CORE
Metadata, citation and similar papers at core.a
Provided by edoc

doi:10.2533/chimia.2015.172 Chimia 69 (2015) 172–175 © Schweizerische Chemische Gesellschaft

Instrumentino: An Open-Source Software for Scientific Instruments

Israel Joel Koenka[§*a], Jorge Sáiz[b], and Peter C. Hauser[a]

[§]SCS-Metrohm Award for best oral presentation

Abstract: Scientists often need to build dedicated computer-controlled experimental systems. For this purpose, it is becoming common to employ open-source microcontroller platforms, such as the *Arduino*. These boards and associated integrated software development environments provide affordable yet powerful solutions for the implementation of hardware control of transducers and acquisition of signals from detectors and sensors. It is, however, a challenge to write programs that allow interactive use of such arrangements from a personal computer. This task is particularly complex if some of the included hardware components are connected directly to the computer and not *via* the microcontroller. A graphical user interface framework, *Instrumentino*, was therefore developed to allow the creation of control programs for complex systems with minimal programming effort. By writing a single code file, a powerful custom user interface is generated, which enables the automatic running of elaborate operation sequences and observation of acquired experimental data in real time. The framework, which is written in *Python*, allows extension by users, and is made available as an open source project.

Keywords: *Arduino* · Computer-control of experiments · Data acquisition · Graphical user interface · Purpose-made instruments · *Python*

1. Introduction

Experimental scientists are often confronted with the need to build purpose-made instruments and experimental systems for investigating new scientific phenomena. Building such systems however, requires a range of technical skills, such as machining, electronics and computer programming, in which many scientists are not proficient. For this reason universities employ technical staff to assist in the building process. But while a mechanical workshop is common, and an electronics engineer is often available, the required programming efforts are usually left to the scientists themselves to fulfill. This has been realized by industry, and visual programming environments (such as LabVIEW) have been made available to alleviate some of the difficulties of programming electronically controlled hardware. Unfortunately, these tools are often very expensive and not always sufficiently easy to use.

In the past few years, a new trend has emerged among experimental scientists.[1–13] More and more research groups are inspired by the open-source hardware world, incorporating open-source microcontroller platforms in their systems, *Arduino*[14] being the best known of these.

Arduinos are small electronic printed circuit boards, carrying a programmable microcontroller that allows users to easily operate attached electronic devices by setting and reading voltage levels on its output and input pins. The *Arduino* programming environment provides all that is necessary to set the voltage levels in a predetermined sequence, but very little to interactively control the system or allow the user to monitor the acquired data in real time. This is usually essential for an experimental system. Moreover, experimental set-ups often require hardware units that are connected directly to the computer (*e.g.* through a USB port) and cannot be interfaced through an Arduino. The need for an easy way to create graphical user interface (GUI) programs for custom experimental set-ups was identified in our group, which led us to develop *Instrumentino*.

Instrumentino is an open-source framework for developing custom GUI programs to control experimental systems and instruments, while requiring only a minimal programming effort. The user only needs to provide a single system configuration file in which details about the set-up are given. The outcome is a fully functional and user-friendly graphical control program for that system which is comparable to commercial products. It allows the user to graphically control each individual component in the system, and to orchestrate their operation to create elaborate and automatic running sequences. Moreover, it graphically presents real-time acquired data, and automatically saves it for later analysis. Importantly, *Instrumentino* enables the simultaneous operation of several hardware controllers, which do not necessarily have to be *Arduino* boards.

Instrumentino was written in *Python*, a popular high-level programming language that is easier to learn and understand than most traditional languages (such as C). *Python* has also the advantage of being platform-independent. Free online courses to introduce non-programmers to the world of *Python* are available.[15]

Initially, *Instrumentino* was developed to answer the system control needs of our research group, in particular to operate purpose-made capillary electrophoresis instruments. Soon it became the standard for system control in our group, and more than a dozen projects have been realized in a relatively short time. A publication on *Instrumentino* has appeared in *Computer Science Communications*.[16] The source code is available at the GitHub repository hosting service,[17] and user-contributed additions may be deposited there as well.

