Introducing Programming and Problem Solving with Arduino-based Laboratories

Dr. Michael Daugherity, Abilene Christian University

Dr. Michael Daugherity is an Associate Professor of Engineering and Physics at Abilene Christian University in Abilene, Texas. He received his PhD in Nuclear Physics from the University of Texas at Austin. His primary research focuses on nuclear physics experiments at the Brookhaven National Lab and Fermi National Accelerator Lab atom smashers performing data analysis and building radiation detectors. Including undergraduate students in research is a major emphasis at ACU's Engineering and Physics department. Dr. Daugherity's other research interests include data science and machine learning as well as education and science outreach.

Introducing Programming and Problem Solving with Arduino-based Laboratories

Abstract

First year engineering and physics undergraduate students at Abilene Christian University begin their studies with an intro course designed to teach fundamental skills, explore career options in engineering and physics, and build community. We have developed a series of labs and activities based on Arduino microcontrollers that helps us accomplish all three of these goals.

Through departmental self-studies, Industrial Advisory Board recommendations and internship programs we identified programming skills as an area to strengthen in the curriculum, particularly for physics students. We now devote roughly 1/3 of class and lab time in our intro course to Excel and programming. Encouraging algorithmic thinking and working within the constrained environment of a programming language helps reinforce the structured approach to design and problem solving introduced earlier in the course. Experience has also shown that many students have an easier time learning programming when coupled with hardware since they can see the effect of code running in the real world. Building simple circuits with Arduino microcontrollers also accomplishes the additional goal of exposing students to different fields of engineering and physics. Many of our students have no prior experience in programming or circuits, so these activities provide a fun first exposure to these fields.

Students purchase a very inexpensive kit with an Arduino Uno and all of necessary components to build many projects. We start with a basic LED circuit and then carefully sequence class and lab activities that gradually introduce more complex components and programming skills. The first major project is a light meter using a photoresistor and servo to display ambient light levels on a calibrated scale. Other projects include displaying Morse code with LEDs, a temperature monitor, a proximity alarm, and a robotic arm. Finally, the students incorporate Arduinos into their major design project for the semester. They are encouraged not only to apply what they have learned but also to explore new capabilities beyond what has been required in class. In conclusion, we have found Arduino-based labs to be a low-cost, high-impact way of introducing programming and problem solving into the engineering physics curriculum.

Introduction

Programming skills are a crucial part of the engineering and physics curriculum. Survey data from the American Institute shows that students who received a physics bachelor's degree in 2013 and 2014 are extremely likely to regularly use programming skills [1]. Of these students working in the private sector, 75% regularly used programming in engineering jobs, while 95% used programming in CS or IT jobs. A similar survey found the vast majority of physics PhDs in 2013 and 2014 use programming regularly [2]. Aside from learning marketable job skills, programming is useful for developing structured problem solving techniques. Programming teaches the design cycle of breaking a problem into steps, developing a flowchart, implementing and testing each step, documenting work, and iteratively testing the complete solution. These skills clearly transfer to non-programming problems in engineering and physics.

However, despite the importance of programming, it is often difficult to find enough room in the crowded undergraduate curriculum for students to adequately master these skills. Engineering and physics programs must be deliberate and creative in finding ways to incorporate

programming throughout the curriculum. This paper describes one way of introducing programming to first-semester freshmen in engineering and physics.

Setting

At Abilene Christian University, the engineering and physics programs both operate within the same department. All entering engineering and physics majors are required to take an *Introduction to Engineering and Physics* class with accompanying lab. The course has undergone significant revision since its inception in 2012, and now includes a variety of topics. The lecture portion emphasizes fundamentals such as vectors, units, estimation, design, problem solving, Excel, programming, and careers. The lab is largely based around a semester-long project to illustrate the entire engineering design process. Past projects include "Angry Bird Launcher" catapults that shoot dodge balls at targets up to hundred feet away, and building Miniature Golf holes incorporating multiple design elements and constraints. As the students develop programming skills, they are able to include microcontrollers into their projects. This year's mini golf holes were full of LEDs, servos, and sensors incorporated in many creative ways.

