW, anything above 2.5V converted to HIGH). In this case, the value of switch.

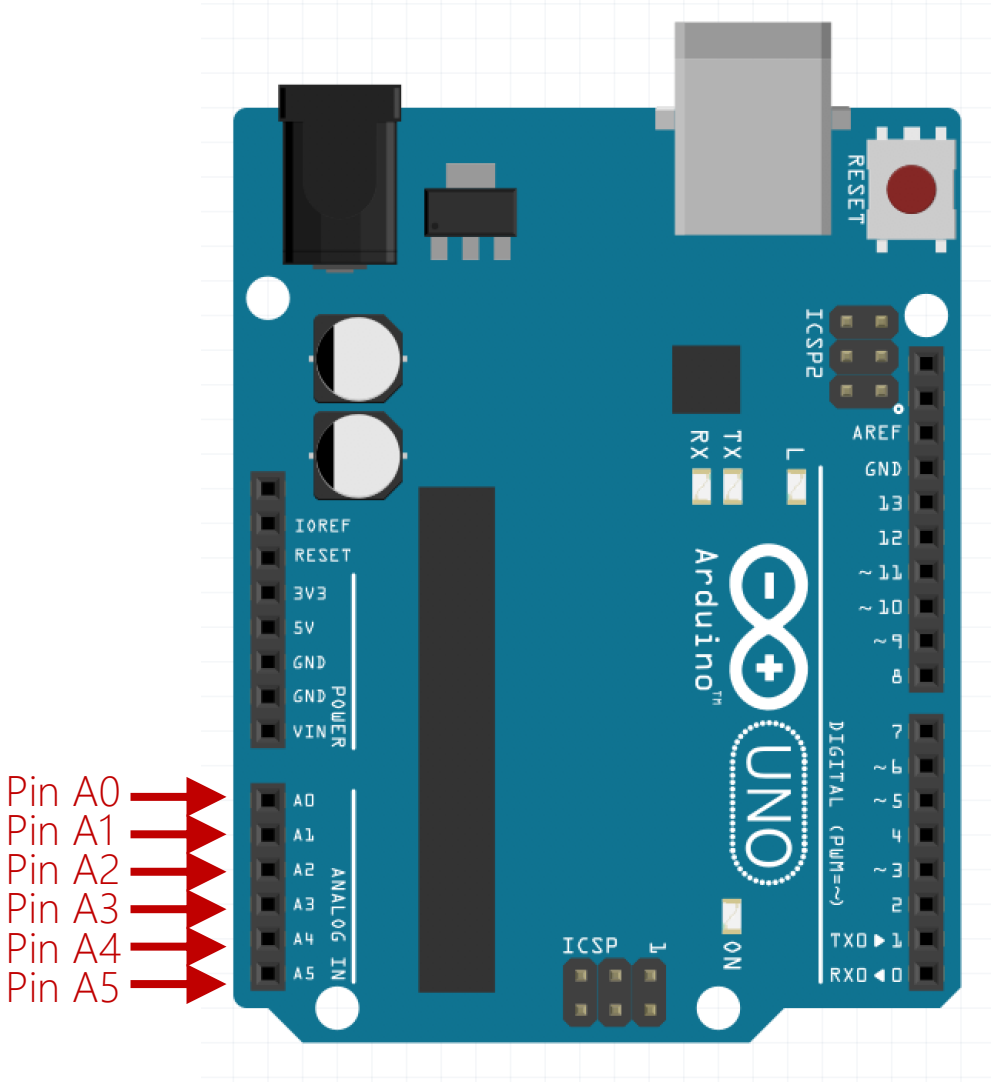
Digital Output on pin 8

Writes out 0V or 5V using digitalWrite function. In this case, turning on and off an LED.

Analog Output on pin 3

Writes out any value between 0V or 5V using analogWrite function. In this case, vibrating a motor (where strength of vibration proportional to voltage). Only pins with a tilde ~ can be analog outputs.

RECALL THAT WE ONLY HAVE SIX ANALOG INPUTS

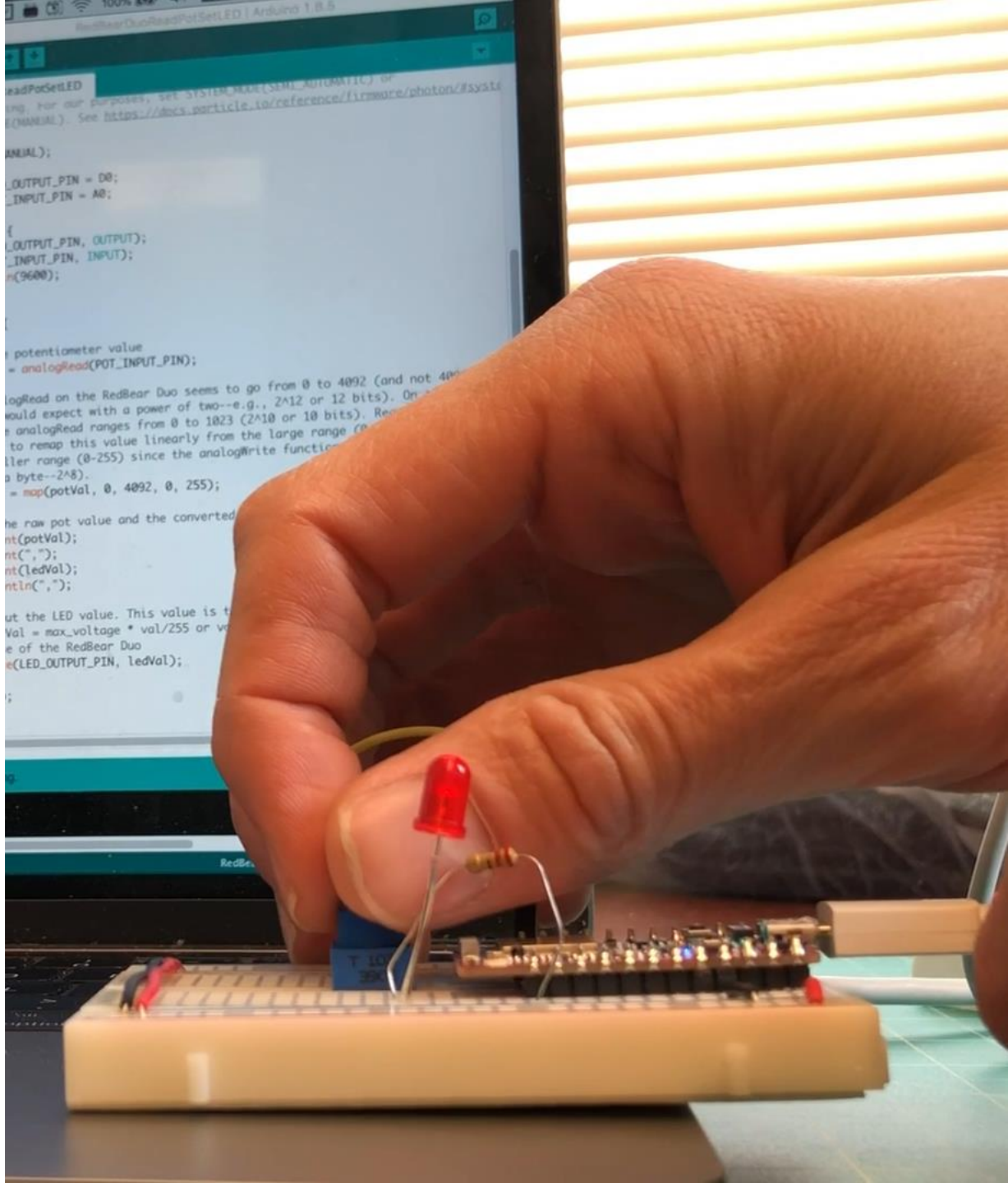


Let’s build a circuit and write code to **fade an LED based on the value of a trim potentiometer**. For this, we will use analog input (analogRead) and output (analogWrite)