*Correspondence: I. J. Koenka[a]
Tel.: +41 61 267 1061
E-mail: yoelk@tx.technion.ac.il
[a]Department of Chemistry
University of Basel
Spitalstrasse 51, CH-4056 Basel
[b]Department of Analytical Chemistry,
Physical Chemistry and Chemical Engineering
University of Alcalá
Ctra. Madrid-Barcelona Km 33.6
Alcalá de Henares 28871, Madrid, Spain

2. Controlling/Monitoring Physical Properties

To clarify how *Instrumentino* is used, a simple example system in which an *Arduino* is employed is given in Fig. 1. The dashed parts of the figure represent optional hardware connected *via* a USB interface to the computer. A more complicated example including third-party controllers is given elsewhere.[16] The example instrument is a small box, in which the temperature and pressure need to be controlled. These physical properties can be set in a closed system, by using a thermostat and a pressure controller respectively, both of which can be electronically controlled *via* an *Arduino*.

The way to electronically control a physical property is to combine a sensor and an actuator in a closed feedback loop. A proportional-integral-derivative (PID) thermostat, for example, is composed of a thermometer (sensor) and a heating element (actuator), which in this example are operated directly by the *Arduino*. The temperature is read using an analog input pin (pin A1) and the heating element is periodically turned on and off to reach the desired value using a digital pin (pin D2). The PID control feedback loop for the thermostat is closed in the *Arduino* itself, *i.e.* the actuator is controlled by the software running on the *Arduino* according to the measured sensor signal.

Another example is the control of pressure with a dedicated electronic pressure controller (*e.g.* OEM-EP, Parker–Hannifin, Cleveland OH, USA). The desired pressure can be set *via* the *Arduino* by supplying a voltage between 0 and 5 V proportional to the pressure within the possible range (1000 and 2000 mbar in the example), so 0, 2.5 and 5 volts will result in 1000, 1500 and 2000 mbar pressure respectively. Since most *Arduino* boards do not have analog output pins, an external digital to analog converter (D/A) converts a PWM (Pulse Width Modulation) digital signal from pin D3 to an analog voltage for the pressure controller. The actual pressure can be read from the pressure controller, similar to the temperature, as an analog signal (input pin A0).

3. PC-Arduino Interaction

The physical properties in the example are controlled and monitored by the *Arduino* I/O pins. In order to let the user issue commands to the *Arduino*, a program (called a *Sketch* in the *Arduino* environment) was written to run on the *Arduino* which constantly listens to incoming textual commands from the USB port of the computer and acts upon them. The *Sketch* was called *Controlino* to imply that it lets the user control an *Arduino* from the PC. For example, when the user types in 'Read A0' or 'Write D3 0', *Controlino* reads the voltage level on analog pin A0 and sends it back *via* the same serial connection, or sets pin D3 to 0 volts respectively. In the example, this would have the effect of reading the pressure, or setting it to its minimal value of 1000 mbar. On the PC side, the interaction is automated with *Instrumentino*.

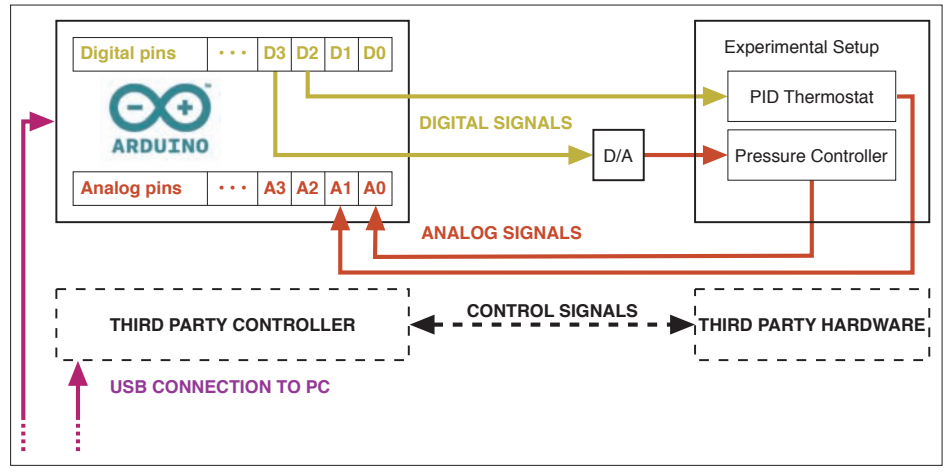

Fig. 1. A simple experimental system, composed of a thermostat and a pressure controller, which are controlled *via* an *Arduino*.