In addition to technical skills, another objective of the intro course is to improve retention by developing a sense of community and working on fun projects. Daily group work and project teams encourage students to connect with each other. Finding fun projects for them to work on can be challenging. Previous iterations of the intro course did "pure" programming in MATLAB with no additional hardware beyond a standard computer. Many students struggled to engage with this component of the course. Over time we introduced some robotics components that were very successful in sparking interest. As we added more hardware elements, we noticed an additional benefit of students having an easier time learning programming by seeing the effect of code running in the real world. The hardware elements also served as a first introduction to electrical engineering for many students, which helps satisfy the careers objective of the course.

Another objective of the intro course is to develop problem solving skills. A natural approach to structured problem solving is to have students develop an algorithm, or flowchart, to carefully break a problem into steps. Programming provides excellent ways to practice these skills because the algorithm then becomes a blueprint for the program. Requiring the students to produce documentation helps them think through their solutions while practicing technical communication.

Therefore, we have included a significant unit on programming with microcontrollers to the first-year *Introduction to Engineering and Physics* course to teach a crucial job skill, improve retention, and develop problem solving skills.

Software

The IEEE Spectrum releases annual reports on the top programming languages [3]. While the current top language is Python, C++ maintains a very close second. In fact, 4 of the top 5 languages in 2018 are in the "C++ family" sharing similar syntax. We also considered input from our department's Industrial Advisory Board, recent graduates, and current faculty and students in selecting a programming language. Two large factors were the prevalence of using C/C++ in our senior capstone projects, and that our computer science department teaches

introductory programming in C++. The Arduino IDE is freely available on Windows, Mac OS X, and Linux (https://www.arduino.cc/). Even though we have chosen to use C++ for these programming labs, they could easily be adapted to other languages that support the Arduino platform such as Python or MATLAB.

Hardware

As discussed above, we find that students are more interested and have an easier time learning programming when we include electronic components. Instead of changing a variable in memory, students get to see LEDs light up and servos move. An additional objective of the freshman intro course is to give students some exposure to different disciplines in engineering and physics, so by using microcontrollers we introduce students to both programming and basic electronics.

We use an Arduino Uno (or compatible) microcontroller for our hardware platform. Arduinos are ideal solutions for being inexpensive, compatible with a huge range of sensors and other components, and has a large community providing support.

In earlier iterations of the course the department provided the Arduinos and other components. However it was difficult to make these available outside of class or lab hours for students, so assigning homework was challenging. We solved the problem this year by having students purchase their own Arduino kit. Not only did this simplify the logistics of the department keeping up with supplies and giving students hardware to use outside of class, the students had a new sense of ownership and investment in the projects. This year they got to keep what they made! There are many manufacturers that supply Arduino kits. We contacted several to find one that would give a bulk educational discount on kits that had enough, but not too many, components.

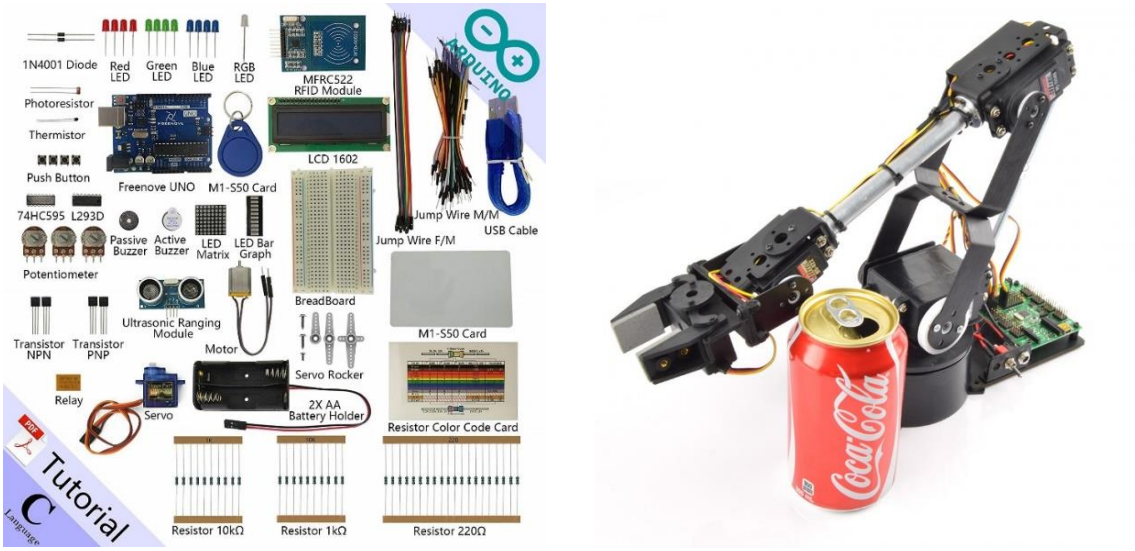


Figure 1: The hardware including the kit components (left) and AL5D robot arm (right)

