INSTRUCTIONAL LABORATORIES AND DEMONSTRATIONS | OCTOBER 01 2024

Flipping the electronics lab: Learning upper division electronics at home

Brian Rasnow

Am. J. Phys. 92, 809–818 (2024)
https://doi.org/10.1119/5.0206534

John Essick, *Editor*
Department of Physics, Reed College, Portland, OR 97202

Articles in this section deal with new ideas and techniques for instructional laboratory experiments, for demonstrations, and for equipment that can be used in either. Although these facets of instruction also appear in regular articles, this section is for papers that primarily focus on equipment, materials, and how they are used in instruction. Manuscripts should be submitted using the web-based system that can be accessed via the American Journal of Physics home page, ajp.aapt.org, and will be forwarded to the IL&D editor for consideration.

Flipping the electronics lab: Learning upper division electronics at home

Brian Rasnow[a)]
Departments of Physics and Mechatronics, California State University, Channel Islands, Camarillo, California 93012

(Received 2 March 2024; accepted 18 July 2024)

A junior-level four-unit electronics curriculum is described in which students learn by building their own automated electronic test equipment and performing a rigorous suite of electronics experiments outside the classroom. Each student builds an electronics "workbench" based on an Arduino Nano microcontroller under the control of a software scripting environment (in our case, MATLAB). In the course of their work, students (1) construct a DC power supply and function generator and iteratively add programmable serial interfaces; (2) create an oscilloscope and spectrum analyzer that provides real-time displays of two analog channels for measurement of AC voltage amplitudes, frequency, and phase difference; (3) automate measurements of device *I–V* curves and transistor gain; (4) design, build, and characterize audio amplifiers and filters through measurement of time constants and frequency responses; and (5) compare the measured and simulated Bode plots. Mixing unconventional topics such as automation, numerical simulation, and programming with basic electronics removes the drudgery of data collection and provides more exposure, repetition, and organization of key concepts. Without being restricted to making measurements within the imposed lab schedule, students can work where and when they are able, repeat automated experiments in seconds, and continue exploring electronics with their workbench. After completing this course, our students demonstrate more confidence and capability with integrating electronics, automation, and programming into other projects and in improving existing systems. Likewise, students glean hands-on experience with scripting repetitive activities, saving enormous effort in their data-gathering work and leaving more time for analysis and creativity. © *2024 Published under an exclusive license by American Association of Physics Teachers.*

https://doi.org/10.1119/5.0206534

I. INTRODUCTION

"Active learning"[1] facilitates the teaching of electronics in areas such as drawing schematics, building circuits, measuring device characteristics, and comparing measurements and theory. Compressing these activities into 2- or 3-h laboratory periods invariably limits deeper exploration for many students. Here, we present a novel four-unit junior-level electronics curriculum, motivated by the COVID lockdown, where students take the electronics laboratory home and conduct a sophisticated suite of instructional electronics activities using a ∼$120 kit, along with their computer. Rather than remote learning reducing content for lack of test equipment, we've expanded content to include programming an Arduino in C and programming MATLAB for laboratory automation, simulation, and digital signal processing in both time and frequency domains. Many of our labs involve the construction of persistent test equipment hardware and software that are subsequently used in the rest of the course (and hopefully afterward), including a programmable DC power supply and function generator, oscilloscope, and spectrum analyzer.

The kits supplied to students include a digital multimeter (DMM), soldering kit, electronics "starter kit" containing resistors, capacitors, transistors, LEDs, solderless breadboard, wires, photoresistor, augmented with a microphone, op-amp, LM317 voltage regulator, perforated circuit board, 12 V DC adapter, Arduino Nano, and basic tools (see supplementary material Table 1). The Arduino software is freely available online,[2] and basic MATLAB[3] software was provided through an academic site license. With minor modifications, much of the software can run under open-source

Octave,[4] and, in principle, be modified for Python. Further details are in the supplementary material.

Most of our junior-level physics students' prior exposure to electronics is basic circuit theory in prerequisite sophomore general physics courses, and often they have no solderless breadboarding, soldering, or prior programming experiences. In our course, activities are demonstrated on a projected computer screen or Zoom, displaying software adjacent to live video from a webcam equipped with a $10\times$ zoom lens[5] mounted 18″ above the desk. The entire computer screen, recorded on Zoom [Fig. 1], shows detailed breadboarding, soldering, troubleshooting, and interactions between software and hardware. At minimum magnification, the camera's field of view covers an $8.5\times 11''$ stack of paper, serving as an analog "white board" for freehand sketching and problem solving. Most students step through the recorded videos to replicate the required hardware and software steps.

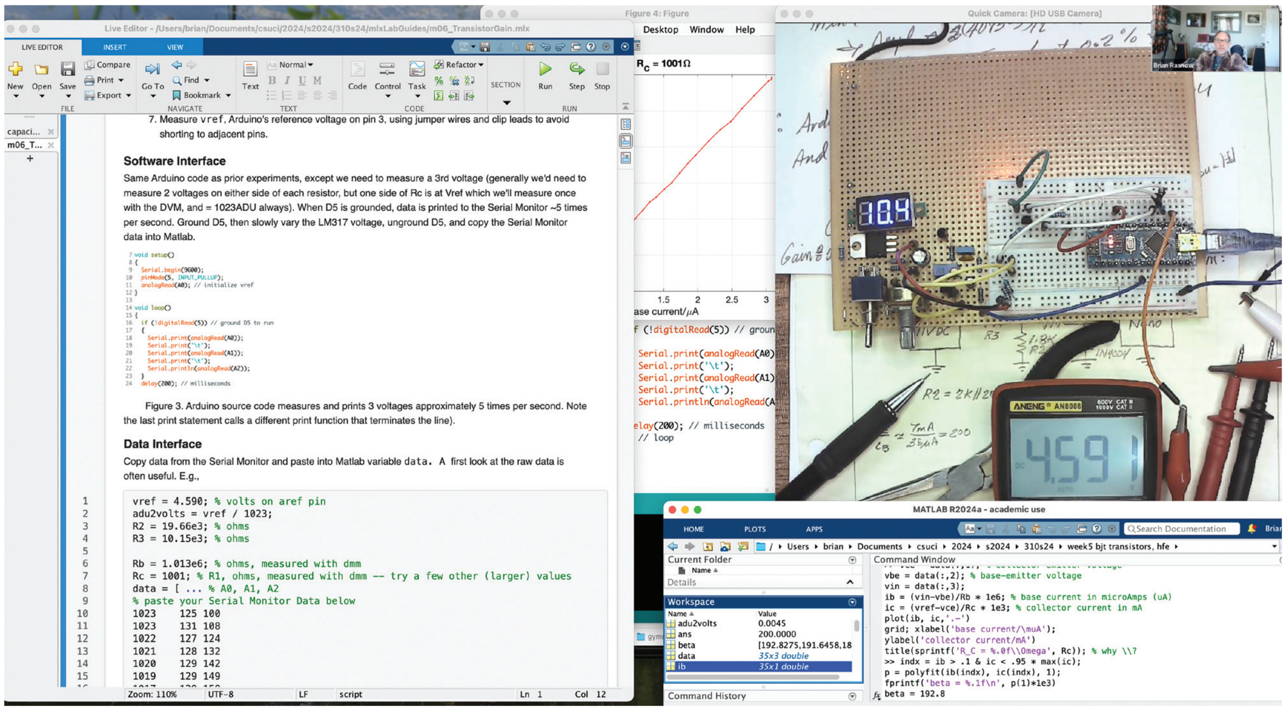
Motivation, theory, and instructions for the lab activities are provided in the format of 16 MATLAB Live Script[6] files. Similar to Jupyter,[7] this file format combines text, images, LaTeX, MATLAB source code, and "live" results from execution of the source code (text, numbers, and graphics) together in one document. Students are provided explanations of the short (1–20 lines) blocks of source code that they have to modify and execute, often *in situ* in the Live Script documents. MATLAB source code is more similar to natural language or pseudocode than most programming languages. Short and simple C programs were also provided to upload into the Arduino. The programs are *readable* and enticingly understandable without requiring the students to develop the knowledge base necessary to *write* source code from scratch. Many code examples are included herein.