4. How *Instrumentino* Works

Instrumentino is a software platform, written in *Python*, for developing custom GUI programs to control scientific instruments. It releases the user from dealing with low-level technical details such as pin numbers and voltage levels.

4.1 *The Python Description File*

An experimental system may involve numerous components of different kinds, and each component in turn may control one or more experimental variables. The thermostat and pressure controller in the example (Fig. 1) are quite simple, having only one variable each (temperature and pressure respectively). Each of these variables are characterized by their operating range and their physical units (*e.g.* 1000–2000 mbar or 0–100 °C), and the required electrical connections for each parameter consist of two pins (an input pin and an output pin). This information is all that is necessary for translating high-level user commands such as 'set the pressure to 2000 mbar' to the low-level operation of setting pin D3 to output 5 volts. This information is provided to *Instrumentino* through a text file, which is written in *Python* and called the *System Configuration File*.

The system configuration is displayed on the left in Fig. 2. As can be seen, the system configuration data is composed of two parts. The first is a list of system components, accompanied by the information required for their operation. Appropriate *Python* classes describe the components and are provided with the essential information such as pin numbers, voltage ranges *etc*. If an appropriate class is not already present, it has to be added to *Instrumentino*. Once a new class has been added to the *Instrumentino* source code repository, everyone can use it, which is one of the benefits of being an open-source project.

The second list contains meaningful basic actions that the system is required to perform. These are short pieces of *Python* code (functions) that achieve basic tasks. For example, running the action 'Pressurize' (see Fig. 2) tells the system to set a certain pressure for a given amount of time, after which the pressure is released. Both the pressure and the time to wait are given as arguments to the action function. The user may define many actions with different levels of complexity to account for all of the functionalities of the set-up.

These two lists contain the core information on the system and represent all the required programming effort. The exact technical details on how to correctly define these lists can be found in documents and examples in the *Instrumentino* directory on GitHub.[17] The outcome is an executable *Python* file, which when run, starts a customized GUI program for the set-up defined.

4.2 *The Graphical User Interface*

Instrumentino was designed to provide a system-dependent GUI in a uniform format, using the information given in the system configuration file. The main window is divided into three sub-windows (views) as shown in Fig. 2: the component view, the automation view and the log view.

4.2.1 *The Component View*

The first view from the left is the component view, which provides the most basic way to access the components of the