We did a final "capstone" lab using Lynxmotion AL5D robot arms provided by the department using custom Arduino software developed for the lab. The goal of this lab is to end this unit with an exciting project that showcases the Arduino in more advanced applications.

Arduino Labs

Students are divided into groups of 2 or 3 people. Each group needs access to one Arduino kit and one laptop computer for programming. While students are highly encouraged to use their own laptops, we made departmental laptops available. We use the 50 minute class times to introduce new programming topics, new electronic components, and practice skills. Class meets only twice per week. Students have a 3 hour lab each week to do more complex projects. The Arduino activities are carefully staged to slowly introduce new programming and electronics concepts one at a time. We invested a total of 3 weeks of the 15-week semester to these labs. We staged them early in the semester so students would be able to incorporate Arduinos into the designs of their major lab projects, as well as to help generate excitement for the new class. The first week of class begins with introducing the syllabus and welcoming new students to the department. Then we discuss *professional* problem solving skills emphasizing the necessity of breaking a problem down into steps, developing an algorithm, and practicing good communication. The Arduino labs begin the second week of the semester.

Table 1 below gives a short description of each class and lab period with new programming and electronics concepts introduced and used in projects.

Class/Lab	**Programming**	**Electronics**	**Project**
Class 1-1	introduction, Arduino IDE, C++ basics	LED, digital output	SOS Morse Code
Class 1-2	expressions, variables, serial output with *print* and *println*	serial communications	Number classifier
Lab 1	simple loop, serial output	voltage, resistors, simple LED circuits	3 Blinking LEDs
Class 2-1	logic, if statements	serial input	Number Constrainer
Class 2-2	if statements	RBG LEDs	Keyboard Controlled RGB LED
Lab 2	reading inputs	servos, photocells	Light Meter
Class 3-1	review	analog input, thermistor	Battery Tester, Temperature Monitor
Class 3-2	complex conditional statements, logical operators	ultrasonic rangefinder	Proximity Alarm
Lab 3	algorithms, loops	robot arms, servos	Robot Arm

Discussion

As an example of how programming and electronics concepts are gradually staged, consider the first week of Arduino activities. In Class 1-1, after a brief introduction to Arduinos, we use the *Blink* example that comes with the Arduino IDE to demonstrate turning the built-in LED on and off. The students then modify that example to make the LED blink SOS in Morse code. In Class 1-2 we focus on programming basics to calculate simple expressions and display output in the serial communication monitor. Then in Lab 1 students get to build their first circuit using LEDs

and resistors from their kits and modifying the *Blink* example to control it. The lab carefully guides students with step-by-step instructions for making a one LED circuit. Then they are told to construct a 3 LED circuit on their own and make the lights blink in a specified pattern. By the end of the first week students have achieved a basic working knowledge of how to use an Arduino.

