Arduino for biologists and environmental scientists

Ed Baked (Ed.)

2025-02-13

Contents

Introduction

Vince Smith 2017

If data collection, analysis and publication are the central tenets of scientific practice, then to date, only two of these processes have been significantly transformed by the Open Science movement. Open Source software has become the prevailing paradigm to support scientific enquiry, providing a new level of transparency and repeatability to the scientific process. Programming and scripting skills are becoming the norm amongst researchers within the biological science community, creating a new wave of innovation and reusable free tools that were once the preserve of a handful of software developers. Likewise, Open Publishing and Open Access to scientific data has accelerated a culture of data sharing and reuse, pioneered by the likes of GenBank and related molecular databases. The Open Access movement reached an important tipping point in 2014 when it was estimated that more that 50% of all scientific publications published from XXX are freely available. Similarly, an increasing number of data repositories have been developed (e.g. O'Leary & Kaufman, 2011; Baker et al, 2015), supporting the specialist needs of select scientific communities and challenging the concept of traditional publication, via technologies like dataset DOI's and the concept of data papers (Chavan & Penev, 2011).

The development of hardware involved in the collection of data, has until recently been largely untouched by the transition towards Open Science. This founding step in the lifecycle scientific enquiry, is for the most part still the preserve of specialist industries, who typically build proprietary and expensive systems and hardware that are difficult to customise and repurpose. Data loggers, tracking technologies and camera traps are examples of commonly deployed hardware systems that are indispensable tools in the monitoring of biodiversity. In many cases the specialist needs of biologists are hinders by the tiny commercial markets for devices, prohibiting deployment and development of these systems to all but the mosts general and common applications. To fill this gap, the principles of modularity, reuse and open licensing which are now commonplace within the open source and open access movement, need to be applied to hardware development.

There is a common need for a low-cost, low-power, miniaturised, general purpose

computing platform that can fulfil the need for multiple common biological use cases. The rise of the maker movement, has popularised the open hardware Arduino platform which fulfils these needs. Arduino has a mature community of hardware and software developers. Coupled with the very low cost and wide availability of modular components, this platform has significantly lowered the barrier to entry for many would-be hardware developers who would otherwise be put off by the expense and expertise required to build specialist equipment.

This book is a foray into the rise of the maker movement for biologists. The introductory chapters cover an array of essential topics which develop the skills necessary to build open hardware devices. These are accompanied by a series of exemplar projects which put this theory into practice, highlighting the ease with which the Arduino platform can be repurposed for multiple uses. These use cases highlight why the transformation to Open Hardware is an essential component of the Open Science Movement. Many address neche scientific endeavours which could never be commercialised. For example, the chapter on building a snail heart rate monitor has a clear scientific objective which opens a new window into the biology of snails, but is unlikely to attract the attention of IBM, Google or Microsoft. Other projects have commercial equivalents available but at a significantly higher cost, and lack flexibility. One advantage of Arduino is that several of these projects could be combined into a single integrated device that addresses multiple questions, increasing the cost saving by an order of magnitude over independent commercial solutions. For example, a data logger can easily be integrated with a camera trap, helping to correlate behavior with environmental conditions. The chapter on the Open Source Data Logger (OSDL) project covers integration of multiple devices over large geographic distances to create a modular sensor network.

The book concludes with a short series of chapters on linking these devices to external platforms such as MatLab and Open Source Web content Management Systems. Linking a sensor network to the Internet, moves us toward the vision of the Internet-of-Things, in which potentially even the biotic and abiotic elements of an environment can be monitored continuously in real time. The potential application for this wealth of data are presently hard to conceive, but like any field exposed to the big data revolution, the outcome is likely to be transformative. Most predictives studies on the natural world, rely upon large and high quality datasets that track spatial and temporal changes and correlate these which environmental change. These data are typically time consuming to collect, involving months and sometimes years of repeated field studies that are exceptionally difficult and costly to sustain. The rise of automated monitoring has the potential to transform field studies, providing continuous data collection at ever improving levels of resolution. This takes the biological community a huge leap towards the goal of modeling the every facet of the biosphere, and being able to make predictive use of these data to address some of the long term challenges facing humanity, such as climate change, developing sustainable food resources and controlling the impact of invasive species.