ACTIVITY: CONTROL LED BRIGHTNESS WITH TRIMPOT

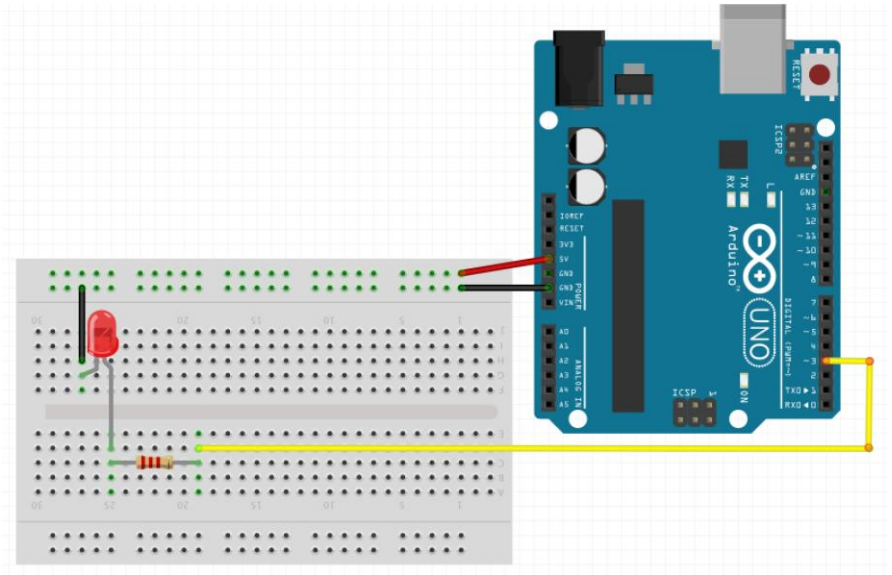
Same output circuit as before

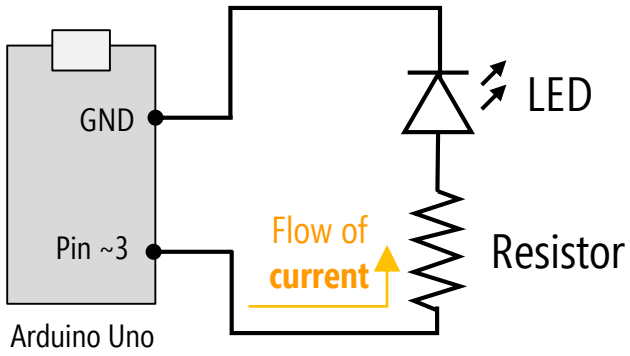
Now we need to add input



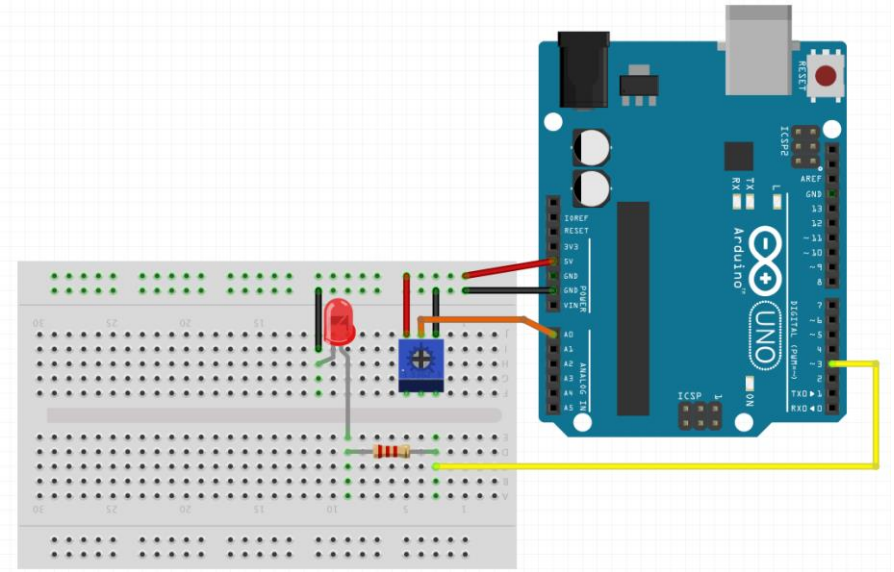
ACTIVITY: CONTROL LED BRIGHTNESS WITH POTENTIOMETER

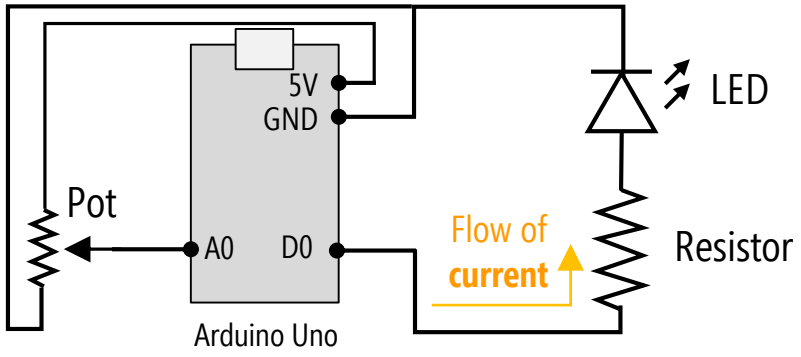
Old Circuit: Fade LED on/off via pin 3





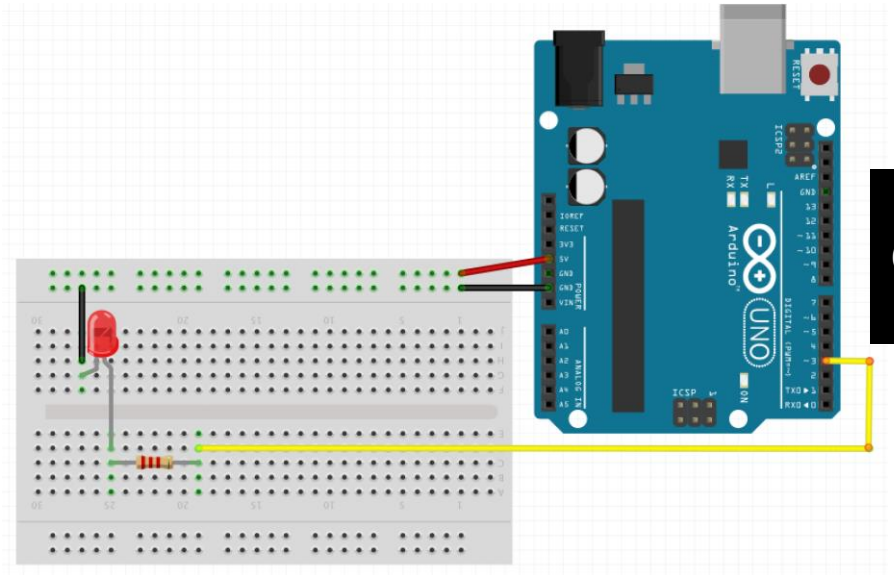
New Circuit: Fade LED on/off via pot value