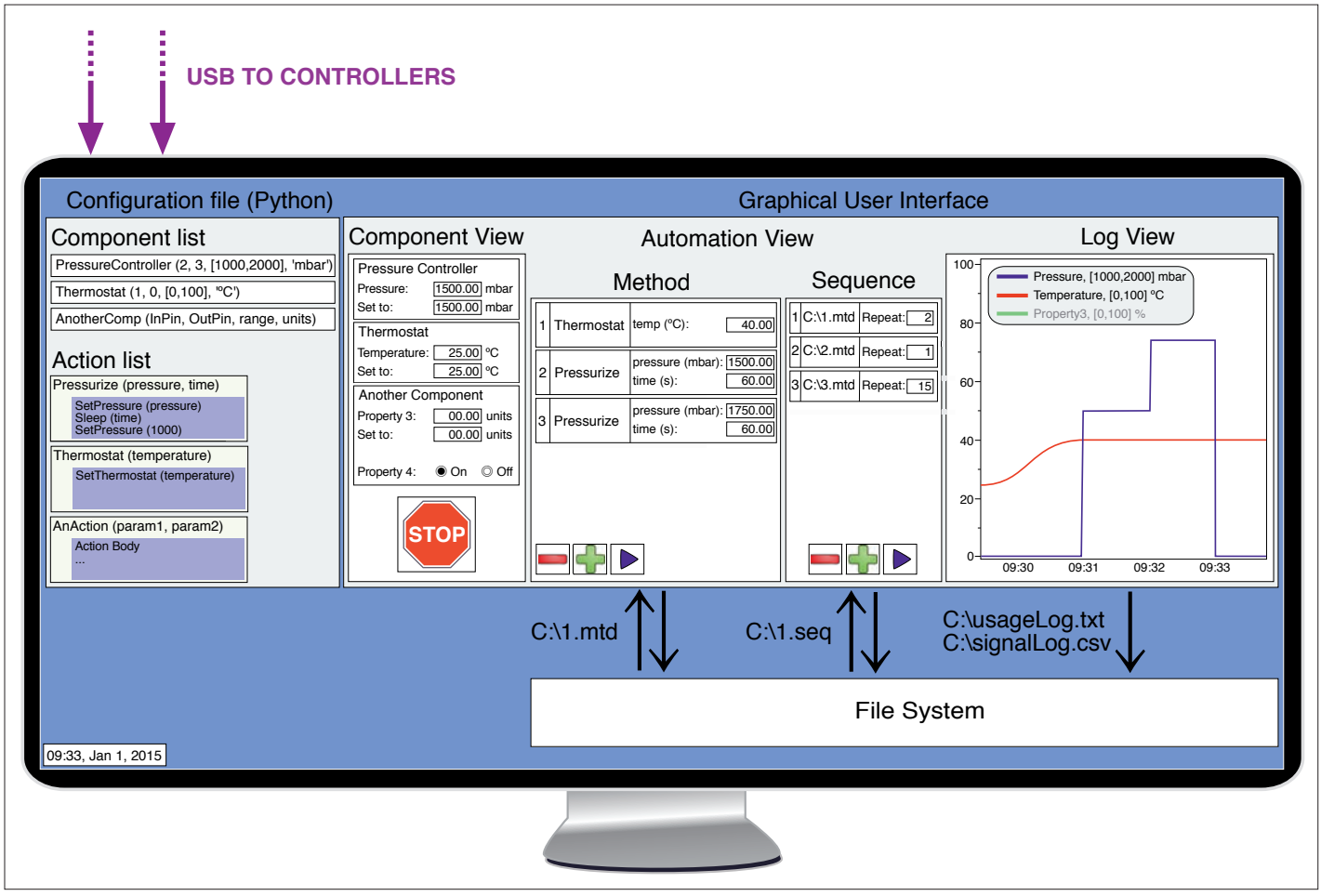

Fig. 2. Schematic diagram of how *Instrumentino* is used to control the example system in Fig. 1.

system. Each component is given a tile in which all of its controllable variables are shown with their current values (*e.g.* as read through the *Arduino*). The values of analog variables such as pressure and temperature can also be set in text boxes within the ranges defined in the system configuration file. Digital on/off-signals are shown as radio buttons and can also be set. It is also possible that a single component of the system may have several variables of different kinds.

4.2.2 *The Automation View*

The next view from the left is the automation view, in which *Methods* and *Sequences* can be defined and run. *Methods* and *Sequences* present two levels of automation. A *Method* is an extendable list of the actions defined in the system configuration file. The arguments given to the action functions (*e.g.* pressure and time) can be set by the user in matching text boxes, shown alongside the name of the action. In this way, an elaborate method can be defined to achieve a desired experimental goal. Once defined, a *Method* can be saved as a *.mtd* file for later use. This allows the user to define a library of *Methods* in order to save time and effort.

The next level of automation is the *Sequence* mode. A *Sequence* is simply a list of *Methods* from the user's *Method* library. Each item in the list can be defined to be repeated several times if necessary in order to build long operational runs that may last days or even months. This is particularly useful when certain experimental parameters need to be optimized. A series of *Methods* can be made to cover the relevant parameter space and automatically run without user intervention. Once finished, the overall results can be analyzed by inspecting the log view.

4.2.3 *The Log View*

The rightmost view is used to present the user with graphical and textual records of the usage of the experimental system. As long as the software is running, experimental data of system variables (*e.g.* pressure, temperature) is recorded on the computer and plotted in an overlaid manner as a time series. The log view in Fig. 2 shows the temperature and pressure traces after the example *Method* was started at 09:30. The temperature, which had been set to 40 °C in the first entry, increased over time from 25 °C and was followed by two pressure steps of 1500 and 1750 mbar (50 and 75% respectively).