The rise of the Open Hardware movement has the potential to add a final layer of accountability of transparency to the scientific process. In the past, scientific hardware has quite literally been a black-box, which is opaque to all but a handful of specialists who understand the constraints and limits of these systems. Often commercial interests reinforce this opacity, leading to patents that prevent others from further development. The rise of the maker movement, and particularly systems like Arduino, provide an opportunity to transform the business models of these industries in the same way that has happened for the Open Source and Open Access movement. Rather than the construction of paid-for systems that limit innovation and reuse, Open Hardware creates an opportunity for the makers of platforms to commercialise services around these systems. For example, within the Open Source software community, RedHat and Aquia sell services around open source products that they have contributed significantly to. Likewise within the Open Access movement, publishers increasingly monetise services around the peer review of scientific articles, rather rather than access to the article. These transformations in business models keep the underlying platforms free and open to innovation for most, while providing commercial levels of support with agreed levels of service provision for those that need it.

It is still early days for the Open Hardware movement, particularly in the biological sciences where specialist skills in niche areas are still required to develop and deploy systems. My hope is that books like this can help seed the same transformation which has taken place in other stages of the scientific process, so that the emphasis on development switches to the building of open platforms for hardware development, rather than bespoke products and proprietary systems.

Chapter 1

Electronics

1.1 Introduction

A basic knowledge of electronics is useful for creating Arduino based projects. The introduction to electronics presented here describes the basics needed to make the projects described in this book. When creating your own projects a more comprehensive reference is likely to be useful. Practical Electronics for Inventors (Scherz, 2007) is a recommended reference that explains clearly many electronic concepts and discusses a wide range of electronic components in depth. The projects in this book (with the exception of the automatic plant watering system) make use of battery power, and can therefore be considered as relatively safe. Working with mains electricity is a more dangerous proposition - always check with a qualified electrician that both what you plan to do, and that what you have built, is safe before working with it.

1.2 Current, Voltage and Resistance

Electric circuits consist of a power supply (battery, mains, etc) and a number of electronic components that effect the flow of the electricity. Some components act primarily to regulate the flow of electric current (such as resistors), others primary purpose is to turn electric energy into another form (such as a light bulb). In order to understand how these components work together it is necessary to have an understanding of how electricity behaves. The basic properties of an electric current are the current, voltage and resistance.

Table 1.1: Typical current values for a variety of electrical devices.

item	current
Light-emitting Dioide (LED)	20mA
100W Lightbulb	1A
Laptop computer	3A
Microwave	10A

1.3 Current

An electric current is a flow of negatively charged electrons through an electrical conductor. The electrons travel from the negative side of the power supply to the positive side through whatever circuit we have constructed. It should be noted that this 'electron flow' travels in the opposite direction to the 'conventional current'. The reason for this is historic: before our microscopic understanding of electrons was developed it was assumed that it was positive particles that moved through the circuit. In fact, the positive charged particles of the conductor remain static while the mobile negatively charged electrons move through it. Current is defined as the amount of charged particles that move through a conductor in a given period of time. The unit of measurement of charge is the Coulomb (C), and an electron has a charge of -1.602x10-19C. The current is measured in Amperes (A; amp), and is defined as the number of Coulombs of charge passing a point in a conductor in a given time interval. One Ampere is the current flowing when one Coulomb of charge passes a point in one second (Fig XX). Table XX shows some typical values of current for a variety of electrical devices.

```
item <- c("Light-emitting Dioide (LED)", "100W Lightbulb", "Laptop computer", "Microwav
current <- c("20mA", "1A", "3A", "10A")
table <- cbind(item, current)
knitr::kable(table, caption = "Typical current values for a variety of electrical devi
```

The number of electrons flowing through a circuit is not changed by the electrical components, so the total current is always conserved (Fig XX).

1.4 Voltage

Electronic components that are not sources of power convert electrical energy to other forms of energy, e.g. a light bulb converts electrical energy to thermal and light energy. Voltage is therefore not conserved through a circuit. We typically consider the negative terminal of the power supply to have a voltage of 0V. When using a 1.5V battery the full 1.5V is dissipated by the connected circuit.

This is the reason that short circuiting a car battery with a wire will often result in that wire heating up to the point that it melts. We will see later that we can use resistors to protect sensitive electronic components that would be damaged if exposed to the full power of the battery.