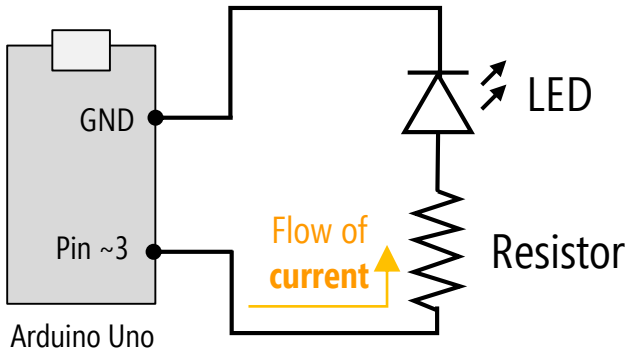




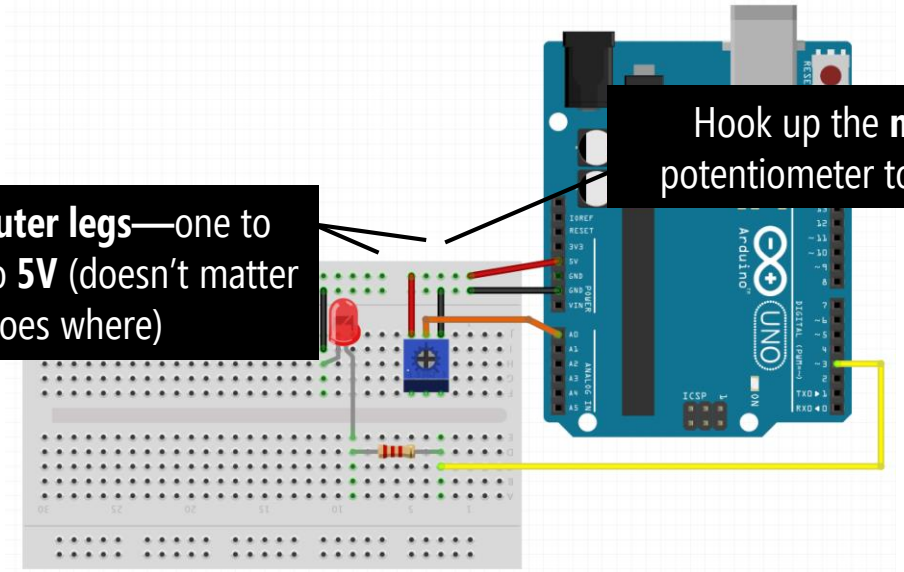
ACTIVITY: CONTROL LED BRIGHTNESS WITH POTENTIOMETER

Old Circuit: Fade LED on/off via pin 3



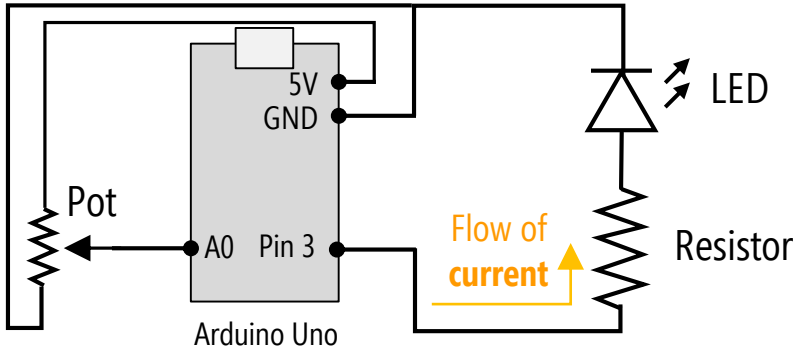


New Circuit: Fade LED on/off via pot value



Hook up the **two outer legs**—one to **GND** and the other to **5V** (doesn't matter which one goes where)

Hook up the **middle leg** of the potentiometer to **A0** on **Analog In**.



ANALOG INPUT

int analogRead()

Reads the voltage from the specified analog pin & returns an integer (0 – 1023)

Syntax

`analogRead(pin)`

Parameters

`pin`: the analog pin to read from

Returns

An integer between 0 and 1023 (inclusive).

TrimpotLED

```
// The Arduino Uno ADC is 10 bits (thus, 0 - 1023 values)
#define MAX_ANALOG_INPUT_VAL 1023

const int LED_OUTPUT_PIN = 3;
const int POT_INPUT_PIN = A0;

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(POT_INPUT_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {

  // read the potentiometer value
  int potVal = analogRead(POT_INPUT_PIN);

  // the analogRead on the Arduino Uno goes from 0 to 1023. We need to remap
  // this value to the smaller range (0-255) since the analogWrite function can
  // only write out 0-255 (a byte--2^8). The map function provides a linear
  // mapping to do this (however, a better way would likely be some sort of
  // non-linear mapping given that perceived LED brightness is not linear with current,
  // perhaps logarithmic)
  int ledVal = map(potVal, 0, MAX_ANALOG_INPUT_VAL, 0, 255);

  // print the raw pot value and the converted led value
  Serial.print(potVal);
  Serial.print(",");
  Serial.println(ledVal);

  // write out the LED value. This value is translated to voltage by:
  // voltageVal = max_voltage * val/255 or voltageVal = 3.3V * val/255 in
  // the case of the RedBear Duo
  analogWrite(LED_OUTPUT_PIN, ledVal);

  delay(100);
}
```

https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

ANALOG INPUT

int analogRead()

Reads the voltage value from the specified analog pin and returns it as an integer (0 – 1023)