Each trace is given a different combination of color and line thickness for distinction. As a highly complicated system may have many overlaid traces, it was made possible to turn their visibility on or off by clicking on the matching entry in the graph legend (in the example, the trace of 'Property3' is toggled off). All traces are plotted on a common vertical axis, reflecting the current values as a percentage of the relevant variable range. In case of bipolar variables (*e.g.* –50 to +100), positive results are plotted as solid lines and negative results as dashed lines (for the ranges 0 to +100 and 0 to –50 respectively). This allows to maintain a common zero point on the percentage scale for all variables, as otherwise the display can be very confusing. A set of graph orientation controls (not shown in Fig. 2) allows the user to freeze the timeline and zoom on different areas of interest in the graph.

Like the automation view, the log view can host either a graphical signal log or a textual operation log (not shown) which records the user's operations (*e.g.* 'run method 1.mtd @ 09:30, 1/1/2015'). The outputs of both logs are automatically saved as *.csv* and *.txt* files respectively, using the current date and time as the file name. The signal log can later be opened in any spreadsheet program to view the raw data. For the sake of simplicity digital signals are not shown on the screen but their values are saved in the signal log file.

5. Current Systems Employing *Instrumentino*

As mentioned earlier in the text, most of the projects using *Instrumentino* so far have concerned capillary electrophoresis instruments, built for different purposes. This includes dual-channel portable instruments for the analysis of fireworks[7] (Fig. 3) and for the environmental analysis of surface water near an old mine, a stationary system for unmanned automatic wastewater monitoring, a system for investigating electrophoresis dynamics by using an array of detectors, a system for handling sub-microliter sample volumes, and systems for on-line pre-concentration of diluted samples. *Instrumentino* has also been chosen to be used in projects outside our lab, such as for an automated coulometric micro-titrator, a low-cost apparatus for measuring thermo-mechanical rock properties, and even for teaching students in physics teaching labs.

Fig. 3. Photograph of the dual-channel portable CE system.[7]

Besides simplifying the construction and operation of purpose-made experimental arrangements, another possible benefit of using *Instrumentino* is cost savings. Instruments which are commonly used in laboratories and are usually bought from commercial suppliers may be built in-house at significantly lower cost. For example, an electronic control box for four mass flow controllers (MFC) was built in our group for as little as 50 CHF, while a similar commercial product would have cost about 1,000 CHF. MFCs are used in many laboratories for the exact setting of gas flows towards a reaction or measurement chamber. Each MFC (*e.g.* from MKS Instruments, Andover MA, USA) has a connector with analog input and output pins for setting the desired mass flow and reporting the current flow value respectively. These devices can be controlled by an *Arduino* as easily as the pressure controller of Fig. 1. Therefore, an *Arduino*-based interface was constructed in our group to control up to four MFCs simultaneously as seen in Fig. 4 and this has been used for different purposes. The design can easily be extended to more MFCs (*e.g.* up to 16 different MFCs when using an *Arduino Mega*[18]). In combination with *Instrumentino* this control box enables the user to control the gas flows in software to conduct experiments automatically.

6. Conclusions

Open-source hardware and software enables researchers with limited expertise in electronics and programming to construct complex apparatus, offering opportunities for substantial cost savings to research labs.[6] *Instrumentino* is one of many such tools to assist in the building of custom-made and routinely used appliances for conducting research. Since its development it has been incorporated in many projects, enabling their quick realization, and it has drawn the attention of peers elsewhere. As a free open-source project, *Instrumentino* has the potential to grow. The authors invite fellow experimental scientists to join the open source development of *Instrumentino*, for a greater mutual benefit.

Acknowledgements

The authors are grateful for financial support by the Swiss National Science Foundation through grants 200020-149068 and IZK0Z2-157622/1.

Conflicts of interest

The authors have declared no conflict of interest.