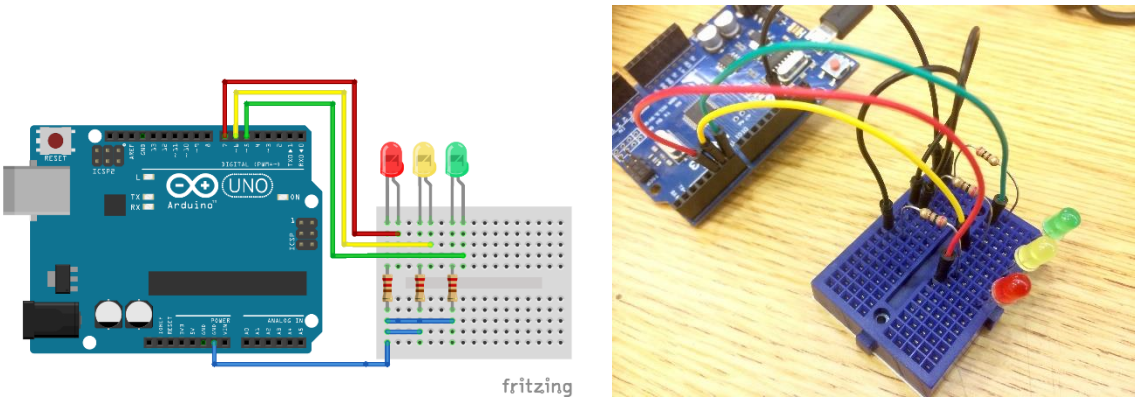


Figure 2: The 3 LED circuit diagram (left) and implementation (right)

The second week's classes introduce logic with the *if* statement. Then they learn how to read numbers from serial input (typed with the computer keyboard into the serial monitor) and do a simple Number Constrainer project where they read a number and ensure it is within a specified range. Class 2-2 continues with RBG LEDs and a project to control the controls individually with keyboard input. Some students require a significant amount of time and practice before they feel comfortable with programming and electronics, but we find that by this project students have pushed through their initial barrier and have begun to enjoy programming.

Lab 2 introduces the concept of reading input voltages as demonstrated through experiments with a photocell. Then we introduce a simple servo that is easily controlled with an included library. Finally, the students must work on their own to combine these two elements into a prototype light meter (see Appendix).

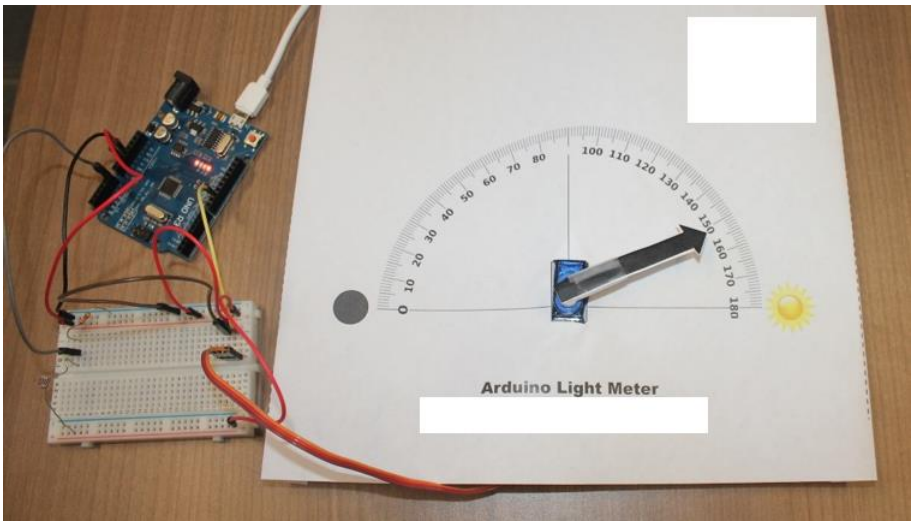


Figure 3: The light meter project

The final week of class is devoted to two short projects. Class 3-1 practices analog input. First we build a battery tester by reading the voltage of a AA battery, then we introduce a thermistor and build a temperature monitor where a red LED lights up if the temperature exceeds a certain value. Class 3-2 introduces the ultrasonic rangefinder and how to use logical operators to make more complex conditional statements. The rest of class is spent working on a proximity alarm project. Finally, Lab 3 concludes the Arduino unit by letting students control 5-servo AL5D robot arms with a custom-made program. This lab returns the emphasis to problem solving skills and algorithmic thinking by making students program the arm to perform a series of actions.

Conclusion

Programming is an important component of the engineering and physics curriculum. By studying programming students are learning important job skills, developing problem solving abilities, and equipping themselves to build interesting projects. We advocate for introducing programming early and using it throughout the entire engineering and physics curriculum. This paper demonstrates a 3 week series of activities using free software and very inexpensive hardware to introduce students to programming and electronics with an emphasis on problem solving skills. Further work will attempt to measure the impact of this programming introduction on student's development within the engineering and physics curriculum, particularly with learning outcomes and student feedback.

