Scientific Research Publishing

Advances in Internet of Things, 2021, 11, 42-58
https://www.scirp.org/journal/ait
ISSN Online: 2161-6825
ISSN Print: 2161-6817

Python Software Integrates with Microcontrollers and Electronic Hardware to Ease Development for Open-Source Research and Scientific Applications

Daniel K. Fisher[1*], Reginald S. Fletcher[2], Saseendran S. Anapalli[1]

[1]United States Department of Agriculture, Agricultural Research Service, Sustainable Water Management Research Unit, Stoneville, Mississippi, USA
[2]United States Department of Agriculture, Agricultural Research Service, Crop Production Systems Research Unit, Stoneville, Mississippi, USA
Email: *daniel.fisher@usda.gov

How to cite this paper: Fisher, D.K., Fletcher, R.S. and Anapalli, S.S. (2021) Python Software Integrates with Microcontrollers and Electronic Hardware to Ease Development for Open-Source Research and Scientific Applications. *Advances in Internet of Things*, **11**, 42-58.
https://doi.org/10.4236/ait.2021.111004

Received: September 25, 2020
Accepted: January 25, 2021
Published: January 28, 2021

Abstract

Many options exist for developing and implementing monitoring systems for research and scientific applications. Commercially, available systems and devices, however, are usually built using proprietary tools and programming instructions, and often offer limited flexibility for end users. The use of open-source hardware and software has been embraced by the research and scientific communities and can be used to target unique data and information requirements. Development based on the Arduino microcontroller project has resulted in many successful applications, and the Arduino hardware and software environment continues to expand and become more powerful but can be intimidating for users with limited electronics or programming experience. The open-source Python language has gained in popularity and is being taught in schools and universities as an introduction to computer programming and software development due to its simple structure, ease of use, and large standard library of functions. A project called CircuitPython was developed to extend the use of Python to programming hardware devices such as programmable microcontrollers and maintains much of the original Python language and features, with additional support for accessing and controlling microcontroller hardware. The objective of the work reported here is to discuss the CircuitPython programming language and demonstrate its use in the development of research and scientific applications. Several open-source sensing and monitoring systems developed using open-source hardware and the open-source CircuitPython programming language are presented and described.

Keywords

CircuitPython, Python, Arduino, Agriculture, Monitoring System, Sensors

1. Introduction

Numerous options exist for developing and implementing sensing and monitoring systems for research and scientific applications. Many electronic equipment manufacturers and vendors offer instruments, instrument systems, and complete monitoring systems for a variety of applications. These systems and devices, however, are usually available in limited configurations designed by the vendor, are built using proprietary tools and programming instructions, and often offer limited flexibility for end users.

The use of open-source hardware and software has been embraced by the research and scientific communities as a means of developing bespoke sensing and monitoring systems to target unique data and information requirements and achieve specific research goals [1] [2] [3]. The open-source Arduino project (https://arduino.cc) consists of a hardware component, a microcontroller development circuit board, and a software component, an Integrated Development Environment (IDE). The Arduino has proven to be a popular and powerful development tool for researchers and scientists, as well as hobbyists, artists, and other makers. Electronic devices and instrumentation have been developed in many different disciplines, such as agriculture [4] [5] [6] [7] [8], irrigation and water management [9] [10] [11] [12], robotics [13] [14] [15], scientific and environmental studies [16] [17] [18] [19], and multipurpose laboratory and field data collection [20] [21] [22] [23] [24].

Development using the Arduino project has resulted in many successful applications, and the Arduino hardware and software environment continues to expand, include additional features, and become more powerful [25]. The original hardware, based on an 8-bit microcontroller, with limited features and a slow processing speed, has been augmented with much faster, feature-rich, and more powerful 32-bit microcontrollers. The software programming environment undergoes continuous upgrades to offer additional features and support for new hardware. A few issues regarding the Arduino IDE have contributed to the success of the Arduino project, but also to some intimidation for users, especially those new to microcontrollers and computer programming. The IDE supports programming based on the C/C++ computer language, which allows powerful and fast-executing programs to be written and deployed. For more than simple programs, however, a fair amount of in-depth knowledge of the language is required. In addition, the Arduino IDE must be downloaded (https://arduino.cc) and installed on a computer, configured by installing additional utilities to support specific microcontroller hardware, and various hardware drivers often must also be installed and special cables used to interface and interact with the mi-

crocontroller.

Much progress in computer software development has been made due to the C/C++ languages, but other programming languages exist and have been gaining in popularity. The open-source Python language (https://python.org) has gained many users and is being taught in schools and universities as an introduction to computer programming and software development due to its simple structure, ease of use, and large standard library of functions. A project called MicroPython (https://microcpython.org) was developed to extend the use of Python to programming and controlling hardware devices such as programmable microcontrollers. MicroPython maintains much of the original Python language and features, with additional support for accessing and controlling microcontroller hardware.

CircuitPython (https://circuitpython.org) is an open-source programming language derived from MicroPython and was developed by Adafruit Industries (https://adafruit.com) to be easier to learn and use. Different from MicroPython, CircuitPython was developed to support a limited and specific subset of microcontrollers, specifically the 32-bit ARM Cortex M0 SAMD21 and ARM Cortex M4 SAMD51 microcontrollers (Microchip Technology; https://www.microchip.com) and the 32-bit nRF52840 ARM Cortex M4 microcontroller (Nordic Semiconductor; https://www.nordicsemi.com). While the number of supported microcontrollers is limited, unique development boards from a variety of manufacturers based on these microcontrollers and compatible with CircuitPython number approximately 150 (https://circuitpython.org/downloads).

The objective of the work reported here is to discuss the CircuitPython programming language and demonstrate its use in the development of research and scientific applications. Several open-source sensing and monitoring systems developed using open-source hardware and the open-source CircuitPython programming language are presented and described.