Syntax

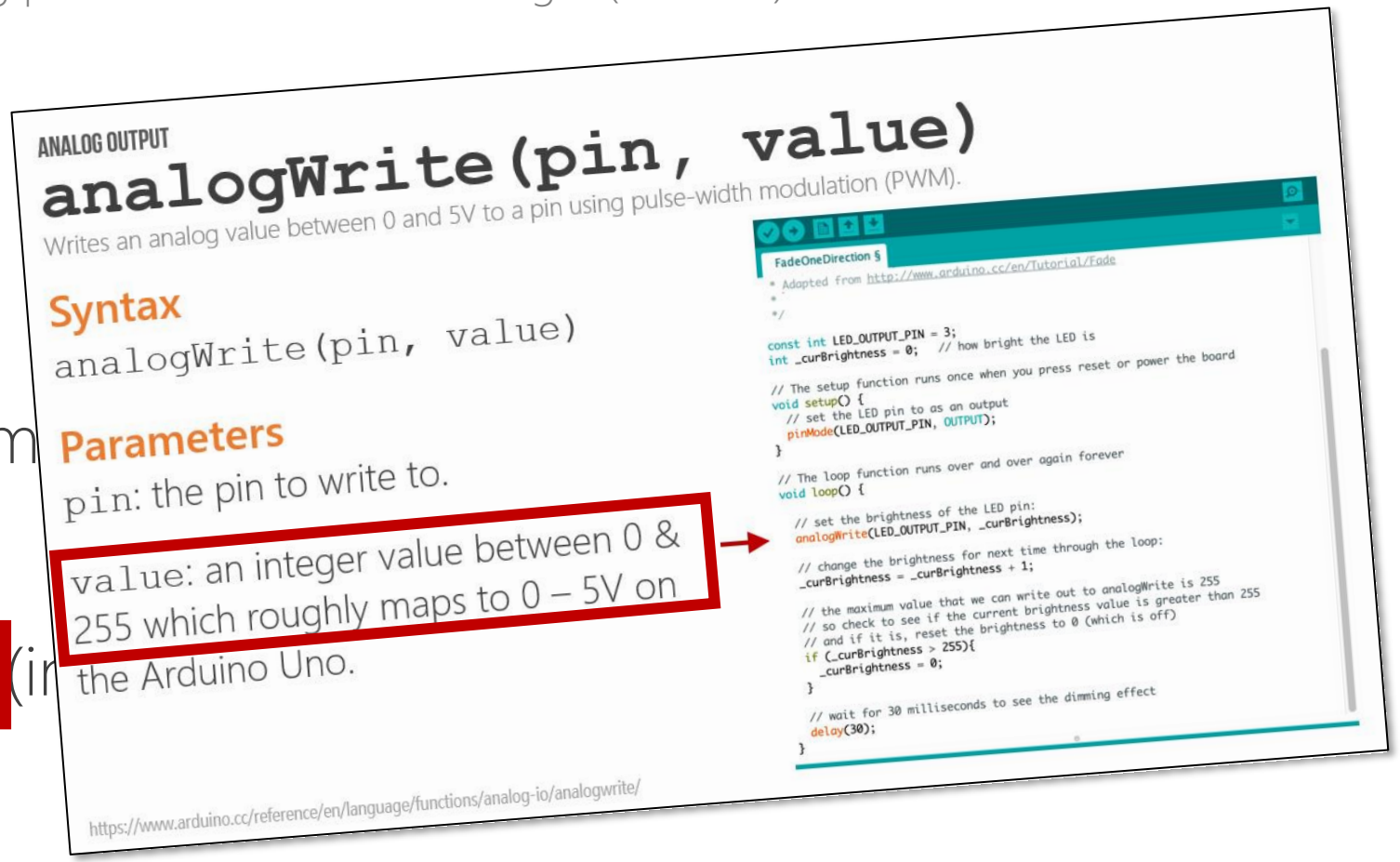
`analogRead(pin)`

Parameters

`pin`: the analog pin to read from

Returns

An integer between 0 and 1023 (in



https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

HELPER FUNCTION

map(value, fromLow, fromHigh, toLow, toHigh)

Remaps a number from one range to another. Warning: it does not constrain values to be within the range.

Syntax

`map(value, fromLow, fromHigh, toLow, toHigh)`

Parameters

`value`: the number to map

`fromLow`: the lower bound of the value's current range

`fromHigh`: the upper bound of the value's current range

`toLow`: the lower bound of the value's target range

`toHigh`: the upper bound of the value's target range

Returns

The mapped value

TrimpotLED

```
// The Arduino Uno ADC is 10 bits (thus, 0 - 1023 values)
#define MAX_ANALOG_INPUT_VAL 1023

const int LED_OUTPUT_PIN = 3;
const int POT_INPUT_PIN = A0;

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(POT_INPUT_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {

  // read the potentiometer value
  int potVal = analogRead(POT_INPUT_PIN);

  // the analogRead on the Arduino Uno goes from 0 to 1023. We need to remap
  // this value to the smaller range (0-255) since the analogWrite function can
  // only write out 0-255 (a byte--2^8). The map function provides a linear
  // mapping to do this (however, a better way would likely be some sort of
  // non-linear mapping given that perceived LED brightness is not linear with current,
  // perhaps logarithmic)
  int ledVal = map(potVal, 0, MAX_ANALOG_INPUT_VAL, 0, 255);

  // print the raw pot value and the converted led value
  Serial.print(potVal);
  Serial.print(",");
  Serial.println(ledVal);

  // write out the LED value. This value is translated to voltage by:
  // voltageVal = max_voltage * val/255 or voltageVal = 3.3V * val/255 in
  // the case of the RedBear Duo
  analogWrite(LED_OUTPUT_PIN, ledVal);

  delay(100);
}
```

https://www.arduino.cc/reference/en/language/functions/math/map/

HELPER FUNCTION

map(value, fromLow, fromHigh, toLow, toHigh)

Remaps a number from one range to another. Warning: it does not constrain values to be within the range.

Syntax

`map(value, fromLow, fromHigh, toLow, toHigh)`

Parameters

`value`: the number to map

`fromLow`: the lower bound of the value's current range

`fromHigh`: the upper bound of the value's current range

`toLow`: the lower bound of the value's target range

`toHigh`: the upper bound of the value's target range

Returns

The mapped value

There is no magic here. The full utility function is simply:

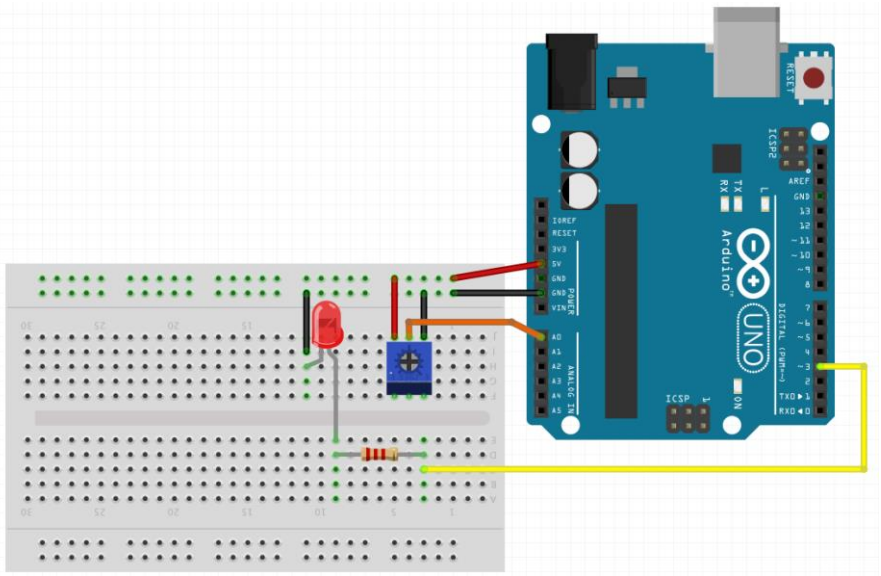
```
long map(long val, long fromLow, long fromHigh, long toLow, long toHigh)
{
  float rangeFraction = (toHigh - toLow) / (fromHigh - fromLow);
  return toLow + (x - fromLow) * rangeFraction;
}
```

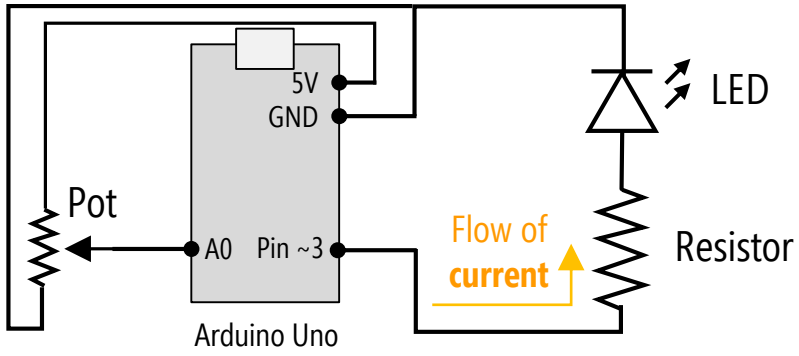
https://www.arduino.cc/reference/en/language/functions/math/map/