Acknowledgments

The author wishes to thank his colleagues Dr. Darby Hewitt, Dr. Lori Houghtalen, and Dr. Tim Kennedy for their work in developing the *Introduction to Engineering and Physics course* and implementing the Arduino activities discussed here.

References

[1] P. Mulvey and J. Pold, "Physics Bachelors: Initial Employment," *American Institute of Physics Statistical Research Center,* April 2017. [Online]. Available: https://www.aip.org/sites/default/files/statistics/employment/bachinitemp-p-14.1.pdf

[2] J. Pold and P. Mulvey, "Physics Doctorates: Skills Used & Satisfaction with Employment," *American Institute of Physics Statistical Research Center,* August 2016. [Online]. Available: https://www.aip.org/sites/default/files/statistics/employment/phds-skillsused-p13v2.pdf

[3] S. Cass, "The 2018 Top Programming Languages", *IEEE Spectrum,* July 31, 2018. [Online]/ Available: https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages

Appendix

To provide a complete example, the Light Meter Lab is included in the following pages

Appendix

Intro Programming Lab #2

The Light Meter

This week you will be building a meter that measures light levels. This is our first lab project that reads input from a sensor, so let's talk about analog inputs.

Part 1) Reading Voltages

We will use the `analogRead` command to measure analog input. It compares the analog pin to the Arduino's reference voltage (presumably 5.0 V from the USB) and returns an integer from 0-1023 where 0 is zero volts and 1023 is the max voltage. The integer comes from an analog-digital convertor (ADC) built into the processor, so ADC value / 1023 is the *fraction* of the max voltage we just read:

$$V_{IN} = V_{MAX} * \frac{\text{ADC}}{1023}$$

One problem is that if we need precise measurements then we have to know V_{MAX}. A USB cable gives us *about* 5.0 volts, but not exactly. We use a calibrated power supply, measure the input voltage directly, or get it with a sneaky trick…

Exercise 1

Every time you write code you need to test it with a case where you know the right answer. **Always start with a reality check.** Start with this simple sketch to read ADC values

```
int readPin = 0;  // Analog input pin (use 0 for A0, 1 for A1, etc)

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly
  int adc = analogRead(readPin);
  Serial.println(adc);
  delay(1000);
}
```

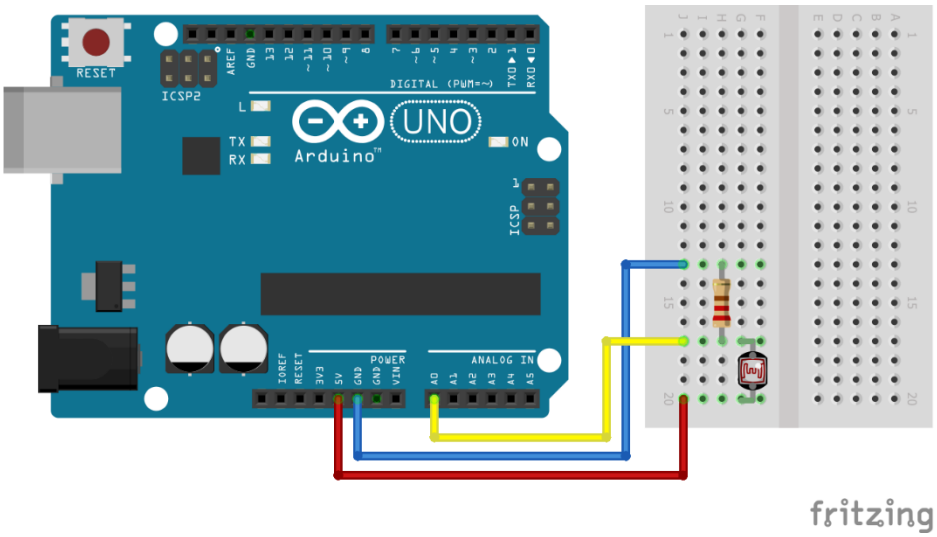
1) Connect a wire from A0 to GND and verify that you read exactly 0.
2) Move the wire so it connects A0 to 5V and verify that you read exactly 1023.
3) Move the wire again so it connects A0 to 3.3V. Record your ADC value.
4) Because of the internal circuitry we know that the 3.3V pin is actually very close to 3.3 volts. So use your ADC value with $V_{IN} = 3.3\ V$ in the voltage equation above to calculate V_{MAX}. Record your V_{MAX} value and use it in all future voltage measurements.