2. Methods and Materials

2.1. Software

CircuitPython is based on the open-source Python 3 programming language and has been ported to run on several specific microcontrollers. CircuitPython is a Python compiler that is loaded onto the microcontroller hardware and runs and interprets Python code. The compiler includes core, standard Python 3 libraries and modules that allow access to mathematical functions, text and string manipulation and parsing, internet protocols, and file and operating-system interaction. Additional modules offer access to the microcontroller hardware to interact with and manage various hardware features, such as input/output pins, analog and digital signal processing, analog-to-digital converters, serial ports, and digital (I2C, SPI) communications protocols. Supplemental libraries and modules are available to support interfacing the microcontroller with external peripherals, such as sensors, displays, real-time clock/calendars, data storage devices,

GPS receivers, cameras, motors, and Bluetooth, wifi, and other wireless devices (https://circuitpython.org/libraries).

CircuitPython is installed on a microcontroller development board via a bootloader utility unique to each microcontroller board. The bootloader is downloaded in the form of a USB Flashing Format (uf2) file for the specific microcontroller from the CircuitPython repository (https://circuitpython.org/downloads). The microcontroller board is connected to a host computer with a standard micro-USB cable. The microcontroller is put into a bootloading mode by pressing a button on the microcontroller development board, and the board appears on the host computer as a removable flash drive. The user then copies the uf2 file to the board's flash drive via the micro USB cable, the file structure is created, and the CircuitPython compiler and core libraries are copied to the drive.

Writing a microcontroller program in CircuitPython is similar to writing a program in Python: the program is written as a text file using any text editor, and the file is saved with a ".py" extension. Text editors used by the Python community, such as Mu (https://codewith.mu) and Atom (https://atom.io), support several versions of Python, including CircuitPython, and offer real-time interaction for writing and testing code, debugging, and plotting. When a program is written, the file is named and it, and supporting libraries, are saved to the board's flash drive. Multiple program files and libraries may be stored on the flash drive. Whenever the microcontroller is powered on, the CircuitPython compiler searches for a file named "code.py" and, if found, immediately runs that file. The user can run any of multiple stored programs by simply changing the program's filename to code.py.

Testing and modifying programs in CircuitPython are different than when using the Arduino IDE. Using the Arduino IDE, the user makes code changes, then must recompile the program and upload the recompiled program to the microcontroller. In CircuitPython, the user enters code or makes changes to the code.py program file using the text editor, then only needs to save the modified file. The microcontroller detects the presence of the modified file, automatically resets itself and restarts, and begins execution of the new code.py file. This allows for user interaction with the microcontroller and its programming without the need for a specific programming environment, like the Arduino IDE: the user can modify CircuitPython with any text editor on almost any computer, tablet, or even smartphone.

2.2. Hardware

CircuitPython is suitable for programming to interface and control a variety of electronic hardware, including microcontrollers, sensors, and communications and other peripheral devices. While many and varied microcontrollers are available from a variety of manufacturers, CircuitPython is targeted and optimized to run on Microchip Technology 32-bit ARM Cortex M0 SAMD21 and ARM Cortex M4 SAMD51 microcontrollers and the Nordic Semiconductor

32-bit nRF52840 ARM Cortex M4 microcontroller. The main differences between the microcontrollers are in processing speed and memory size: the ATSAMD21 operates at 48 MHz, with 256 KB of flash and 32 KB of RAM; the ATSAMD51 operates at 120 MHz, with 512 KB flash and 192 KB RAM; and the nRF52840 operates at 64 MHz, with 1 MB flash and 256 KB of RAM. The nRF52840 microcontroller also different in that it has a built-in Bluetooth Low Energy 2.4 GHz radio which the user can access and program.

Microcontroller development boards consist of a microcontroller and auxiliary electronic components mounted on a circuit board that breaks out the multiple input/output pins of the microcontroller, making the pins and built-in features conveniently accessible to the user. Built-in features include: multichannel analog-to-digital converters for measuring analog voltages; multiple digital communications ports, including serial, I2C, and SPI, for interfacing digital sensors and peripheral devices; timers and clocks for event interval and pulse measurement and timekeeping; pulsewidth modulation for controlling motors and lights; and USB support for interfacing with a host computer. On boards explicitly designed for use with CircuitPython, a flash memory chip is incorporated to store the Python interpreter and programming and library files, resulting in a self-contained programming and run-time device.

While many manufacturers offer microcontroller development boards, discussion here is limited to boards produced by Adafruit Industries (https://adafruit.com) and Sparkfun Electronics (https://sparkfun.com), with which the authors are familiar and have experience developing in both Arduino and CircuitPython programming environments. Microcontroller boards, as well as sensors and peripheral components from these manufacturers, are well documented and supported by their respective manufacturers, and are available in many parts of the world via online electronics retailers. The discussion is applicable to devices from other manufacturers, but the user would need to identify and install the appropriate uf2 bootloader file and specific libraries for the exact hardware used.

The original Arduino development board had a specific form factor (size of the board, spacing of input/output pins, and arrangement of specific pins and features) that allowed for interchangeability of boards from different manufacturers, and for the development of plug-in boards, called shields. While that form factor is still in use and supported by many manufacturers, the Feather series of development boards (Adafruit Industries) introduced a new physical form factor, different from the original Arduino microcontroller board. The new form factor has been adopted by several other manufacturers.

A wide range of sensors and peripheral device are available, from Adafruit Industries, Sparkfun Electronics, and many other manufacturers, that interface easily with microcontrollers and are supported by CircuitPython libraries. Similar to Arduino shields, peripheral plug-in boards for Feather boards, called wings, allow, for example, external sensors, clock/calendars, GPS receivers, wifi, Bluetooth, cellular, and other wireless modules, and SDcard storage devices, to

plug directly into the microcontroller board and share pins. No external circuitry is required by the user to develop powerful sensing and monitoring systems.

3. Example Applications

Electronic monitoring systems have been developed to support research related mainly to agricultural and water-management efforts by researchers with the United States Department of Agriculture's Agricultural Research Service at the Jamie Whitten Delta States Research Center, Stoneville, Misissippi USA. Several of these systems are described in the following sections. In the spirit of open-source collaboration and sharing of information, all hardware and software details, including CircuitPython code, are freely available by contacting the authors.

3.1. Desktop Weather Station

A simple "weather station" was developed to access and display a time-series of weather data using internet-based services. The desktop weather station is a virtual weather station that connects to the internet over a wifi network and accesses weather data from an internet service. Weather data are downloaded and displayed on a data-hosting website.