During the COVID lockdown, the course was conducted synchronously online and recorded. Presently, most of our students have returned to the classroom, bringing their kits with them; however, some of our students still choose to take the course remotely and rely exclusively on the videos and remote interaction. Interaction with peers is encouraged and strongly correlates with success. Fully "flipping the labs" means that students watch the videos as homework and use class time to discuss (and debug) the activities. Class time is usually a mix of the professor demonstrating the activities and helping students troubleshoot and understand their own circuits by sharing their screen (if remote) or bringing their circuit board beneath the professor's camera.

Developing technical writing skills is emphasized by requiring students to submit lab reports describing their work on five of the 16 activities. The provided Live Script files serve as templates for these reports. Students delete general theory sections, replace sample data with their own, execute and debug the analyses and results, and interpret and reflect on what they have learned. As part of their final grade, students correct and assemble their reports into a pdf portfolio that serves as a record of their accomplishments and also documents their experimental apparatus, which they keep. By assembling, programming, debugging, calibrating, and documenting the apparatus, students became more intimately familiar with their schematics, IC datasheets, and source codes and are more likely to use and modify them later.

II. INTRODUCTORY LABS

The first activity is an introduction to MATLAB, based on numeric explorations of voltage, current, and power in a simulated circuit consisting of a Thevenin voltage source[8] and resistor. Students learn MATLAB syntax for making, multiplying, and plotting vectors on linear and log scales (see supplementary material "Voltage and Current Sources"). The second lab is a hands-on introduction to electronic engineering, with the explicit goal of modifying a DC adapter into an LED nightlight. Students are shown how to cut and solder



Live Editor - /Users/brian/Documents/csuci/2024/s2024/310s24/mlxLabGuides/m06_TransistorGain.mlx
LIVE EDITOR
INSERT
VIEW
FILE
NAVIGATE
TEXT
CODE
RUN
Figure 4: Figure
R_C = 1001Ω
1.5
2
2.5
3
ase current/μA
Quick Camera: [HD USB Camera]
MATLAB R2024a - academic use
HOME
PLOTS
APPS
Search Documentation
/ ▸ Users ▸ brian ▸ Documents ▸ csuci ▸ 2024 ▸ s2024 ▸ 310s24 ▸ week5 bjt transistors, hfe ▸
Current Folder
Workspace
Command Window
Command History
Zoom: 110%
UTF-8
LF
script
Ln 1
Col 12

7. Measure `vref`, Arduino's reference voltage on pin 3, using jumper wires and clip leads to avoid shorting to adjacent pins.

Software Interface

Same Arduino code as prior experiments, except we need to measure a 3rd voltage (generally we'd need to measure 2 voltages on either side of each resistor, but one side of Rc is at Vref which we'll measure once with the DVM, and = 1023ADU always). When D5 is grounded, data is printed to the Serial Monitor ~5 times per second. Ground D5, then slowly vary the LM317 voltage, unground D5, and copy the Serial Monitor data into Matlab.

```
void setup()
{
  Serial.begin(9600);
  pinMode(5, INPUT_PULLUP);
  analogRead(A0); // initialize vref
}

void loop()
{
  if (!digitalRead(5)) // ground D5 to run
  {
    Serial.print(analogRead(A0));
    Serial.print('\t');
    Serial.print(analogRead(A1));
    Serial.print('\t');
    Serial.println(analogRead(A2));
  }
  delay(200); // milliseconds
```

Figure 3. Arduino source code measures and prints 3 voltages approximately 5 times per second. Note the last print statement calls a different print function that terminates the line).

Data Interface

Copy data from the Serial Monitor and paste into Matlab variable `data`. A first look at the raw data is often useful. E.g.,

```
vref = 4.590; % volts on aref pin
adu2volts = vref / 1023;
R2 = 19.66e3; % ohms
R3 = 10.15e3; % ohms

Rb = 1.013e6; % ohms, measured with dmm
Rc = 1001; % R1, ohms, measured with dmm -- try a few other (larger) values
data = [ ... % A0, A1, A2
% paste your Serial Monitor Data below
1023    125 100
1023    131 108
1022    127 124
1021    128 132
1020    129 142
1019    129 149
```

```
f (!digitalRead(5)) // groun

 Serial.print(analogRead(A0)
 Serial.print('\t');
 Serial.print(analogRead(A1)
 Serial.print('\t');
 Serial.println(analogRead(A

elay(200); // milliseconds
 // loop
```

Name	Value
adu2volts	0.0045
ans	200.0000
beta	[192.8275,191.6458,18
data	35x3 double
ib	35x1 double

```
vbe = data(:,2); % base-emitter voltage
vin = data(:,3);
ib = (vin-vbe)/Rb * 1e6; % base current in microAmps (uA)
ic = (vref-vce)/Rc * 1e3; % collector current in mA
plot(ib, ic,'.-')
grid; xlabel('base current/\muA');
ylabel('collector current/mA')
title(sprintf('R_C = %.0f\\Omega', Rc)); % why \\?
>> indx = ib > .1 & ic < .95 * max(ic);
p = polyfit(ib(indx), ic(indx), 1);
fprintf('beta = %.1f\n', p(1)*1e3)
beta = 192.8
```

Fig. 1. Typical screen shot of the class recorded on Zoom, showing MATLAB's Live Editor (left), Command Window (bottom right), Arduino IDE (center in background), and live video image of a circuit under test on the desk (right).

wires, use solderless breadboards and their DMM, measure the Thevenin voltage and resistance of their DC adapter, measure the *I–V* curve of an LED, and finally design, prototype, solder, and heat-shrink their nightlight.

The challenge of measuring an *I–V* curve motivates the next lab in which students build a variable DC power supply using an LM317 integrated circuit (IC).[9] We repeat a pattern of theory, design, prototype on solderless breadboard, solder the circuit (this time on a perforated board), debug, validate, and document. New concepts include reading device datasheets, resolving ergonomic and mechanical considerations, and mounting five LEDs *underneath* the board to function as feet or stand-offs, which most students also solder together in series to illuminate when the power is on [Fig. 2(a)].