Received: January 14, 2015

[1] G. C. Anzalone, A. G. Glover, J. M. Pearce, *Sensors* **2013**, *13*, 5338.
[2] E. T. da Costa, M. F. Mora, P. A. Willis, C. L. do Lago, H. Jiao, C. D. Garcia, *Electrophoresis* **2014**, *35*, 2370.
[3] C. Galeriu, *The Physics Teacher* **2013**, *51*, 156.
[4] G. Gasparesc, '36th International Conference on Telecommunications and Signal Processing (TSP)', **2013**, p. 340, DOI:10.1109/tsp.2013.6613948,
[5] J. A. Kornuta, M. E. Nipper, J. B. Dixon, *J. Biomech.* **2013**, *46*, 183.
[6] J. M. Pearce, *Science* **2012**, *337*, 1303.
[7] J. Sáiz, M. T. Duc, I. J. Koenka, C. Martín-Alberca, P. C. Hauser, C. García-Ruiz, *J. Chromatogr. A* **2014**, *1372*, 245.
[8] J. Sáiz, M. Thanh Duc, P. C. Hauser, C. Garcia-Ruiz, *Electrophoresis* **2013**, *34*, 2078.
[9] A. H. Shajahan, A. Anand, 'International Conference on Energy Efficient Technologies for Sustainability (ICEETS)', **2013**, p. 241.
[10] M. Thanh Duc, P. Thi Thanh Thuy, P. Hung Viet, J. Sáiz, C. Garcia Ruiz, P.C. Hauser, *Anal. Chem.* **2013**, *85*, 2333.
[11] V. Velusamy, K. Arshak, O. Korostynska, A. Al-Shamma'a, *Key Eng. Mat.* **2013**, *543*, 47.
[12] G. S. Zahn, C. Domienikan, R. P. M. Carvalhaes, F. A. Genezini, 'XXXV Brazilian Workshop On Nuclear Physics', **2013**, p. 141.
[13] M. Zolkapli, S. A. M. Al-Junid, Z. Othman, A. Manut, M. A. Mohd Zulkifli, 'International Conference on Technology, Informatics, Management, Engineering and Environment (TIME-E 2013)', **2013**, p. 43.
[14] *Arduino*, *http://www.arduino.cc*, retrieved Jan. 2015.
[15] B. Klein, online Python course, *http://www.python-course.eu/*, retrieved Jan. 2015.
[16] I. J. Koenka, J. Sáiz, P. C. Hauser, *Comput. Phys. Commun.* **2014**, *185*, 2724.
[17] I. J. Koenka, Instrumentino repository, *https://github.com/yoelk/Instrumentino*, retrieved Jan. 2015.
[18] *Arduino Mega*, *http://arduino.cc/en/Main/arduinoBoardMega*, retrieved Jan. 2015.

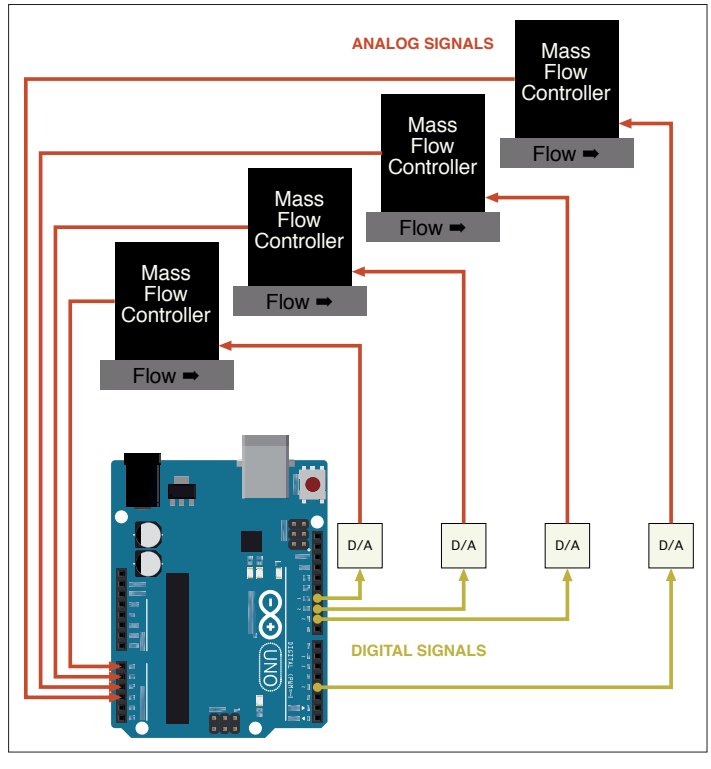

Fig. 4. Schematic diagram of the electrical connections for an MFC control box.