A number of internet-based services provide current and historical weather data and weather forecasts, often at no cost to the user. Services such as OpenWeather (https://openweathermap.org), Meteostat (https://meteostat.net), weatherstack (https://weatherstack.com), and AccuWeather (https://accuweather.com) allow users access to weather data via a simple Application Programming Interface (API). The API consists of a URL (essentially a website address) containing information such as specific geographic location, units of measure, time period, and user account number embedded in the URL. Upon sending the URL, weather data and descriptive information are returned, in a specific JSON-formatted response, and the user parses the response to retrieve individual data values.

For this project, the OpenWeather service (https://openweathermap.org) was selected based on its simple API and ease in parsing the formatted data response. Upon registering on the OpenWeather website, a user account is created, and a unique user API key is assigned. The API allows the user to specify geographic location by city name, postal code, or latitude and longitude coordinates, and units as metric or imperial. To access data, the user assembles and sends a URL consisting of OpenWeather's base URL, geographic location, measurement units, and user API key, as shown in the following example:
http://api.openweathermap.org/data/2.5/weather?zip=38776&units=imperial&APPID=user_api_key.

A variety of internet-based data-hosting services are available that allow users to transfer data from remote locations and store, view, and share data via a web browser. The ThingSpeak platform (https://thingspeak.com) offers no-cost, limited-use access, and users can select and customize output formats for dis-

playing their data in graphical and text formats. Interaction with ThingSpeak is similar to interaction with the OpenWeather website in that an API defines the format of a URL, which the user assembles and sends to the website. To use the ThingSpeak service, the user registers on the ThingSpeak website, an account is created, and a unique user API key and data channel number (webpage address) are assigned. The user configures the data channel/webpage by defining the number of data values to be uploaded and the graphical format of the data displays. Data are uploaded to the website in a manner similar to that for downloading OpenWeather data, by assembling a URL including the ThingSpeak base URL, user API key, and data values. The data can then be viewed via a web browser by specifying the channel number on the ThingSpeak website.

The Desktop Weather Station hardware consists of a Feather M4 Express microcontroller development board (Adafruit Industries) and an AirLift FeatherWing ESP32 WiFi Co-Processor board (Adafruit Industries). The two boards are designed with the same physical form-factor to mate directly and share pins, with no external circuitry required. The two boards, shown in **Figure 1**, plug together using male header pins soldered to one board and female headers soldered to the other. The assembled boards share electrical power via a microUSB cable and power supply. Cost of the hardware components totaled US$ 42.

The Desktop Weather Station microcontroller was programmed in the CircuitPython programming language. The CircuitPython uf2 bootlader file for the Feather M4 Express board was downloaded from the CircuitPython repository (https://circuitpython.org) and loaded onto the microcontroller board. CircuitPython libraries were also downloaded (https://circuitpython.org/libraries), and specific modules needed for the project were stored on the flash drive on the board. Programming was accomplished using the open-source Mu code editor.

The program begins by importing libraries needed to set up digital communications between microcontroller and Airlift ESP32 and configure wifi functions. Specific pins which interconnect the microcontroller and the ESP32, and credentials required for logging on to the user's wifi network (network name, login, and password), accessing the OpenWeatherMap website (user API key), and uploading data to the ThingSpeak website (user API key), are defined. Wifi services

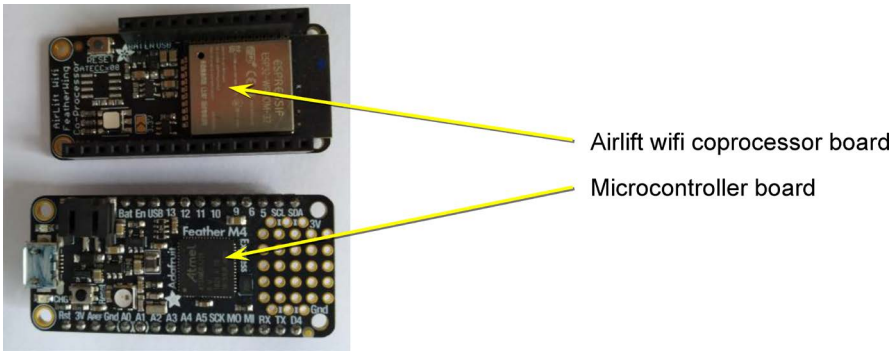

Figure 1. Desktop weather station hardware components.

are then enabled and the Airlift ESP32 connects to the user's wifi network, providing access to internet sites and services.

Weather data are downloaded from the OpenWeatherMap website by assembling a URL as described above and sending the URL via the internet. The response is received as a JSON-formatted string, which appears to CircuitPython as a Python dictionary containing key-word and data-value pairs. The response is parsed to extract measurements of air temperature, relative humidity, wind speed, atmospheric pressure, precipitation, and cloud cover. The program will make up to three attempts to retrieve data, in case an error occurs and an incomplete or no response is received.

The weather data are then uploaded to the ThingSpeak website by assembling a URL containing the weather data values. An example of a ThingSpeak URL for uploading two data values would be of the form https://api.thingspeak.com/update?api_key=user_API_key&field1=12.3&field2=4.56. A sample of weather data downloaded from OpenWeatherMap for the Stoneville, Mississippi USA location during a seven-day period in August 2020 and uploaded to ThingSpeak is shown in Figure 2.