The next activity introduces the Arduino Nano microcontroller and its integrated development environment (IDE).[2] A $<$ \$10 Arduino can be programmed to rapidly measure and log voltages at multiple locations in a circuit, eliminating two common sources of data errors: moving the DMM wires between measurements and manually transcribing readings from the DMM's display. However, using an Arduino requires protecting its sensitive inputs from overvoltages, programming it to measure and communicate through its serial port, and then "unpacking" and interpreting the serial data into results and conclusions. These steps are made explicit in a design pattern that is repeated in each subsequent lab:

(1) Construct a *hardware interface* between the Arduino and circuit or device under test. Voltage dividers and current-limiting resistors protect the Arduino from voltages $>$5 V.
(2) Write a *software interface*, a short Arduino program that queries the hardware interface and returns data via the serial port. We first copy-and-paste data from the Arduino IDE into MATLAB. Later, we develop a fast bidirectional serial connection to support oscilloscope and spectral analysis functions and software voltage and frequency control.
(3) Write a *data interface* to process and visualize the copious amount of Arduino data. Automated data acquisition radically lowers the cost of collecting data, necessitating automated digital signal processing and visualization, programmed in MATLAB.

Students recognize that the higher complexity of our data-gathering approach is worth the radical reduction of drudgery resulting from not having to manually record and process data. Entire experiments can be repeated in mere seconds once these interfaces are working; hence, students no longer find themselves stuck with problematic data. Some students begin to realize how MATLAB can be more than just a lens to visualize data when they transition from running code in the Live Script to initiating dialogs at the command line to troubleshoot their system. For example, if noise is a suspected problem, the command, `std(data)` displays the standard deviation. Another question might be, "Did the analog-to-digital converter (ADC) saturate (clip)?" The answer is returned by `max(data) == 1023`. The built-in, extensible help system is readily accessible with the commands `doc`, `help`, and `lookfor`, to explain functions and syntax. Once their system is working, the relative ease with which it produces gorgeous results motivates students to persevere and learn more.

III. TRANSISTORS

Next, we study bipolar junction transistors (BJTs) followed by MOSFETs using the hardware interface in Fig. 2(b). R2 and R3 divide the LM317 output voltage by a factor of 3, and R5 and R6 limit current to Arduino's sensitive inputs in the event of a wiring error. The corresponding software interface programmed into the Arduino simply reads and prints the three analog input voltages five times per second when pin D5 is grounded. Students slowly turn the potentiometer on their power supply, then copy-and-paste their data from the Arduino IDE into the Live Script file and run it. The data interface converts analog-to-digital units (ADU) to volts, computes base and collector currents with Ohm's law, plots the currents, and least-squares-fits the current gain β from the slope as follows:

```
data = [... % A0, A1, A2
% paste your Serial Monitor Data below,
1023 131 108
1022 127 124
...
] *vref/1023; % - > volts
vce = data(:,1); % collector-emitter voltage
vbe = data(:,2); % base-emitter voltage
vin = data(:,3);
ib = (vin - vbe)/Rb; % base current
ic = (vref - vce)/Rc; % collector current
plot(ib, ic)
p = polyfit(ib, ic, 1); % linear least squares fit
fprintf('beta = %.1f\n', p(1))
```

Repeating this measurement with different collector resistances quickly produces curves like Fig. 2(c) without repetitively moving probes or tediously combining data from multiple meters. Automated data logging lets students focus on the transistor's characteristic behaviors instead of being

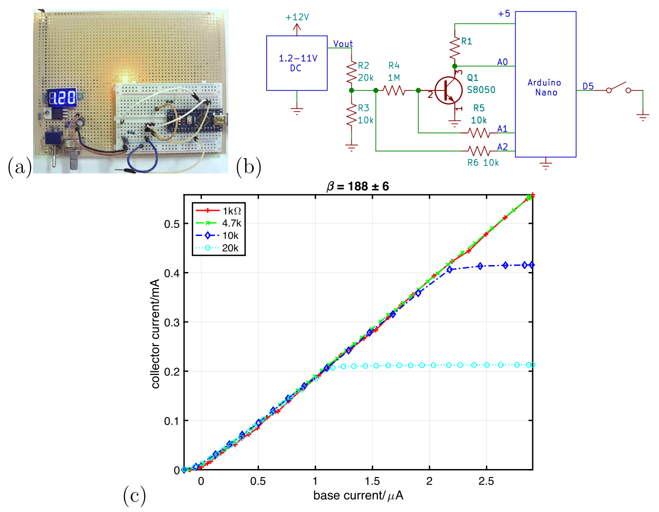


Fig. 2. (Color online) (a) DC power supply constructed on the left side of the circuit board; transistor circuit and Arduino are on the solderless breadboard on the right. LEDs are used as feet (stand-offs) glow beneath the board. (b) Hardware interface is used to measure a transistor's current gain β. (c) Results repeated with four collector resistors (R1). β is the slope.

busy with data collection, reduction, and visualization mechanics. This lab guide is included in the supplementary material.

The next lab activity explores BJT and MOSFET switches and impedances. Plotting power gains further demystifies transistors, because student's comparisons and conclusions are based on the patterns they observe in hundreds of their finely spaced measurements, a procedure that is free of confounding transcription errors, unstable connections, or movement artifacts.

IV. PROGRAMMING

We next focus on developing software to automate the manual copy-and-paste transfer of data between the Arduino IDE and MATLAB. This activity requires understanding serial port interfaces and ASCII vs binary number formats. Although MATLAB provides a versatile but slow Arduino interface,[10] we built a simpler client–server interface that is 500 times faster. Our final ~30-line Arduino program responds to two single-character commands.

After MATLAB establishes a software connection to the Arduino via an `ard=serialport(...)` command, `writeline(ard,'b')` tells the Arduino to measure 400×2 analog voltages on inputs A0 and A1 at Arduino's top measurement speed ~8927 samples/s, and return the data and elapsed time (in microseconds). The second command, invoked from MATLAB with `writeline(ard,'p <pwmValue> ')`, causes Arduino to execute `analogWrite(D5, pwmValue)`, where integer `pwmValue` = 0 – 255 corresponds to pulse width modulation (PWM) on Arduino's pin `D5` of 0%–100%.[11] This enables MATLAB to set the LM317 voltage and our function generator's frequency, via a control circuit described in Sec. VI.

The first client MATLAB script called `oscope1.m` implements a simple oscilloscope display with the following loop (initialization code is omitted for brevity, see the supplementary material):

```
while runBtn.Value
        writeline(ard, 'b');
        bin=read(ard, 802, 'uint16');
        data=reshape(bin(1:800), 2, 400)';
        plot(data, '.-'); grid;
        xlabel('sample #'); ylabel('voltage/ADU')
end
```

While a graphical user interface (GUI) button is checked, the 'b' command is sent to the Arduino, serial data are read and unpacked, and the two voltage time series are plotted, repeatedly many times per second. Subsequent iterations of `oscope1` calibrate the time axis, add a frame-per-second counter (bottom left in Fig. 3(c)), display the spectrum in the frequency domain [Fig. 4(c)], and display amplitudes, frequency, and phase shift of alternating current (AC) signals in the title of the graph. Starting simply, and then incrementally building complexity, gives students repetitive experiences with less tendency to overwhelm.