To measure voltages, just modify this sketch to calculate V_{IN} using this new V_{MAX} value.

Now we can work on your light meter circuit:

Setup

- Connect +5V and GND to your breadboard
- Plug in one leg of the photocell in +5V, other leg connects to 10k resistor
- connect second leg of 10k resistor to GND
- Attach wire from ANALOG IN to place where photocell and resistor are connected



Exercise 2

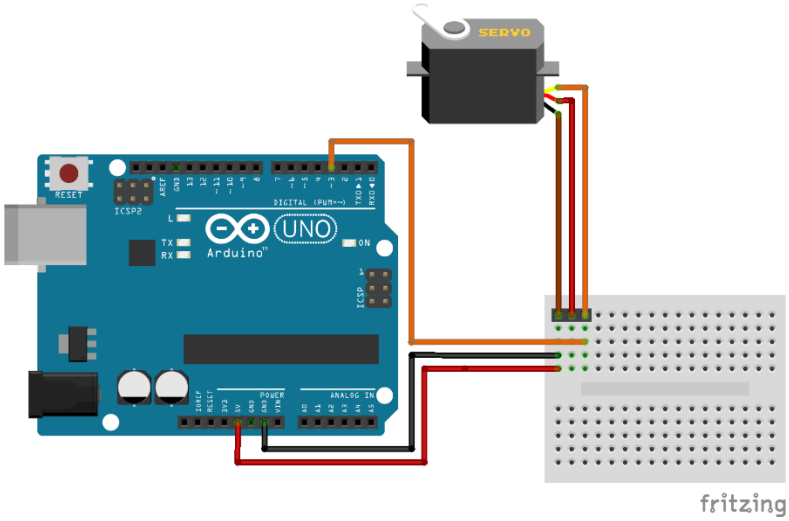
The photocell is light sensitive. Play around covering it with your hand or using your phone's flashlight. Record the resulting voltages in your worksheet.

Part2 – Servos

Servos are like motors that can move to a given angle. Most servos like ours can only rotate 180 degrees. The plastic attachment that turns is called the servo horn.

Setup

- Attach servo to breadboard. This is easiest using a block of 3 header pins. Make sure:
  - orange = signal to PWM pin (marked by ~)
  - red = +5V
  - brown = ground



Code

Servos require a few lines of code to set up, and then we will use the `write` command to set the position from 0 to 180 degrees. Increasing the angle rotates the servo counter-clockwise.

```
// Servo Example
#include <Servo.h>

int servoPin = 3; // PWM pin for servo
Servo s;  // declare servo

void setup() {
  s.attach(servoPin);
}

void loop() {
  s.write(90);  // set servo position
  delay(1000);
}
```

Exercise 3

Write a program that sweeps the servo from 0 to 180 in steps of 15 with a delay in between. Do *not* just copy and paste `s.write` a bunch of times. Instead, declare a variable for your servo position and add 10 to it in the loop. Use an if statement to reset the position to 0 if it gets bigger than 180. Write your loop function on your worksheet.

Exercise 4

Get the light meter paper and set your servo position to 90. Then tape an arrow to the horn and stick it on the servo pointing at 90 degrees on the paper. Now use your program in exercise 3 to take some calibration data. *It will be going backwards* since servos move counter-clockwise and protractors go clockwise, and it may go off scale. Regardless, these numbers will help you understand how to control the servo.

Part 3 – Light Meter

Your job now is to combine these two elements into a working light meter. Remember that once your code is running on the Arduino, we only use the USB computer connection for power. If we used a battery pack you could walk around with your light meter.

Exercise 5

Organize your data to help with the next step. For both extremes of light and dark copy the voltages from exercise 2, decide which angle *on the paper* the servo should be pointing *to*, then use your data from exercise 4 to estimate what position to *set* the servo at.

Exercise 6

Now finish the project! Use your data and figure out how to calculate the servo position from the input voltage. There are many ways to solve this problem! Write down your solution for exercise 6.

Extensions

How can we make this even better?

1) Calibrate it so that ambient room lighting points at exactly 90 degrees
2) Make it portable by getting a battery pack
3) Make it self-calibrating by remembering the max and min voltages seen