1.5 Resistance

Chapter 2

Arduino Hardware

2.1 Microcontollers

Microcontrollers are at the heart of many everyday electronic devices, from simple electronic toys to washing machines. But what is a microcontroller? Microcontrollers are best viewed as a miniature version of a home or office computer, they have a processing unit, various types of memory and methods for input and output. Compared to your computer though they may seem woefully under-powered. Take the Arduino Uno as an example, its processor runs at 16MHz rather than a couple of gigahertz, it has a flash memory of 32KB rather than several gigabytes. You may be surprised therefore to learn that microcontrollers are as popular as ever. The simple reason for this is that they are cheap and have a very low power consumption, and these two features make them suitable for a wide range of purposes. Low cost devices capable of computation, writing data to SD cards, and sharing data over a local network or the internet have huge potential for biological research. Being able to run such devices from batteries, linking to the world using mobile phone data connections or SMS could revolutionise ecological studies.

One of the advantages of using a microcontroller is that it is a multi-functional device, by writing a program to run on the microcontroller you can define what it does. The Arduino is an easily re-programmable microcontroller board that can be re-purposed using your computer and a USB cable. This makes an Arduino an incredibly useful piece of lab equipment. When it's not being used as a temperature and humidity monitor alongside a malaise trap it can be used to monitor plant growth using an ultrasonic distance detector, or even act as a miniature web server. The alternative to a programmable microcontroller is learning what the vast array of individual integrated circuit chips each do, and painstakingly finding a way to get them to work together, for a single purpose. While this is still a useful skill for electronic engineers and hobbyists it is a

daunting and time consuming new skill set for most biologists. The Arduino is intended to democratise access to microcontrollers, making their functionality available to interaction designers as well as artists and scientists. You will still need to learn some basics, but far less than you would need to know without the Arduino system.

Chapter 3

Arduino Software

3.1 Introduction to Arduino Programming

Programming or coding is the process of creating a set of instructions to be carried out by a computer. The micro-controller on the Arduino board understands instructions written in binary: strings of the digits 0 and 1. Thankfully we do not have to interact with the Arduino in this way, instead we use a higher level language based on C. High level languages allow us to write instructions for the micro-controller using commands that are (at least in general) understandable in English. The use of English words not only makes it easier to read code you (or somebody else) has written, it also makes it easier to remember the instructions you can give the Arduino allowing you to write code far more quickly. One thing to note however is that the grammar of English is rather flexible: the ongoing debate on the merits or otherwise of the Oxford Comma is just one example. In the language of the Arduino the grammar (or syntax in computer science speak) is rigidly fixed. You will at some point run across errors as seemingly trivial as a missing semi-colon that prevent your code from working. Just remember that the Arduino cannot think, it is just a device that follows your instructions. For that to happen your instructions need to be correct and precise.

So we know that the Arduino understands only 0s and 1s, but that we can program it using a high level language; how does one get converted into the other? The Arduino software includes a compiler - software that takes your high level work (the source code), checks it for errors, and then compiles it. Compiling is the process of turning source code into code that is executable (can be run on the Arduino). The Arduino software also handles the process of transferring the executable code onto the Arduino board over a USB cable.

It should be noted that the Arduino software provides a single interface to a number of tasks that would otherwise have to be performed independently:

source code compilation and writing of the executable. It is still possible to do these processes as separate steps, and if you become an expert at Arduino there are times when this process has its advantages. For the purposes of this book however all programming and writing can be performed using the simplified graphic user interface of the Arduino software.

3.1.1 A note on C //TODO: & C++ as Arduino uses both

While C is a high-level language it is not as abstract as other languages you may be familiar with such as Python or R. Python is the language of choice for many people using the Raspberry Pi - another cheap hardware device that may be used for many biological applications. Python code, unlike C, is not compiled but is interpreted another program runs the program that you have written. The advantages of this are that Python is a simpler language to learn, and generally more forgiving to beginners. The disadvantage is that the Python interpreter requires more resources than are available on the Arduino.

C is also a strongly-typed language, unlike Python and PHP. This means that it will not automatically convert integers into decimal (floating point) numbers for you. You must also specify the type of each variable that you use. This may seem a disadvantage but working this out takes memory, extra code and more processor cycles - all things that are lacking on resource-constrained microcontrollers.

That said I believe learning C in a low-resource environment such as the Arduino will in the long run make you a more competent programmer.

3.1.2 Git and GitHub

Many of the examples discussed here can be downloaded from the website GitHub. GitHub is built on git, a tool developed to allow for distributed version control. Git, GitHub and 'social coding' are discussed in a later chapter. For now you can just use the zip file download available from the GitHub page given the project.