V. ALTERNATING CURRENT (AC)

The mathematics of complex numbers is built into MATLAB, so AC signals are easily represented as complex

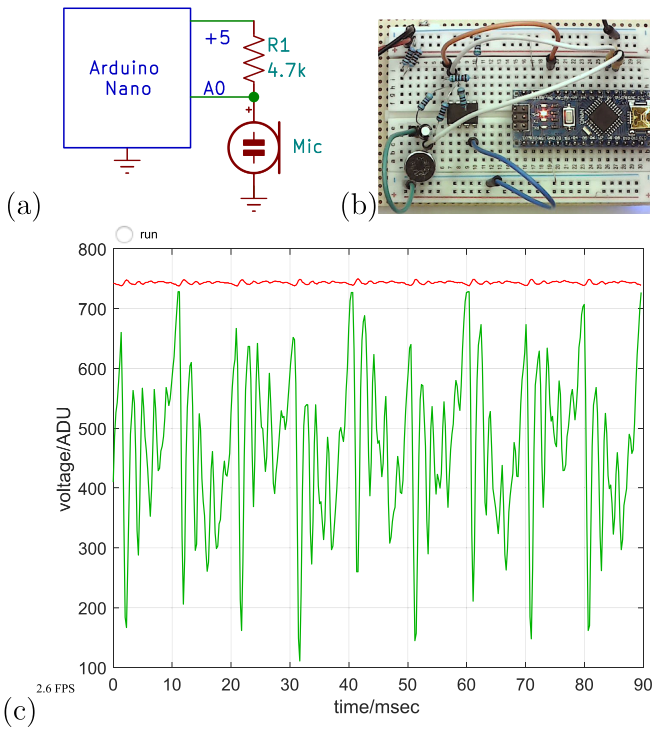


Fig. 3. (Color online) (a) Hardware interface for an electret microphone without amplification produces top trace in (c) when shouting. (b) Op-amp amplifier with AC gain = −50 displayed simultaneously on `oscope`'s second channel [(c), bottom].

numbers (phasors),[12] without the typical drudgery of manual complex number calculations. For example, this function returns the complex impedances of a capacitor at a spectrum of frequencies:

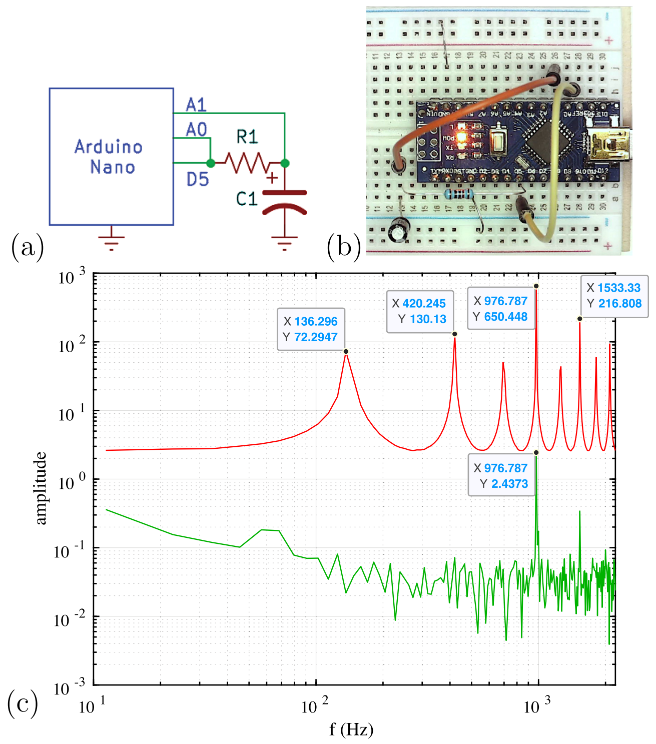


Fig. 4. (Color online) (a) and (b) Hardware interface to explore PWM signals before and after a low pass filter. (c) Spectra (ignoring DC) of 50% duty PWM (top) and filtered with R1 = 10 kΩ, C1 = 4.7 μF (bottom). Most peaks in the top are aliased harmonics of the 977 Hz PWM modulation.

```
function z = zc(farad, hz)
zc = 1./(1j*2*pi*farad*hz); % hz can be a vector
```

Our first measurements of AC voltages are from an electret microphone [Fig. 3(a) and top trace of Fig. 3(c)]. The low signal amplitude motivates learning about amplifiers, leading to an activity in which we build an op-amp amplifier [Fig. 3(b)]. To avoid clipping, the microphone signal is AC coupled and re-biased, using a high-pass filter and voltage divider, respectively. We thus organically apply these concepts, reducing theory to practice. Students greatly enjoy seeing their voices and spectra, as well as exploring sound sources from family, pets, and so on.

The second AC lab examines PWM signals [Fig. 4]. Many students are surprised that the Arduino Nano's `analogWrite` output looks nothing like a DC signal. The voltage flickers rapidly between 0 and 1023 ADU, but `mean(data)` is always four times the 8-bit PWM value. Students are even more surprised that the resistor and capacitor in Fig. 4(a) "compute" the mean voltage.

Discretely sampled time series are prone to aliasing artifacts.[13] Most commercial systems reduce aliasing by sampling at high speeds and/or incorporating anti-aliasing filters. The Arduino does neither, so the top curve in Fig. 4(c) offers a stark example of aliased harmonics of the Arduino's ~ 977 Hz PWM square wave. Disappearance of the lower frequency peaks in the filtered trace of Fig. 4(c) confirms those frequencies aren't really there at all, but are reflected aliased phantoms of higher frequencies. For example, the ninth harmonic appears at $136\,\text{Hz} = 2 \times (\text{sample rate} = 4464\,\text{Hz}) - 9 \times 977\,\text{Hz}$ with measured 72.29 ADU amplitude, in comparison to the theoretical amplitude[14] $= 4/\pi \times (1023\ \text{ADU}/2)/9 = 72.36$ ADU. The ability to measure (discrete harmonic) signals far above the Nyquist frequency[15] at resolution $\ll 1$ ADU is unexpected for many students.

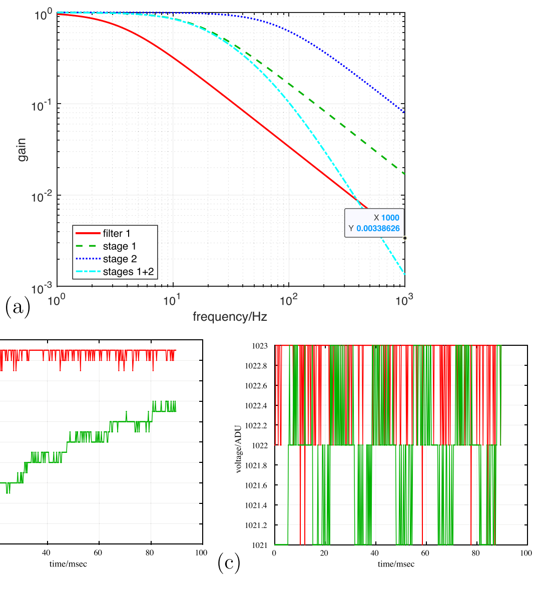


Fig. 5. (Color online) Simulations of lowpass filter gains and transient responses ~ 200 ms after a voltage step. (a) Filter 1 has adequate attenuation of the PWM modulation, but its transient response (b) is slow. (c) The two-stage filter has better convergence and PWM attenuation.