ANALOG INPUT

CONTROL LED BRIGHTNESS WITH POTENTIOMETER

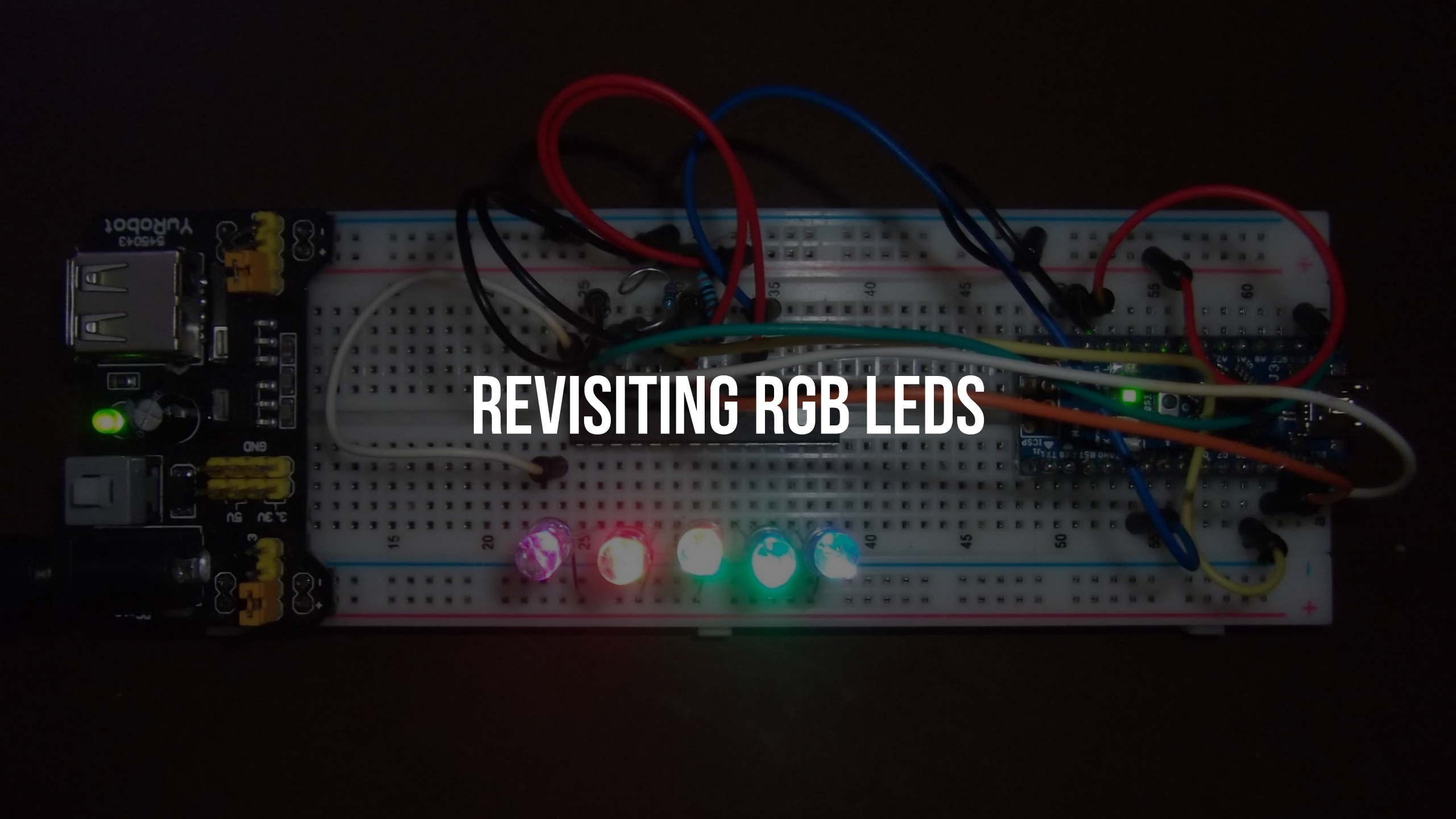
Circuit: Fade LED on/off via pot value





Code: Fade LED on/off via pot value

```
// The Arduino Uno ADC is 10 bits (thus, 0 - 1023 values)
#define MAX_ANALOG_INPUT_VAL 1023

const int LED_OUTPUT_PIN = 3;
const int POT_INPUT_PIN = A0;

void setup() {
  pinMode(LED_OUTPUT_PIN, OUTPUT);
  pinMode(POT_INPUT_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {

  // read the potentiometer value
  int potVal = analogRead(POT_INPUT_PIN);

  // the analogRead on the Arduino Uno goes from 0 to 1023. We need to remap
  // this value to the smaller range (0-255) since the analogWrite function can
  // only write out 0-255 (a byte--2^8). The map function provides a linear
  // mapping to do this (however, a better way would likely be some sort of
  // non-linear mapping given that perceived LED brightness is not linear with current,
  // perhaps logarithmic)
  int ledVal = map(potVal, 0, MAX_ANALOG_INPUT_VAL, 0, 255);

  // print the raw pot value and the converted led value
  Serial.print(potVal);
  Serial.print(",");
  Serial.println(ledVal);

  // write out the LED value. This value is translated to voltage by:
  // voltageVal = max_voltage * val/255 or voltageVal = 3.3V * val/255 in
  // the case of the RedBear Duo
  analogWrite(LED_OUTPUT_PIN, ledVal);

  delay(100);
}
```


REVISITING RGB LEDS

Diffused RGB (tri-color) LED - Common Anode

PRODUCT ID: 159

$2.00

49 IN STOCK

1 ADD TO CART

QTY	DISCOUNT
1-9	$2.00
10-49	$1.75
50+	$1.50

ADD TO WISHLIST

adafruit SHOP BLOG LEARN FORUMS VIDEOS ADABOX

LEDS / BARE LEDS / DIFFUSED RGB (TRI-COLOR) LED

Diffused RGB (tri-color) LED - Common Anode

PRODUCT ID: 159

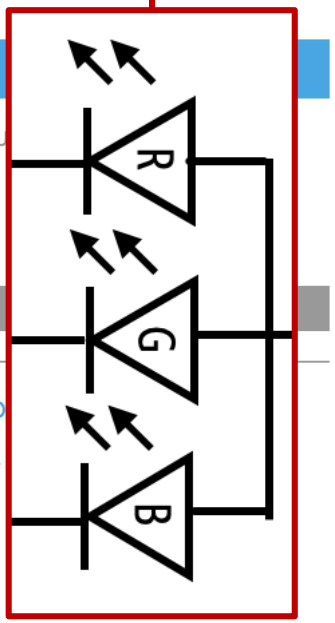
$2.00

49 IN STOCK

1

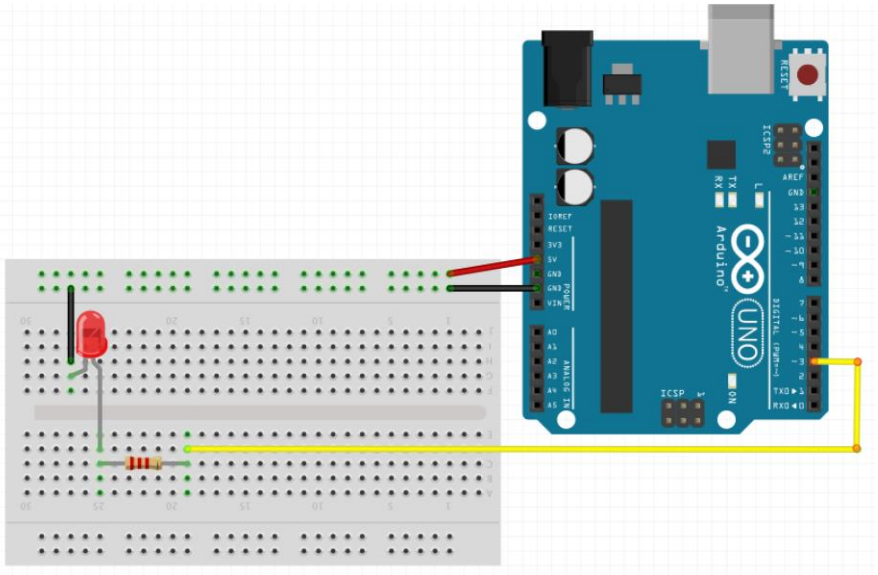
QTY DISCOU
1-9 $2.00
10-49 $1.75
50+ $1.50