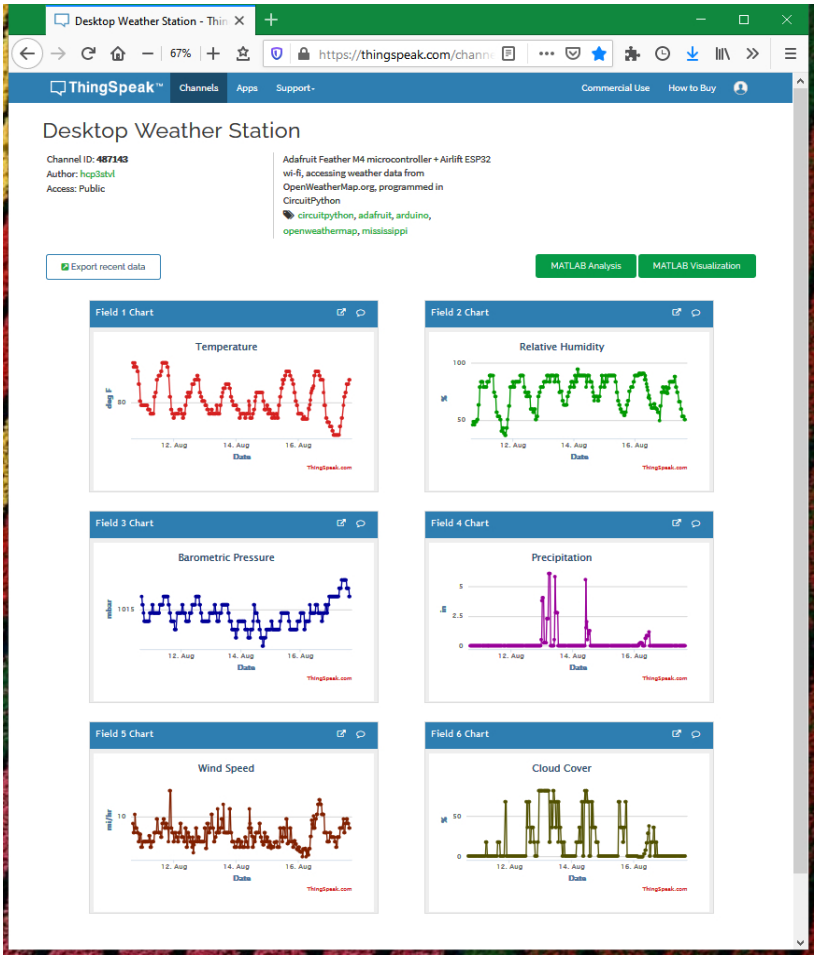

Figure 2. Time-series of weather data downloaded from OpenWeather and displayed on the ThingSpeak website.

3.2. Cellular Weather Station

The Cellular Weather Station, different from the virtual Desktop Weather Station, is a physical device deployed in the field to monitor actual weather conditions. The device consists of a microcontroller development board, weather sensors, cellular modem, and rechargeable battery power. The microcontroller was programmed in CircuitPython to periodically take measurements of atmospheric temperature, humidity, and pressure, and light level. Data are transmitted to the internet-based ThingSpeakdata-hosting website via the cellular communications network.

The circuit is based on an ItsyBitsy M4 Express microcontroller development board (Adafruit Industries) and a SIM7000 LTE CaT-M1/NB-IoT + GPS cellular shield (Botletics; https://botletics.com). The ItsyBitsy circuit board has a different physical format from that of the Feather microcontroller board but is based on the same Cortex M4 microcontroller. The SIM7000 cellular shield features a SIMCom 7000 LTE Cat-M1 cellular module (SIMCom; https://www.simcom.com), which interfaces with the microcontroller over a standard two-wire serial port. The SIM7000 module operates on the recently commissioned LTE Cat-M1 cellular network (https://www.3gpp.org), which is built on the existing 4G network. The LTE Cat-M1 network is designed for low-frequency, small-packet data transfer, sometimes referred to as Internet of Things, applications. To enable cellular data transfer, cellular network service is enabled via a data plan purchased from Hologram (https://hologram.io).

Two sensor breakout boards, each containing a solid-state sensor and auxiliary electronic circuitry, interface with the microcontroller to provide measurements of several weather parameters. The BME280 Temperature, Humidity, and Pressure breakout board (Adafruit Industries) measures air temperature, relative humidity, and atmospheric pressure using a BME280 Temperature, Humidity, and Pressure sensor (Bosch Sensortronics; https://www.bosch-sensortec.com). The TSL2591 Digital Light Sensor (Adafruit Industries) measures light level using a TSL2591 Light to Digital Converter (ams; https://ams.com). The two sensors interface with the microcontroller via I2C (Inter Integrated Circuit), a two-wire digital communications protocol: both sensors connect to the microcontroller's I2C port and the microcontroller communicates with and controls each sensor using the sensor's unique I2C address.

To enable long-term, battery-powered operation, microcontrollers and accompanying circuitry are often placed in a low-power sleep mode, waking periodically to perform circuit functions, then returning to sleep. CircuitPython, however, is currently not able to provide this feature. Instead, a hardware solution is incorporated into the circuit to reduce power consumption and enable operation under battery power. The TPL5110 Low-Power Timer (Sparkfun Electronics; https://sparkfun.com) is installed between the microcontroller and the 2500-mAh Lithium Ion Polymer Battery (Adafruit Industries). The low-power timer regulates electrical power from the battery to the microcontroller

and sensor circuit. At periodic intervals, the TPL5110 energizes and switches battery power on to the microcontroller and circuit. The CircuitPython program begins execution, reading sensors and transmitting data via the cellular network. At the end of program execution, the microcontroller sends a signal to the TPL5110, which completely turns off battery power to the circuit. The interval at which power is turned on and off, ranging from milliseconds to 2 hours, is determined easily by the user by configuring a set of switches installed on the TPL5110 circuit board. A schematic of the electrical circuit is shown in **Figure 3**.

The electrical circuit was then used to fabricate a circuit board. Female headers were located and soldered to an Arduino ProtoShield (Sparkfun Electronics), which has the same form-factor as that of the Botletics cellular module. Male header pins were soldered to the microcontroller board and each of the sensor and low-power timer boards. A functioning circuit board, shown in **Figure 4**, resulted by plugging all the components together. To install the circuit in the field, the assembled circuit was placed inside a protective electronics enclosure (Adafruit Industries), which was then inserted into a Solar Raditation Shield (AcuRite, https://www.acurite.com) to protect from direct exposure to sunlight. Total cost of the Cellular Weather Station components was approximately US$ 160. The monthly charge for the hologram.io cellular data plan was US$ 0.60.