The following code simulates this filter in Fig. 4:

```
R1 = 10e3; % ohms
C1 = 4.7e-6; % farads
freqs = logspace(0,3)'; % Hz
ZC1 = zc(C1, freqs); % capacitor's impedance
gain = ZC1./(R1 + ZC1); % voltage divider equation
loglog(freqs, abs(gain));
```

This code produces the `filter 1` curve in Fig. 5, with $\text{gain} = 3.4 \times 10^{-3}$ at the PWM frequency, within 5% of measurements with a precision mylar capacitor. Inexpensive electrolytic capacitors in the kits deviate from theory by up to a factor of 2 due to their equivalent series resistance[16] at 1 kHz. That such small effects are detected is a testament to the sensitivity of our apparatus and its quantitative environment.

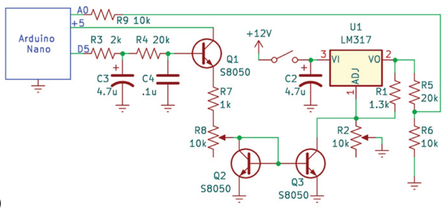
We measured filter settling times [Fig. 5(b)] by programming a voltage step followed by data acquisition 200 ms later (the delay was previously programmed in the Arduino):

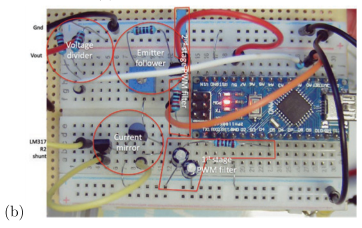
```
writeline(ard,'p 0'); pause(1); % let the value of 0
settle
runOnce = true;
writeline(ard,'p 255');
oscope2;
```

Guided by simulations, we prototyped a two-stage lowpass filter and verified it produced a clean DC voltage and fast convergence [Fig. 5(c)]. This result was a key step in a "bottom up"[17] design of an automated DC power supply.

VI. AUTOMATING AN LM317 VOLTAGE REGULATOR

With more experience analyzing circuits, we revisited the LM317 datasheet to understand *how* the circuit works (Sec. VIII A in the datasheet).[18] An internal 1.25 V voltage reference produces a current $i_1 \sim 1$ mA through resistor R1 in Fig. 6(a), which continues through the potentiometer R2 to ground. This generates a voltage $i_1 *$ R2 that adds to the 1.25 V reference. Algebra yields the characteristic equation, $V_{out} = V_{Ref}(1 + \text{R1/R2})$. This relation suggests that adding a





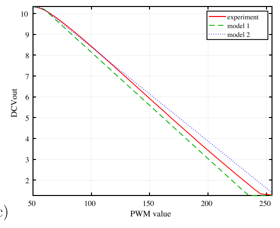


Fig. 6. (Color online) Circuit controls LM317 voltage via Arduino PWM output. (a) Schematic, (b) prototype, and (c) measured performance vs two numerical simulations of the circuit.

parallel current path to R2 to shunt some of i_1 would control (lower) the output voltage.

A current mirror[19] is a previously introduced electronics "building block" that seemed ideal to connect between the LM317 and the PWM filter. However, our two-stage PWM filter has an output impedance (22 kΩ) that is too high to drive the ~ 1 mA mirror current. An emitter follower[20] can solve this problem by reducing the PWM filter's impedance by $\beta \sim 200$. A breadboard prototype [Fig. 6(b)] reveals the circuit works, and this automated script produced the voltage sweep in Fig. 6(c) in <20 s:

```
pwm = (50:5:255)'; % expect Q1 and Q2 are off below 65
for i=1:length(pwm)
        writeline(ard, ['p' num2str(pwm(i))]);
        runOnce = true;
        oscope2; % → new data
        v317(i) = mean(data(:,1))/1023 * vref * (R5+R6)/R6;
end
plot(pwm, v317);
```

Figure 6(a) represents an enormous increase in complexity compared to an LED nightlight that challenged students just 10 weeks earlier. Students come to understand the importance of recognizing modules instead of just components, analogous to recognizing that words, sentences, and paragraphs, as opposed to just letters, are key to reading comprehension. Parsing Fig. 6(a) in this manner, we see a PWM filter next to the Arduino. Q1 is the emitter follower, lowering the filter's impedance to drive current through R7 + R8 that programs the current mirror, Q2 and Q3. To the right is the now familiar LM317 circuit and voltage divider that we've been using for weeks. Modules are also emphasized during troubleshooting, e.g., in asking questions such as "Is the filter working?", "Is the LM317 output >10 V with the current mirror disconnected?."

Automation makes it fun to adjust R8 to change the slope of Fig. 6(c). Maximizing the useful PWM range permits approximately 190 distinct programmable voltages. Soldering this circuit to the perforated breadboard is another opportunity to practice construction and troubleshooting. Most students find their bugs after rewatching class videos, meeting with a TA, or emailing photos and virtual meetings, and they learn many practical lessons. After repeating the calibration sweep, `polyfit` computes cubic polynomial coefficients from a least-squares fit of the data. These coefficients are pasted into a function that sets the output voltage:

```
function pwmVal = volts2pwm(volts, ard)
pv = [-0.0452 0.8474 -24.5377 284.502]; % coefficients
from polyfit
pwmVal = constrain(round(polyval(pv, volts)), 0, 255);
if nargin == 2, writeline(ard, ['p' num2str(pwmVal)]);
end
```

Each student should have unique coefficients that reflect their transistor's β's, resistor values, and other parameters. Curve fitting minimizes all the stationary systematic errors in a least squares sense.

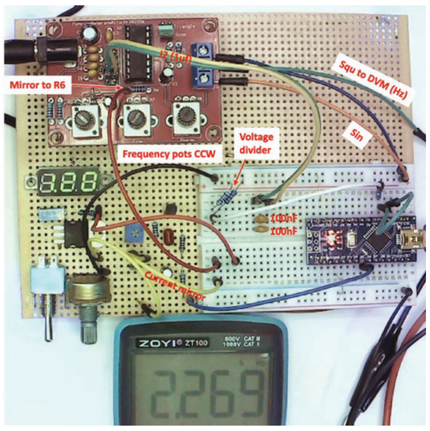
VII. FUNCTION GENERATOR

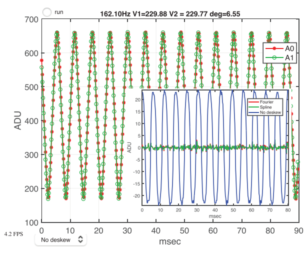
To measure voltages at other frequencies, we assembled a ~\$10 function generator kit based on an XR2206 IC[21] that includes a printed circuit board (PCB) and all components. After soldering the three-transistor control circuit on the perforated board without any traces, students appreciated the ease of a PCB. After verifying it worked with `oscope`, it is affixed to the perforated breadboard [Fig. 7(a)].

The need for stable triggering of the oscilloscope display became immediately apparent since Arduino's data acquisition is asynchronous with the function generator. We implemented a software solution with a 20-line function called `trim` that locates threshold crossings and trims the data to an integral number of periods at stable phase.