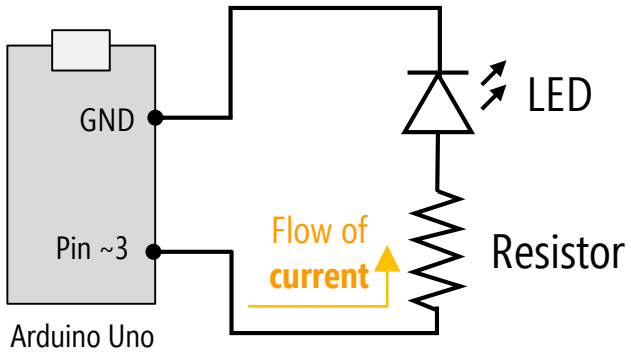
DESCRIPTIO
TECHNICAL
LEARN



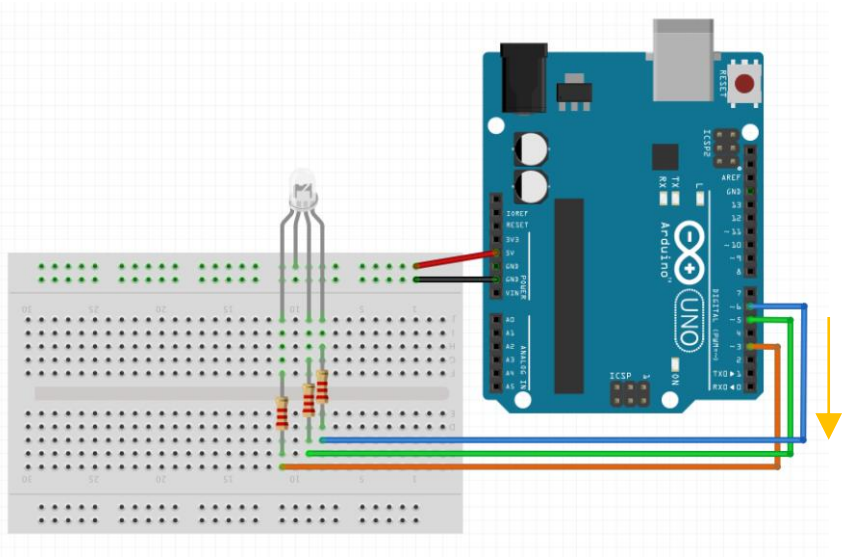
USING AN RGB LED: THE CIRCUIT

Old Circuit: Fade LED on/off via pin 3

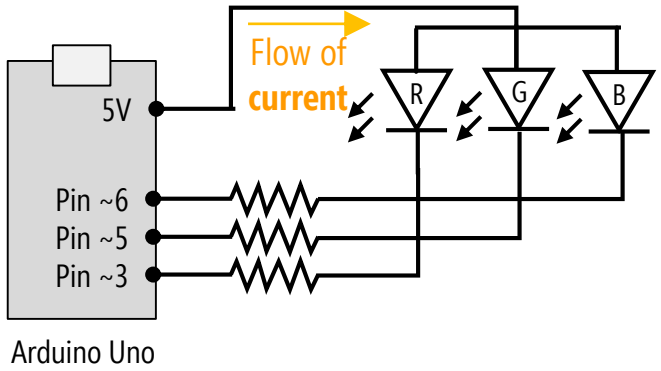




New Circuit: Set R, B, G values via pin 6, 5, and 3

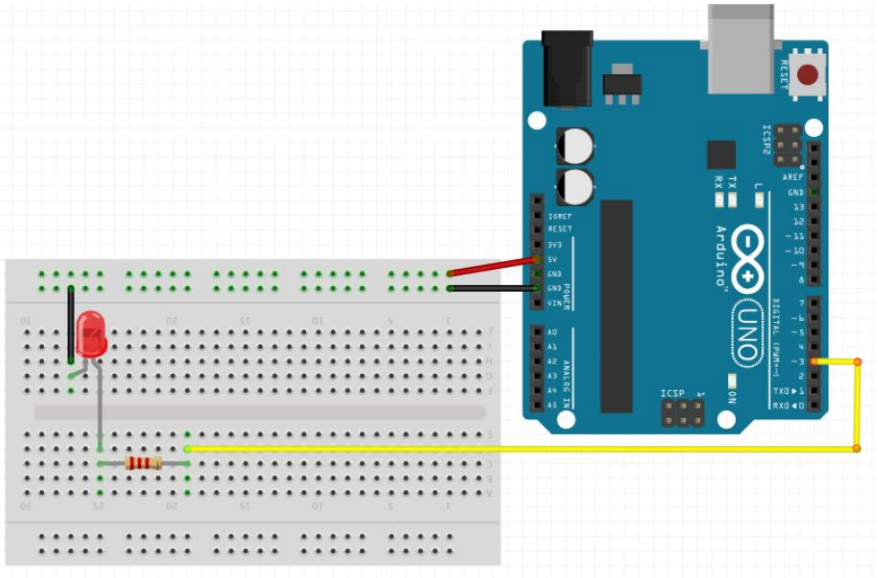


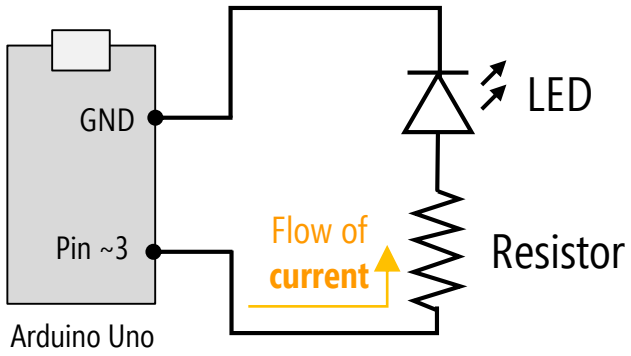
Flow of current
Because this is a "common anode" RGB. We set these pins to zero to turn on the light and 5V to turn off the light!



USING AN RGB LED: THE CIRCUIT

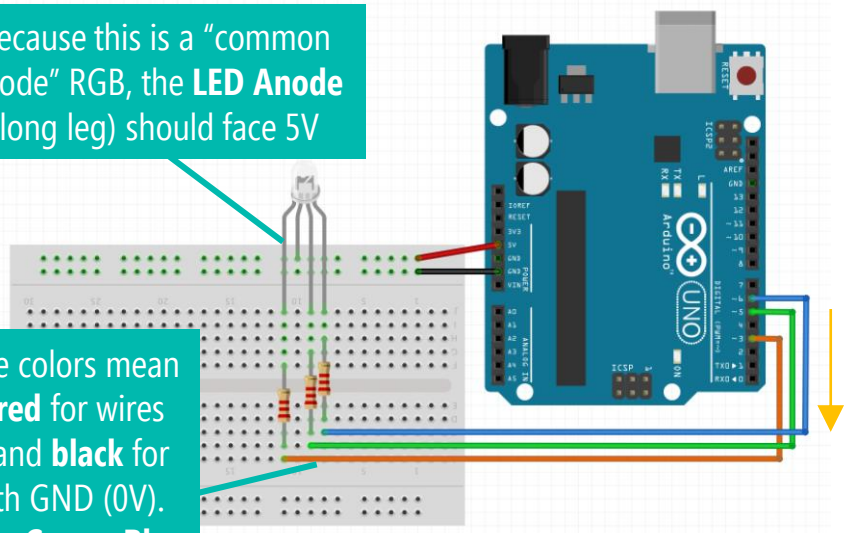
Old Circuit: Fade LED on/off via pin 3





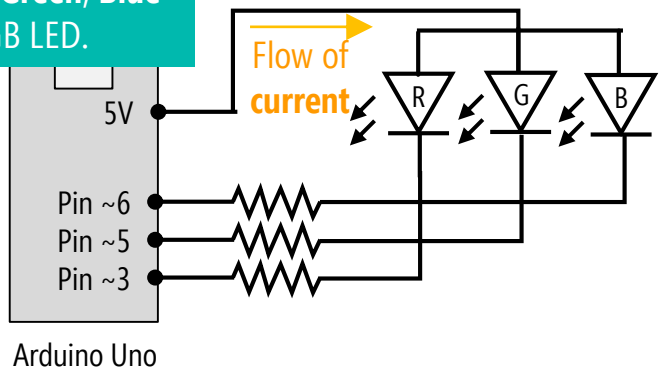
New Circuit: Set R, B, G values via pin 6, 5, and 3