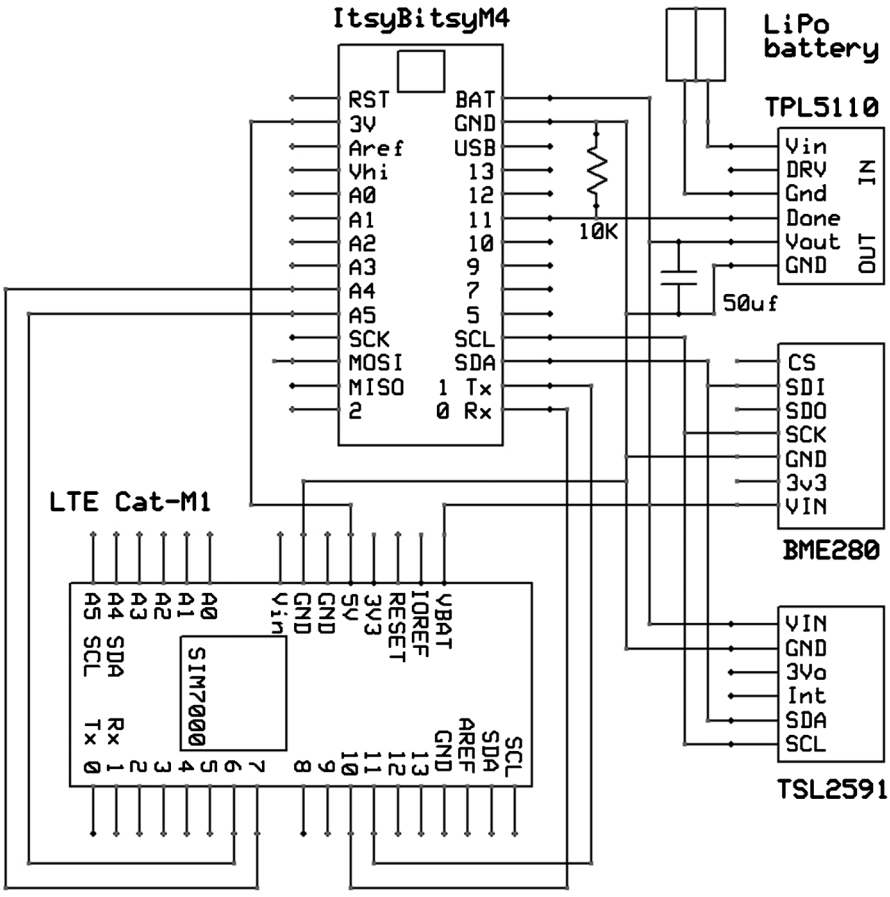

Figure 3. Electrical schematic of the cellular weather station.

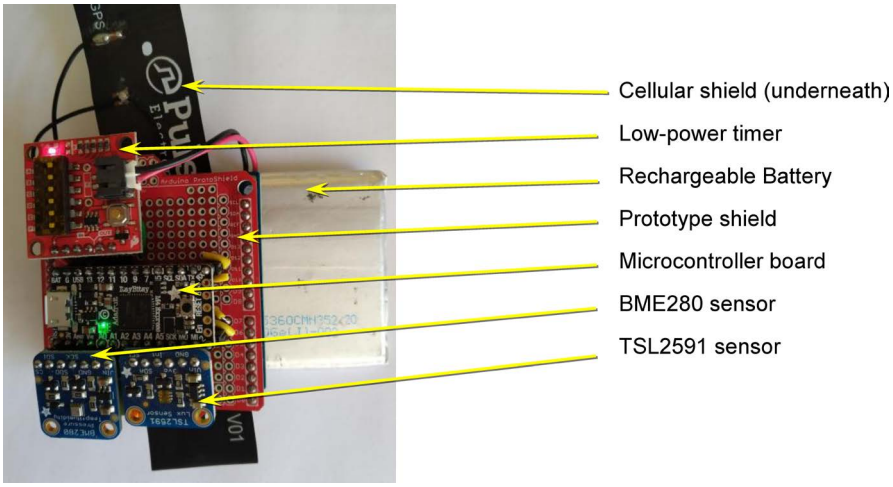

Figure 4. Cellular Weather Station hardware components.

The microcontroller was programmed in a manner similar to that described for the Desktop Weather Station described above. The CircuitPython uf2 file for the ItsyBitsy M4 Express board was loaded onto the microcontroller board, specific modules needed for communicating via the I2C protocol and reading the BME280 and TSL2591 sensors were downloaded and copied to the flash drive. Programming was accomplished using the open-source Mu code editor.

The program begins by importing libraries needed to set up serial and I2C communications and enable the BME280 and TSL2591 sensors. Specific pins which interconnect the microcontroller and cellular shield and TPL5110 are defined, and serial and I2C communications protocols and sensors are configured and started.

The two sensors interface with the microcontroller using the same two I2C pins, and each sensor is read in turn. The microcontroller first reads the BME280 sensor by sending its unique I2C address to specify that individual sensor, then sends a series of commands to retrieve sensor measurements of air temperature, humidity, and pressure. The microcontroller then reads the TSL2591 similarly, sending its I2C address and commands to retrieve a light level measurement.

To send data to the ThingSpeak website, the cellular module is powered on, and a connection between the microcontroller and cellular module is established via a two-wire serial port. Control of the cellular module is accomplished via standard Hayes AT commands, which were developed for computer modems in the early 1980s and still in use today by many serial devices. Commands are sent for the module to register on the cellular network, specify the access point for the hologram.io cellular service, and enable data and internet services. Sensor data are sent by assembling a URL as previously described for the Desktop Weather Station, and the URL is sent. Data and internet services are then terminated, and the module is detached from the cellular network and powered down. The circuit is then put into a low-power state by sending a signal to the TPL5110 low-power timer. The TPL5110 disconnects the battery from the circuit, and no battery power is used by the circuit until the next measurement

interval.

A sample of weather sensor data from the Cellular Weather Station is shown in Figure 5. The data were collected from the same location and time period as downloaded by the Desktop Weather Station, shown in Figure 2.

3.3. Cellular Soil-Moisture Monitor

Soil-moisture sensors are commonly used to monitor the soil-water status in the root zone of agricultural crops. Knowledge of soil-moisture status is used to evaluate water use by the crops and to offer guidance for water management and determining the proper timing of irrigation water application. A monitoring system was developed previously [25] to read and transmit soil-moisture sensor measurements and was designed to allow remote access of sensor data via the cellular communications network. The existing monitoring system, consisting of a microcontroller-based circuit, soil-moisture sensors, and a cellular modem, was modified for operation under CircuitPython programming.