Measuring the same sine wave on inputs A0 and A1 revealed the two are not synchronous but interleaved [Fig. 7(b)]. The constant temporal shift between them creates a phase shift that increases with frequency. Because this is a systematic error, we make a new version, `oscope3`, that eliminates it, first using a `spline` to numerically simulate resampling channel A1 at the same times as A0. This time-domain algorithm begins to fail at high frequencies, so a frequency domain algorithm that shifts each frequency component by phasor multiplication is used. Both algorithms (along with "do nothing") are combined in an 18-line function called `arddeskew`, and two more code lines make the applied algorithm selectable and visible in the GUI [Fig. 7(b), bottom left].

To measure Bode plots[22] with automation, a search of the XR2206 datasheet (p. 11)[21] revealed the function generator's frequency is controlled by a DC current of approximately 1 mA, which can also be controlled with our current mirror circuit. Soldering a shunt wire to the function generator board, we repeated the calibration procedure used for the LM317. Achieving the widest usable frequency range, limited at the low

(a)


(b)


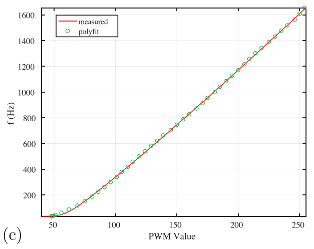
(c)


Fig. 7. (a) Function generator kit mounted above power supply and control circuit. (b) Interleaved sampling results in systematic temporal shifts that were eliminated using time domain and frequency domain algorithms. Inset shows both algorithms reduce the difference between A0 and corrected A1 at low frequencies to <1 ADU. (c) Automated frequency sweep with the current mirror circuit.

end by sampling at least a full period and at the high end by the Arduino's Nyquist frequency, is accomplished with $2 \times 100\,\mathrm{nF}$ capacitors in parallel on the solderless breadboard, jumpered to the function generator [Fig. 7(a)]. Another automated calibration run, along with `polyfit`, created the function `freq2pwm(Hz, ard)` and Fig. 7(c):

```
function pwmVal = freq2pwm(freq, ard)
% set the function generator frequency
% polyfit renormalized with f(kHz) and (PWM-150)
% for better conditioning
pf=[-26.942 145.068 -308.532 331.482 -196.661 193.522 -99.464];
pwmVal = constrain(round(polyval(pf, freq/1e3)) + 150, 0, 255);
if nargin == 2, writeline(ard, ['p' num2str(pwmVal)]); end
```

Our Arduino-controlled current mirror circuit demonstrates a versatility unexpected for hardware. Ironically, Hayes and Horowitz (p. 222)[23] introduce current mirrors with the remark, "You can skip this ... [because] you are unlikely to adopt a mirror as an element in your own designs." With `polyfit` compensating for stationary errors such as transistor differences, the mirror+software is a robust and versatile object. Furthermore, this suggests a hierarchical view of electronics akin to object hierarchies in programming languages. We're constructing an organizing framework, with discrete components at the base, IC's, "building block" circuits, and so on. The synergy of interconnected hardware and software that epitomizes lab automation culminates in the final lab.

VIII. BODE PLOT

The last lab activity measures and models the frequency response of an *RC* series circuit. We first refactor[24] `oscope` by adding a data structure called `dat` to clean up MATLAB's now cluttered workspace. The following code executes a frequency sweep while measuring 40 000 voltages in $<45\,\mathrm{s}$ and saves 50 frequencies and 100 Fourier amplitudes for subsequent analysis:

```
targetFreqs = logspace(log10(31),log10(1910),50)';
amplitudes = zeros(length(targetFreqs), 2);
freqs = zeros(size(targetFreqs));
for i = 1:length(targetFreqs)
   freq2pwm(targetFreqs(i),ard); pause(.5);
   runBtn.UserData = true; % runOnce
   oscope4; % measure new data in dat
   freqs(i) = dat.freq; amplitudes(i,:) = dat.ampl;
End
```

The data are renamed and normalized and passed with simulated results to a `bodePlot` function to format the appropriate graphics [Fig. 8(a)].

```
vr = amplitudes(:,1) - amplitudes(:,2); % voltage
across R
vc = amplitudes(:,2); % voltage across C
measuredGain = [vr vc]./amplitudes(:,1); % normalize to
gain
zc1 = zc(C1, freqs); % impedance at the measured freqs
theoryGain = [R1./(R1+zc1), zc1./(R1+zc1)]; % voltage
divider equation
bodePlot(freqs, [measuredGain theoryGain])
```

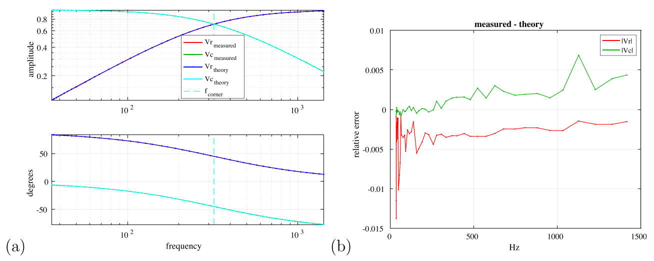


Fig. 8. (Color online) (a) Bode plot of an *RC* series circuit measured and simulated at 50 frequencies in $<45\,\mathrm{s}$. (b) Difference between amplitude measurements and theory values are $<1.5\%$.

A difference plot is needed to resolve the $<1.5\%$ differences between model and measurements [Fig. 8(b)].

IX. STUDENT EXPERIENCES

These lab units exposed students to diverse circuits with active and passive components, in DC and AC domains, explored via both simulations and measurements. It did so without needing conventional test equipment. Roughly half of the activities are related to designing, constructing, characterizing, and debugging the student's own test equipment, component by component and code line by line, and iteratively improving its functionality (and complexity). These activities also have a permanence atypical for student lab activities, which may make them more memorable and potentially more useful, as well as enabling in later classes and extracurricular projects.

Figure 9 enumerates key concepts covered in each lab activity. Repetitive exposure to concepts, especially in multiple contexts, assists integrative learning[25] and retention. In particular, students commented positively about using MATLAB and Live Scripts every day. What started as a foreign environment became more familiar. The curriculum overlaps with all of the learning outcomes recommended by the American Association of Physics Teachers (AAPT).[26] Curriculum is always a work in progress; feedback and collaboration to improve this one and expand its impact is most welcome.

Our electronics classes typically consist of about ten students split between physics and mechatronics majors, numbers that are less than optimal and insufficient for quantitative education research results. That said, here's a report on our experience: Average scores on a 30-question electronics knowledge inventory final were 80%; however, the same assessment was not used on the previous curriculum, so no comparison is possible. With this active curriculum, students appeared to better appreciate the versatile functions of electronic components, and especially understood transistors much better. Their acquired skills soldering, troubleshooting, debugging, and programming are difficult to quantify with standard assessments.