Because this is a "common anode" RGB, the **LED Anode** (long leg) should face 5V



Pro tip: make your wire colors mean something. Only use **red** for wires that connect with 5V and **black** for wires that connect with GND (0V). Here, I'm using **Orange**, **Green**, **Blue** to represent the **Red**, **Green**, **Blue** signal for my RGB LED.

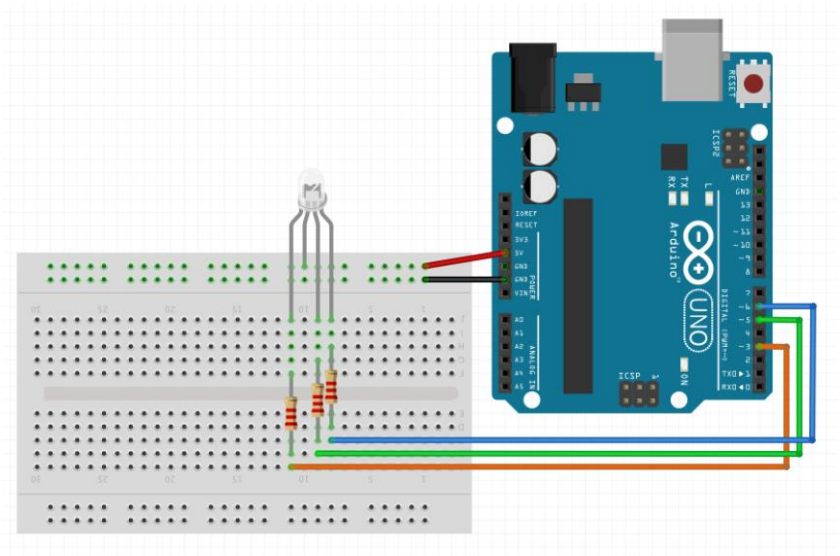
Flow of current
Because this is a "common anode" RGB. We set these pins to zero to turn on the light and 5V to turn off the light!

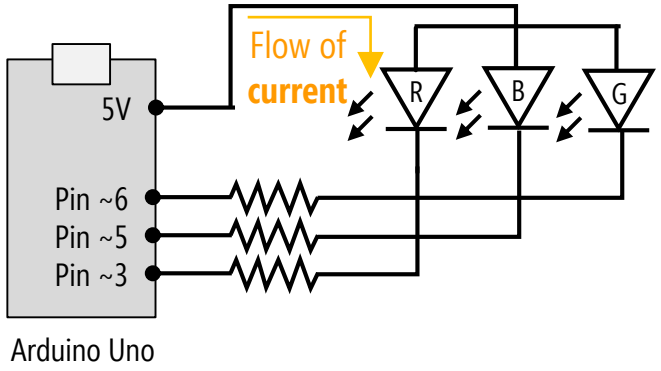


ANALOG OUTPUT

USING AN RGB LED: THE CODE

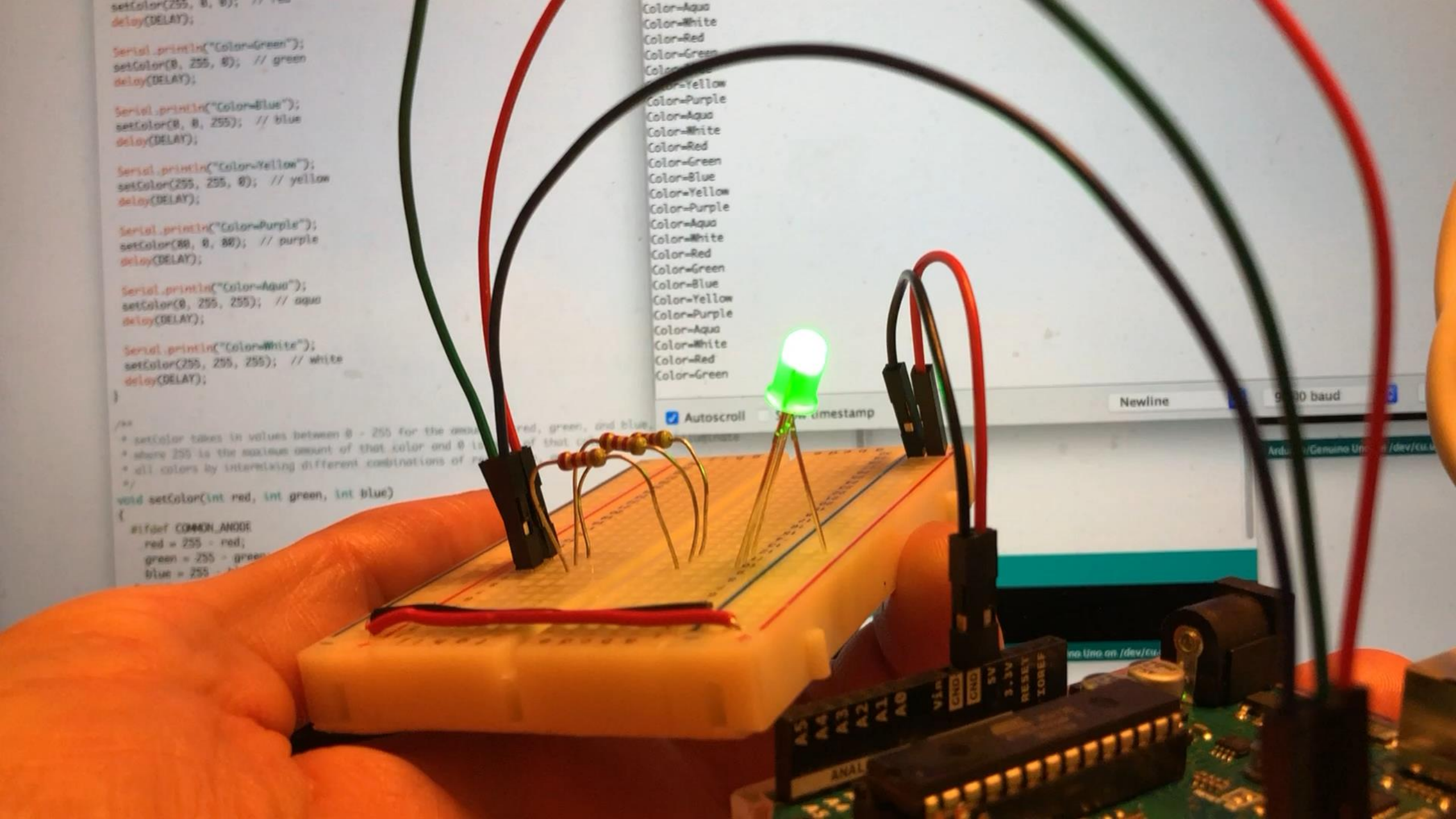
Circuit: Set R, B, G values via pin 6, 5, and 3