The new monitoring system is very similar in design and operation to the Cellular Weather Station described above. Both systems are based on the Itsy-Bitsy M4 Express microcontroller development board (Adafruit Industries),

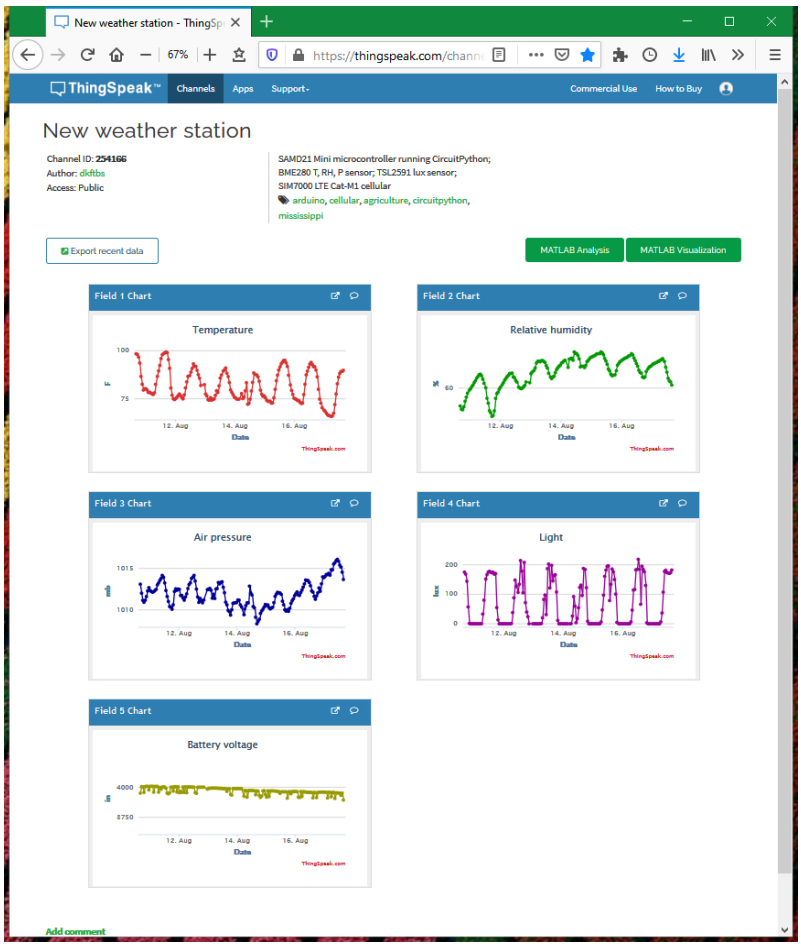

Figure 5. Time series of weather data collected with the cellular weather station.

SIM7000 LTE CaT-M1/NB-IoT + GPS cellular shield (Botletics), and TPL5110 Low-Power Time (Sparkfun Electronics) and rechargeable battery. Slight circuit modifications are made to replace the weather-related sensors with circuitry to interface and read soil-moisture sensors.

For the soil-moisture monitor, four soil-moisture sensors, Watermark Model 200-SS (Irrometer Company, Inc.; https://irrometer.com), interface with the microcontroller via half-bridge (voltage-divider) resistor networks. Each Watermark sensor acts as a variable resistor, with resistance proportional to sensor water content, and the sensor and a fixed-value (10 k ohm) resistor form a half-bridge network. When a voltage is applied to the sensor network, the voltage at the center of the half-bridge is measured with the microcontroller's built-in analog-to-digital converter, and the resistance of the Watermark sensor can be calculated. An electrical schematic of the soil-moisture monitor is shown in Figure 6.

A circuit board was fabricated based on the electrical schematic, with similar components and methods as that used for the Cellular Weather Station. In place of the weather sensors, a Spring Terminal Block (Adafruit Industries) was soldered to the ProtoShield to connect the soil-moisture sensors. The circuit components are shown in Figure 7, and the Cellular Soil-Moisture Monitor had a total cost of approximately US$ 120. The monthly charge for the hologram.io cellular data plan was US$ 0.60. The Watermark soil-moisture sensors cost US$ 30 each.

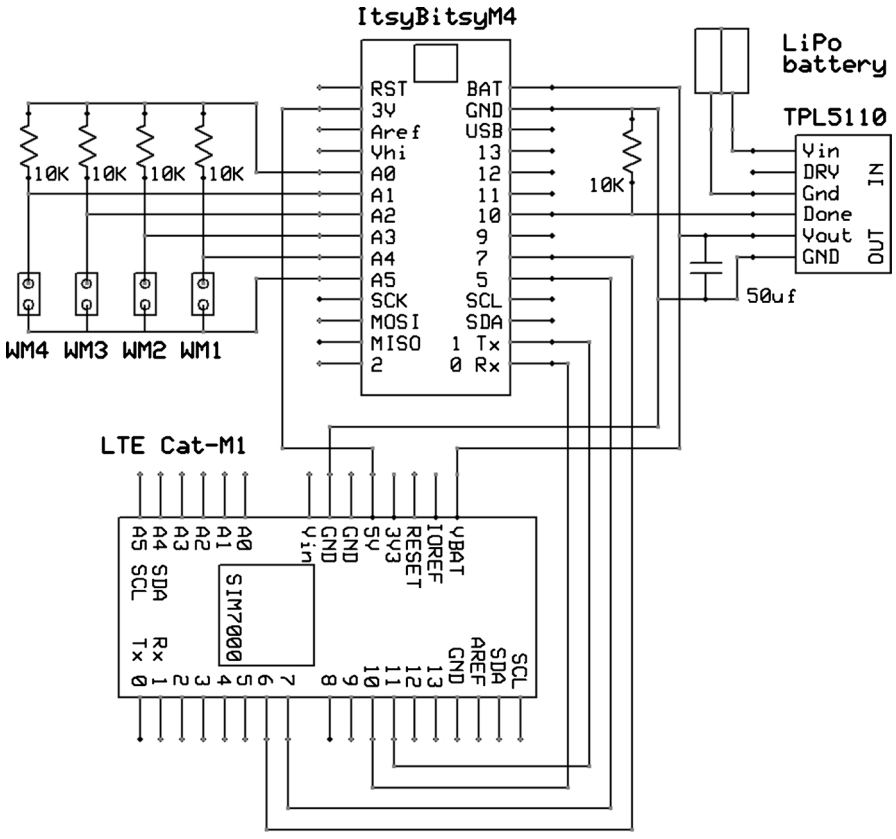

Figure 6. Electrical schematic for cellular soil-moisture monitor circuit.