Student feedback was consistently highly favorable about soldering and the fact that they were able to keep the apparatus. "It was really cool to build a DC power supply and function generator, and then modify them to be controlled by the Arduino and MATLAB." "It was eye opening how easy it was to modify them once you understand a little about how they work." The mechatronics students a semester later in a subsequent course shared these lessons. With hindsight, they

	Concept Lab Module	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	Voltage Divider	x	x		x	x	x		x	x	x	x	x			x
2	Series/parallel	x		x			x		x	x	x	x			x	x
3	Hydraulic analogies to electricity	x				x										
4	Ohm's Law	x	x	x	x	x	x				x	x				
5	Schematics	x	x	x	x	x	x		x		x	x	x		x	x
6	Design		x	x					x	x	x				x	
7	Simulation		x	x		x			x	x	x	x		x	x	x
8	DMM		x	x							x		x		x	
9	Diodes		x		x		x									
10	Soldering		x	x							x		x		x	
11	Solderless Breadboard		x	x	x	x	x		x	x	x		x		x	x
12	Troubleshooting (HW, and SW)		x	x	x	x	x		x	x	x	x	x		x	x
13	Thevenin's Theorem, impedances		x				x			x	x	x				x
14	Characteristic curves		x	x	x	x	x				x				x	
15	Device Data Sheets			x		x	x		x		x		x		x	
16	Switches			x	x		x									
17	LM317 voltage regulator			x							x	x				
18	Arduino programming in C				x	x		x								
19	Matlab programming/Scripts				x	x		x	x	x	x	x	x	x	x	x
20	ADU, conversion factors, rescaling, normalization				x			x	x		x			x	x	x
21	Laboratory Automation				x	x	x	x			x		x		x	x
22	Transistor states					x	x				x	x				
23	Transistor applications, e.g., emitter follower, current mirror, ...						x				x	x			x	
24	Curve fitting (polyfit, spline)					x					x		x		x	
25	Refactoring code							x	x							x
26	Serial communication, binary data							x		x	x					
27	Oscilloscope, Error handling (try/catch), User interfaces							x	x	x	x		x		x	
28	Electret microphone, signal transduction, audio								x							
29	Spectrum analysis, frequency domain, FFT								x	x	x		x		x	x
30	AC -- time varying voltages, phase								x	x	x					x
31	Op amps, bias, gain, noise, SNR								x			x				
32	RC filter, PWM, settling time, RMS									x	x					x
33	Time difference in measurement (Multiplexing, demux)									x			x	x		
34	Aliasing									x	x				x	
35	Threshold and triggering												x	x		

Fig. 9. Concept matrix shows which lab modules discuss 35 topics.

- Learned to approach troubleshooting with trial-and-error and iterative strategies.
- Developed resilience and adaptability when faced with unexpected challenges.
- Learned to seek information online, in datasheets, code repositories, and chatbots.
- Learned about project and time management, and the importance of iterative design, testing, and documentation.
- Used discrete electronics to interconnect and modify other devices and systems.

The largest complaints were about the amount of time spent debugging (although they expressed satisfaction with their hard-won skills) and general difficulty in writing formal lab reports. For most students, writing lab reports was their least favorite activity (and grading them was the professor's least favorite, as well). In response, more scaffolding in the Live Script files was provided, and students were asked to do more reading and deleting (e.g., of theory and examples) and less structural writing. TA's or peer mentors were also recruited from the Campus Writing Center, and when possible, former students to assist with troubleshooting and encouragement.

Each week involved constructing complex apparatus (hardware and software working synergistically) to measure and enlighten understanding of electronic devices and circuits. For most students, these activities provided their first experience in soldering and working this way with their hands, their first time programming, and their first time using a computer as a lab partner. Interacting with a command line to control an automated apparatus, acquire data, and perform exploratory data analyses supercharges our intellectual capabilities and our productivity. That's why these skills are highly valued by scientists and employers. However, it was also too much for some students to digest, especially those unable to commit adequate time to the class. About one-fifth of students repeat the class, generally taking fewer and/or easier concurrent classes.

Any curriculum invariably favors certain types of learners and discriminates against others. Flipping the electronics lab is no different. Some students struggle to focus and self-motivate outside class, whereas others, e.g., who have difficulty regularly attending class, benefit from the opportunity to work on their labs when and where their schedules allow. Therefore, this approach is not best for all students, but it especially helps many students who do poorly in conventional structured lab settings that end with the class bell. The course shares elements of an Authentic Learning Experience (ALE).[27]

X. CONCLUSIONS

Electronics is hard to learn in part because we have no innate electric sense.[28] We have to rely on complex tools, such as oscilloscopes and spectrum analyzers, just to perceive the signals and construct intuition and heuristics that model the underlying and unfamiliar solid-state physics. Many students are intimidated at their first encounters with the myriad controls on an oscilloscope. However, integrating programming, digital signal processing, and, most importantly, automation by introducing a microcontroller at the beginning of the course eliminates huge amounts of tedious manual data collection and data entry. Students measure far more data more often and can devote more time to understanding it. The Arduino provides practical applications of voltage dividers, unit conversions, impedance, aliasing, and systematic vs random errors, as topics that constantly arise in natural contexts. The flood of data demands automated data reduction and visualization, which provides students with 16 weeks of practice using MATLAB.

Many of our lab activities focus on constructing the electronics workbench. This work gives students diverse experiences looking up datasheets, designing, simulating, prototyping, soldering, debugging, and so on. Building these systems demystifies a lot about these devices, and as we iteratively add functionality and features (e.g., in the evolution of the `oscope` program), students participate in processes of adding and managing complexity. These activities also have permanence. The goal isn't to do a one-off measurement, but for students to develop skills that are used throughout the course.

Live Scripts plays a key role in helping students combine data with metadata, algorithms with their results, and precise sequences of measurement and analysis expressed unambiguously as executable code. Conventionally, students organize the diverse pertinent information and context about their labs more haphazardly, in their heads, notes, spreadsheets, lab guides, textbooks, etc. In our increasing technological world, we need to use computers more as active lab assistants—and certainly for offloading mundane tasks like data logging and sweeping voltages and frequencies.

Learning the skills to read and tweak source code, as opposed to writing code *tabula rasa*, is likely a better introduction to programming. It is our hope that the examples of code included in this paper are short and simple enough to engage readers (and students) to read and appreciate their concise meaning. The more verbose and complex a language is, the harder it is for students to engage with it. By this metric, I believe MATLAB is superior to Python, Julia,[29] and even Octave, and the challenges of using it in this class are more than offset by its benefits.

Introducing multiple concepts concurrently rather than sequentially has demonstrable advantages, as shown in the study by Schwartz.[30] As courses become more advanced, instructors tend to introduce a concept and move on to the next one to "cover" ever more material. In our curriculum, most key concepts explicitly repeat in many units [Fig. 9]. Voltage dividers, transistor circuits, and filters are used every week since their introduction, not to mention the constant reuse of the MATLAB software. The effectiveness of repetition in learning doesn't cease in college.

We believe our course can be more effective and successful if it is more self-paced. Many students simply fell behind, due to other classes and obligations, and couldn't catch up. We envision a completely self-paced version of the course, where completing assessments (including working hardware) is required to enable or activate the next unit. Frequent automated assessments could identify misconceptions and create individualized lesson sequences that help each student progress. This approach could also reduce the drudgery of grading.

Giving every student a $120 kit (and requiring a computer) isn't universally feasible. Although more affordable than many textbooks, the curriculum might be adapted to have students share tools and equipment in a common space. For our curriculum, professional-level power supplies, function generators, and oscilloscopes are no longer required, even though experience with commercial electronic equipment is a key educational experience the students must now seek elsewhere. Understanding the challenges, pitfalls, and value of time and frequency domain measurements, students should be able to pick up the mechanics of using commercial instruments much quicker. However, it's also intriguing to think how subsequent courses could benefit from every student having their own electronics workbench. These students will graduate owning physical toolkits that augment their new intellectual abilities and further empower them to turn their ideas into reality.