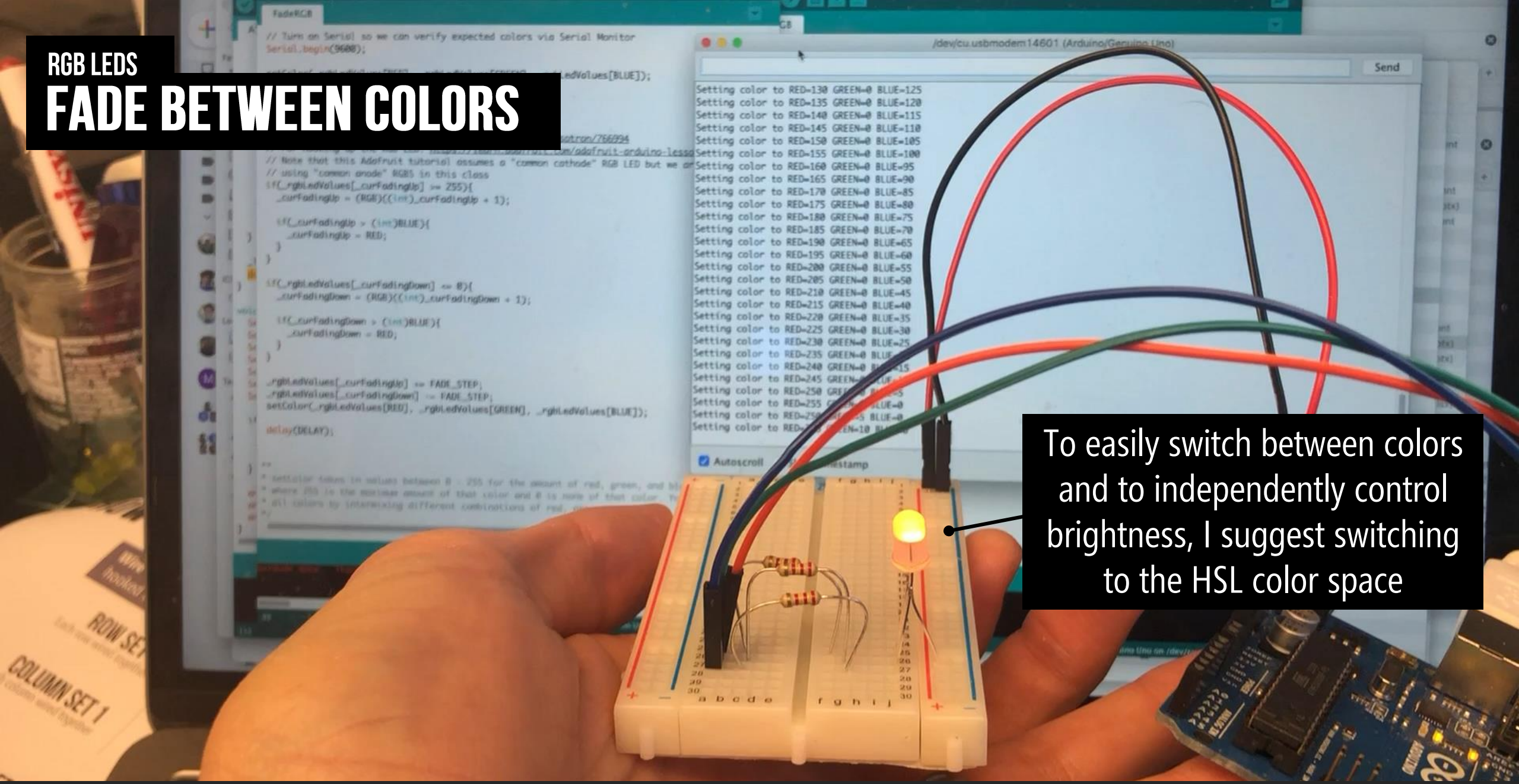
BlinkRGBSimple

```
const int RGB_RED_PIN = 6;
const int RGB_GREEN_PIN  = 5;
const int RGB_BLUE_PIN  = 3;
const int DELAY = 1000; // delay in ms between changing colors

void setup() {
  // Set the RGB pins to output
  pinMode(RGB_RED_PIN, OUTPUT);
  pinMode(RGB_GREEN_PIN, OUTPUT);
  pinMode(RGB_BLUE_PIN, OUTPUT);
}

void loop() {

  // Because the RGB LED we purchased for the class is a 'common anode'
  // RGB, the way we turn on each light is counter to our intuition
  // We set a pin to 0 to turn on the corresponding LED to its maximum brightness
  // and 255 to turn it off (the opposite of what you might think)

  // Set the RGB LED to red
  analogWrite(RGB_RED_PIN, 0);      // turn on the red LED
  analogWrite(RGB_GREEN_PIN, 255);  // turn off the green LED
  analogWrite(RGB_BLUE_PIN, 255);   // turn off the blue LED
  delay(DELAY);

  // Set the RGB LED to green
  analogWrite(RGB_RED_PIN, 255);    // turn off the red LED
  analogWrite(RGB_GREEN_PIN, 0);    // turn on the green LED
  analogWrite(RGB_BLUE_PIN, 255);     // turn on the blue LED
  delay(DELAY);

  // Set the RGB LED to blue
  analogWrite(RGB_RED_PIN, 255);    // turn off the red LED
  analogWrite(RGB_GREEN_PIN, 255);  // turn off the green LED
  analogWrite(RGB_BLUE_PIN, 0);     // turn on the blue LED
  delay(DELAY);

  // Set the RGB LED to yellow (by turning on green and blue!)
  analogWrite(RGB_RED_PIN, 255);    // turn off the red LED
  analogWrite(RGB_GREEN_PIN, 0);    // turn on the green LED
  analogWrite(RGB_BLUE_PIN, 0);     // turn on the blue LED
  delay(DELAY);

  // Set the RGB LED to purple (by turning on red and blue!)
  analogWrite(RGB_RED_PIN, 0);      // turn on the red LED
  analogWrite(RGB_GREEN_PIN, 255);  // turn off the green LED
  analogWrite(RGB_BLUE_PIN, 0);     // turn on the blue LED
  delay(DELAY);
}
```


Color=Aqua
Color=White
Color=Red
Color=Green
Color=Yellow
Color=Purple
Color=Aqua
Color=White
Color=Red
Color=Green
Color=Blue
Color=Yellow
Color=Purple
Color=Aqua
Color=White
Color=Red
Color=Green
Color=Blue
Color=Yellow
Color=Purple
Color=Aqua
Color=White
Color=Red
Color=Green
Autoscroll
Newline
baud
A0
A1
A2
A3
A4
A5
VIN
GND
GND
5V
3.3V
RESET
IOREF
ANAL

RGB LEDS

FADE BETWEEN COLORS



Video: Jon Froehlich

PHYSICAL COMPUTING 3

TODAY'S LEARNING GOALS

What is **analog output?**

What is **PWM** and why does it matter?

How to use **analogWrite()**

Using **Serial Plotter** for debugging

Introduction to **vibration motors** and how to use them

(Partial) introduction to **potentiometers** and **analog input** (if time)

PHYSICAL COMPUTING 3: ANALOG OUTPUT

CSE 599 Prototyping Interactive Systems | Lecture 4 | April 11

Jon Froehlich • Jasper O'Leary (TA)

DESIGN USE BUILD

MAKEABILITY LAB

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

UNIVERSITY of WASHINGTON