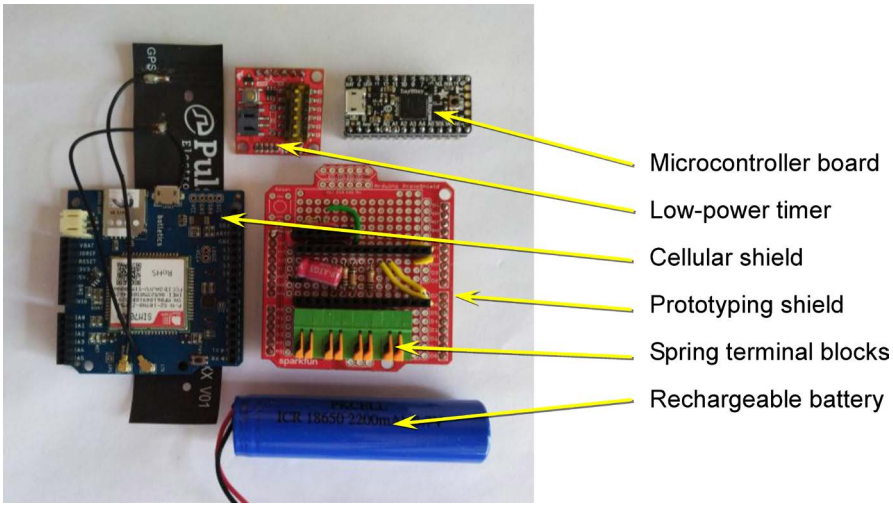

Figure 7. Soil-moisture sensor monitoring system hardware components.

The soil-moisture monitoring system was programmed in CircuitPython and since the system uses many of the same hardware components, it reuses many of the same software routines written for the Cellular Weather Station. The program begins by importing libraries needed to set up serial communications with the cellular module, specifying pins to interconnect the microcontroller, cellular shield, TPL5110 and soil-moisture sensors, and configuring and starting serial communications.

Each of the four soil-moisture sensors is read in turn by providing an excitation voltage to the network. Voltage is applied to the sensor's half-bridge network with one polarity (high voltage at one end, low voltage/ground at the other end) and the center voltage is measured with the microcontroller's analog-to-digital converter. The polarity is then switched, and the center voltage measured again. The alternating excitation is applied five times to provide a stable response and to avoid polarization of and damage to the sensor. The half-bridge center voltage is used to calculate the resistance of the sensor, and that resistance is input to a calibration equation to estimate soil-moisture status in units of matric potential, kPa.

Following reading of the four sensors, data are transmitted to the ThingSpeak internet data-hosting website as described for the Cellular Weather Station. The TPL5110 low-power timer is then signaled and battery power is turned off to the entire circuit for a period of 2 hrs, after which the circuit is powered on and the sensor measurement process is repeated. A sample website showing data from a soil-moisture monitoring system, installed in an agricultural field with soil-moisture sensors at four depths below the soil surface, is shown in **Figure 8**.

4. Conclusion

Open-source hardware and software have been used by the research and scientific communities to develop sensing and monitoring systems, often based on the Arduino microcontroller project. The original hardware and software continue

Figure 8. Time series of sensor data collected with the cellular soil-moisture monitor.

to evolve and become more powerful, with upgrades to both the microcontrollers and programming features and support. The Arduino programming environment, based on C/C++ computer language allows for development of powerful programs and devices, but a fair amount and expertise are required by the user. The open-source Python language has gained many users and a version derived for programming microcontrollers, called CircuitPython, has been developed. CircuitPython retains much of the core Python functions and programming options and offers features and advantages for users to interact with hardware and develop sensing and monitoring systems. Numerous microcontroller development boards, sensors, and peripheral devices are available that are supported by CircuitPython programming. Three sensing and monitoring systems were described and discussed to examine the usefulness of CircuitPython for research and scientific applications. Users, especially those with limited electronics or programming experience, can take advantage of CircuitPython's ease of use and extensive features to develop unique applications for data collection and

monitoring and enhance hardware and programming skills.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Pearce, J.M. (2012) Building Research Equipment with Free, Open-Source Hardware. *Science*, **337**, 1303-1304. https://doi.org/10.1126/science.1228183

[2] Pearce, J.M. (2012) The Case for Open Source Appropriate Technology. *Environment, Development and Sustainability*, **14**, 425-431. https://doi.org/10.1007/s10668-012-9337-9

[3] Fisher, D.K. and Gould, P.J. (2012) Open-Source Hardware is a Low-Cost Alternative for Scientific Instrumentation and Research. *Modern Instrumentation*, **1**, 8-20. https://doi.org/10.4236/mi.2012.12002

[4] Mangundu, E.M., Mateus, J.N., Zodi, G.A.L. and Johson, J. (2017) A Wireless Sensor Network for Rainfall Monitoring, Using Cellular Network: A Case for Namibia. 2017 *Global Wireless Summit*, Cape Town, 15-18 October 2017, 240-244. https://doi.org/10.1109/GWS.2017.8300469

[5] Ray, P.P. (2017) Internet of Things for Smart Agriculture: Technologies, Practices and Future Direction. *Journal of Ambient Intelligence and Smart Environments*, **9**, 395-420. https://doi.org/10.3233/AIS-170440

[6] Fisher, D.K., Woodruff, L.K., Anapalli, S.S. and Pinnamaneni, S.R. (2018) Open-Source Wireless Cloud-Connected Agricultural Sensor Network. *Journal of Sensor and Actuator Networks*, **7**, 47. https://doi.org/10.3390/jsan7040047

[7] Rodriguez-Juarez, P., Junez-Ferreira, H.E., Trinidad, J.G., Zavala, M., Burnes-Rudecino, S. and Bautista-Capetillo, C. (2018) Automated Laboratory Infiltrometer to Estimate Saturated Hydraulic Conductivity using an Arduino Microcontroller Board. *Water*, **10**, 1867. https://doi.org/10.3390/w10121867

[8] Fletcher, R.S. and Fisher, D.K. (2019) Spatial Analysis of Soybean Plant Height and Plant Canopy Temperature Measured with On-the-Go Tractor Mounted Sensors. *Agricultural Sciences*, **10**, 1486-1496. https://doi.org/10.4236/as.2019.1011109

[9] Fisher, D.K. and Sui, R. (2013) An Inexpensive Open-Source Ultrasonic Sensing System for Monitoring Liquid Levels. *Agricultural Engineering International: CIGR Journal*, **15**, 328-334.