SUPPLEMENTARY MATERIAL

Please click on this link to access the supplementary material, which includes the following:

(1) `Supplementary.pdf` describes the contents of the kit, explains key Arduino and MATLAB source codes, briefly discusses software ports to Octave and Python, and lists the lab modules.
(2) `ElectronicsToolbox.zip` contains all the MATLAB and Arduino source code files and a sample Live Script file, `m02_voltageAndCurrent.mlx` that can be executed in MATLAB.
(3) `m02_voltageAndCurrent.pdf` is a pdf of the corresponding Live Script file.
(4) `m06_TransistorGain.pdf` is a pdf of the sixth Live Script module.

For more information, contact the author and describe your interest. Print readers can see the supplementary material at https://10.60893/figshare.ajp.c.7355569. Printed color figures were funded by NSF EES #1928693.

AUTHOR DECLARATIONS

Conflict of Interest

The author has no conflicts to disclose.

[a]Electronic mail: brian.rasnow@csuci.edu, ORICID: 0009-0007-9829-6290.

[1]J. Micheal, "Where's the evidence that active learning works?," Adv. Physiol. Educ. **30**, 159–167 (2006).

[2]See https://www.arduino.cc/en/software to download the IDE. A serial driver must also be installed for our Arduinos, see https://learn.sparkfun.com/tutorials/how-to-install-ch340-drivers.

[3]Only basic MATLAB is required from https://www.mathworks.com/academia.html. MATLAB's Web version currently doesn't support serial port devices; hence students must run an executable application on their computer.

[4]Octave (https://octave.org) is an open-source language designed for compatibility with MATLAB but is more difficult for students to configure and learn. The instrument-control package is required <https://gnu-octave.github.io/packages/instrument-control/>.

[5]<https://www.amazon.com/dp/B08M9F3LK5> has 10× manual zoom lens that produces excellent results at low cost. It is mounted on a ball head 46 cm (18″) above my desk sideways (software rotate the image to portrait orientation). Field of view is 3.6–22.2 cm wide (1.25–8.8″ W × 1.68–11.65″ H).

[6]See <https://www.mathworks.com/help/matlab/live-scripts-and-functions.html> for "Live Scripts and Functions."

[7]See <https://jupyter.org> for "Jupyter."

[8]See <https://en.wikipedia.org/wiki/Thevenin's_theorem> for "Thevenin's Theorem."

[9]A. Bindra, "Three-terminal linear regulator evolution continues unabated," IEEE Power Electron. Mag. **1**, 12–15 (2014).

[10]<https://www.mathworks.com/discovery/arduino-programming-matlab-simulink.html>is not used here.

[11]See <https://docs.arduino.cc/learn/microcontrollers/analog-output/> for "Basics of PWM (Pulse Width Modulation)."

[12]See <https://en.wikipedia.org/wiki/Phasor> for "Phasor."

[13]See <https://en.wikipedia.org/wiki/Aliasing> for "Aliasing."

[14]See <https://mathworld.wolfram.com/FourierSeriesSquareWave.html> for "Wolfram Mathworld."

[15]See <https://en.wikipedia.org/wiki/Nyquist_frequency> for "Nyquist Frequency."

[16]https://en.wikipedia.org/wiki/Equivalent_series_resistance for "Equivalent Series Resistance."

[17]See <https://en.wikipedia.org/wiki/Bottom-up_and_top-down_design#Computer_science> for "Bottom-Up and Top-Down Design"

[18]See <https://www.ti.com/lit/ds/symlink/lm117.pdf> for "LM317 datasheet."

[19]See <https://en.wikipedia.org/wiki/Current_mirror> for "Current Mirror."

[20]See <https://en.wikipedia.org/wiki/Common_collector> for "Common Collector."

[21]See <https://www.sparkfun.com/datasheets/Kits/XR2206_104_020808.pdf> for "XR-2206 Monolithic Function Generator."

[22]See https://en.wikipedia.org/wiki/Bode_plot for "Bode Plot."

[23]T. Hayes and P. Horowitz, *Learning the Art of Electronics* (Cambridge University Press, Cambridge, UK 2016).

[24]See https://en.wikipedia.org/wiki/Code_refactoring for "Code Refactoring."

[25]See https://en.wikipedia.org/wiki/Integrative_learning for "Integrative Learning."

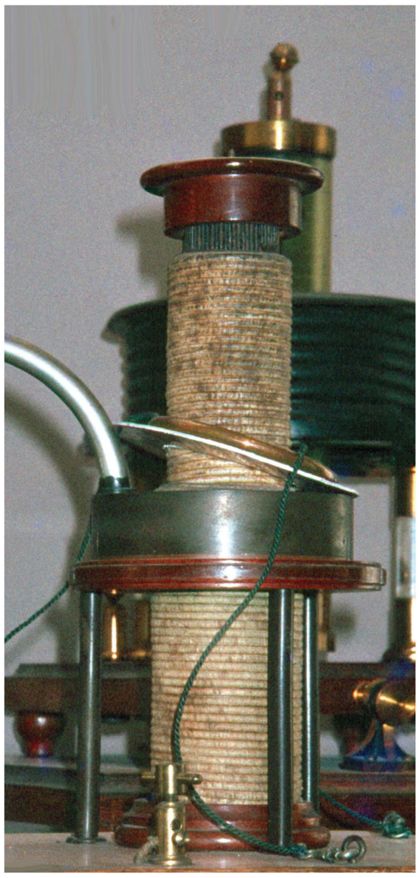
[26]J. Kozminski *et al.*, see https://www.aapt.org/Resources/upload/LabGuidlinesDocument_EBendorsed_nov10.pdf for "AAPT Recommendations for The Undergraduate Physics Laboratory Curriculum" (2014).

[27]J. A. Mraz-Craig, K. L. Daniel, C. J. Bucklin, C. Mishra, L. Ali, and K. L. Clase, "Student identities in authentic course-based undergraduate research experience," J. College Sci. Teach. **48**(1), 68–75 (2018).

[28]C. Assad, B. Rasnow, and P. K. Stoddard, "Electric organ discharges and electric images during electrolocation," J. Exp. Biol. **202**(10), 1185–1193 (1999).

[29]Julia <https://julialang.org> is designed for high performance across platforms.

[30]P. Schwartz, "Focusing on concepts by covering them simultaneously," Phys. Teach. **55**(5), 280–284 (2017).



Elihu Thomson Apparatus

This was designed by Elihu Thomson (1853-1937), the prolific electrical inventor and entrepreneur who was one of the founders of the General Electric Company. The basis of the apparatus is a heavy-duty electromagnet with a core made of parallel iron wires to reduce eddy current heating. The aluminum ring is placed around the coil and flies off when a line voltage A.C. signal is applied to the coil; this is due to the repulsion of the magnetic field induced in the ring. This apparatus is in current catalogues, and it can also be found in the Max Kohl catalogue of 1900. This example is at the Callan Museum at St. Patrick's College in Maynooth, Ireland, and was photographed during a visit in1999. (Picture and text by Thomas B. Greenslade, Jr., Kenyon College)