[10] Payero, J.O., Mirzakhani-Nafchi, A., Khalilian, A., Qiao, X. and Davis, R. (2017) Development of a Low-Cost Internet-of-Things (IoT) System for Monitoring Soil Water Potential Using Watermark 200SS Sensors. *Advances in Internet of Things*, **7**, 71-86. https://doi.org/10.4236/ait.2017.73005

[11] Cao-Hoang, T., Tinh, T., Van, P. and Nguyen Duy, C. (2017) Design of a Cost Effective Soil Monitoring System to Support Agricultural Activities for Smallholder. *Journal of Information Communication Technology and Digital Convergence*, **2**, 1-5.

[12] Spinelli, G.M. and Gottesman, Z.L. (2019) A Low-Cost Arduino-Based Datalogger with Cellular Modem and FTP Communication for Irrigation Water Use Monitoring to Enable Access to Crop Manage. *Hardware X*, **6**, e00066. https://doi.org/10.1016/j.ohx.2019.e00066

[13] Li, Q.Q. and Wu, T. (2013) Research and Design of Small Humanoid Robot Based

on the Arduino. *Applied Mechanics and Materials*, **431**, 258-261. https://doi.org/10.4028/www.scientific.net/AMM.431.258

[14] Araújo, A., Portugal, D., Couceiro, M.S. and Rocha, R.P. (2015) Integrating Arduino-Based Educational Mobile Robots in ROS. *Journal of Intelligent & Robotic Systems*, **77**, 281-298. https://doi.org/10.1007/s10846-013-0007-4

[15] Ariawan, K.U., Santyadiputra, G.S. and Sutaya, I.W. (2019) Design of Hexapod Robot Movement Based on Arduino Mega 2560. *The* 1*st International Conference on Vocational Education and Technology*, Bali, 3 November 2018, Article ID: 012011. https://doi.org/10.1088/1742-6596/1165/1/012011

[16] Vidal-Pardo, A. and Pindado, S. (2018) Design and Development of a 5-Channel Arduino-Based Data Acquisition System (ABDAS) for Experimental Aerodynamics Research. *Sensors*, **18**, 2382. https://doi.org/10.3390/s18072382

[17] Busquets, J., Busquets, J.V., Tudela, D., Pérez, F., Busquets-Carbonell, J., Barberá, A., Rodríguez, C., García, A.J. and Gilabert, J. (2012) Low-Cost AUV Based on Arduino Open Source Microcontroller Board for Oceanographic Research Applications in a Collaborative Long Term Deployment Missions and Suitable for Combining with an USV as Autonomous Automatic Recharging Platform. 2012 *IEEE/OES Autonomous Underwater Vehicles*, Southampton, 24-27 September 2012, 1-10. https://doi.org/10.1109/AUV.2012.6380720

[18] Rivas-Sanchez, Y.A., Moreno-Perez, M.F. and Roldan-Canas. J. (2019) Environment Control with Low-Cost Microcontrollers and Microprocessors: Application for Green Walls. *Sustainability*, **11**, 782. https://doi.org/10.3390/su11030782

[19] Bridge, E.S., Wilhelm, J., Pandit, M.M., Moreno, A., Curry, C.M., Pearson, T.D., Proppe, D.S., Holwerda, C., Eadie, J.M., Stair, T.F., Olson, A.C., Lyon, B.E., Branch, C.L., Pitera, A.M., Kozlovsky, D., Sonnenberg, B.R., Pravosudov, V.V. and Ruyle, J.E. (2019) An Arduino-Based RFID Platform for Animal Research. *Frontiers in Ecology and Evolution*, **7**, 257. https://doi.org/10.3389/fevo.2019.00257

[20] Teikari, P., Najjar, R.P., Malkki, H., Knoblauch, K., Dumortier, D., Gronfier, C. and Cooper, H.M. (2012) An Inexpensive Arduino-Based LED Stimulator System for Vision Research. *Journal of Neuroscience Methods*, **211**, 227-236. https://doi.org/10.1016/j.jneumeth.2012.09.012

[21] D'Ausilio, A. (2012) Arduino: A Low-Cost Multipurpose Lab Equipment. *Behavioral Research Methods*, **44**, 305-313. https://doi.org/10.3758/s13428-011-0163-z

[22] Grinias, J.P., Whitfield, J.T., Guetschow, E.D. and Kennedy, R.T. (2016) An Inexpensive, Open-Source USB Arduino Data Acquisition Device for Chemical Instrumentation. *Journal of Chemical Education*, **93**, 1316-1319. https://doi.org/10.1021/acs.jchemed.6b00262

[23] Wickert, A.D., Sandell, C.T., Schulz, B. and Ng, G.H.C. (2018) Open-Source Arduino-Derived Data Loggers Designed for Field Research. *Hydrology and Earth Systems Sciences Discussions*. https://doi.org/10.5194/hess-2018-591

[24] Fisher, D.K., Fletcher, R.S., Anapalli, S.S. and Pringle III, H.C. (2018) Development of an Open-Source Cloud-Connected Sensor-Monitoring Platform. *Advances in Internet of Things*, **8**, 1-11. https://doi.org/10.4236/ait.2018.81001

[25] Fisher, D.K., Fletcher, R.S. and Anapalli, S.S. (2020) Evolving Open-Source Technologies Offer Options for Remote Sensing and Monitoring in Agriculture. *Advances in Internet of Things*, **10**, 1-10. https://doi.org/10.4236/ait.2020.101001