Smart System for Automated Irrigation Using Internet of Things Devices

Rhuanito Soranz Ferrarezi^{1,2} and Tzu Wei Peng¹

ADDITIONAL INDEX WORDS. embedded microcomputer, low-cost, management, open-source, water

SUMMARY. Precision agriculture involves applying artificial intelligence, computers, sensors, and automation to improve crop field productivity while monitoring environmental conditions to conserve soil, water, and other natural resources focusing on agricultural sustainability. Despite many applications in agriculture, data monitoring and recording technologies have limited use due to the price. Low-cost open-source systems, like the ones available with the Internet of things (IoT) world, can potentially be developed as a universal-fit and cloud-connected technology for multiple applications. We designed and built a basic data collecting system using a commercial standalone embedded computer with Python programming language, serial data interface (SDI)-12/analog sensor adaptor, and digital sensors to monitor soil moisture and transmit the data remotely. SDI-12 is a standard communication protocol that transfers digital sensor measurements to a data recorder. We set up a pilot study that automatically collected and uploaded the data into the Internet to allow remote data transfer and access. The system performed reliably over 1 week with potting soil under field conditions without maintenance and successfully recorded data in real-time. The volumetric water content ranged from 0.03 to 0.23 $\text{m}^3 \cdot \text{m}^{-3}$, dielectric permittivity from 3.3 to 18.9 (unitless), EC from 0.0 to 0.3 $\text{dS} \cdot \text{m}^{-1}$, and soil temperature from 20.7 to 44.8 °C. All the data were successfully collected and uploaded to the cloud every 20 min, allowing users to remotely monitor the data using a free online application. However, heavy rainfall and high insolation could damage the system through excessive moisture or overheating, requiring a waterproof and heavy-duty protection case. The ThingSpeak channel allows customizing to suit a user's specific requirements or adding more features for further development, such as automated irrigation, which can improve irrigation and fertilization efficiency by applying water and fertilizers at the right time based on sensor readings.

The United Nations predicts that the world's population will reach 9.7 billion by 2050 (United Nations Department of Economic and Social Affairs, 2015), which will require global food production to rise rapidly to meet consumer demand. Understanding the future environmental challenges of global crop production and how to achieve greater yields with fewer inputs require quantitative assessments of environmental variables such as soil moisture and water availability (Tilman et al., 2011). The efficient use of irrigation water in agricultural production is vital to farming operations' economic and environmental sustainability. Precision irrigation technologies allow farmers to apply water when, where, and in the amount needed to maximize profits while protecting the environment (Payero et al., 2017). According to a study with large-scale commercial corn (*Zea mays*) in Nebraska, irrigation scheduling based on sensors reduced water application rates by 33%. It

also reduced pumping cost by \$28.5/acre per year, with no significant reduction in yield compared with the farmer-managed fields (Irmak et al., 2012). With the advance of Internet connectivity and the popularization of technology adoption in agriculture, new tools are becoming available to improve the decision process in farm operations.

Internet of things (IoT) describes the network of devices or "things" embedded with sensors, software, and other technologies to smartly connect and exchange data between equipment and systems over the Internet. In smart farming, automated systems

monitor horticultural crops with sensors that measure environmental parameters such as solar radiation or light intensity, relative humidity, air temperature, soil moisture, electric conductivity (EC), wind speed, and direction, and control field operations (Ravindra, 2020). IoT-based smart farming allows users to automatically collect real-time information remotely as long as an Internet connection is available. Data are stored in cloud computing, which can assist users in making more intelligent decisions. IoT systems are expected to increase the operation efficiency compared with conventional agriculture due to the potential to lower dependence on frequent labor for decision processes. IoT devices are becoming more popular since consumers connect and integrate more equipment online, making this a dynamic and evolving market suitable for different operations.

With the introduction of industrial IoT in agriculture, more advanced sensors are being used. Sensors are either analog or digital. An analog sensor measures the external parameters and returns analog voltage as an output. The output voltage is usually in the range of 0 to 5 V direct current (DC). Unlike analog sensors, digital sensors produce discrete values (0 and 1); these values are often called digital (binary) signals in digital communication. Analog signals are highly affected by external factors such as noise and signal degradation, creating errors in the output signal and affecting data integrity. Noise is an unwanted electrical or magnetic phenomenon that corrupts a signal from the sensor. Digital signals are less susceptible to noisy environments, and hence digital sensors are usually preferred by users over analog ones. Digital sensors use a standard communication protocol known as serial/digital interface at 1200 baud [serial data interface (SDI)-12] that quickly allows user interfacing data recorders with microprocessor-based sensors. The

Units

To convert U.S. to SI,
multiply by

U.S. unit

SI unit

To convert SI to U.S.,
multiply by

0.4047

acre(s)

ha

2.4711

0.0283

ft³

m³

35.3147

2.54

inch(es)

cm

0.3937

1

µmho/cm

µS·cm⁻¹

1

1

mmho/cm

dS·m⁻¹

1

(°F - 32) ÷ 1.8

°F

°C

(°C × 1.8) + 32

“intelligent” sensor typically takes a measurement, makes computations based on the raw sensor reading, and outputs the measured data in meaningful units that are easy to understand. These multiparameter digital sensors help battery-powered controllers with minimal current drain, allowing several sensors to connect to a single datalogger. As a result, they can be wired to the same control port, and less processing power is required from the datalogger as complex algorithms are executed by the sensor. In addition, SDI-12 sensors can measure several soil parameters such as volumetric water content (VWC), dielectric permittivity (ϵ), temperature, and EC. Although various data logging systems and soil moisture sensors are commercially available, the large-scale application of these technologies to make irrigation scheduling decisions among farmers is still limited.

Factors affecting the limited adoption of these technologies include the lack of knowledge about the system applications, little information in system design and operation, high cost, difficulty installing and maintaining equipment, and problems transmitting data in real-time from sensors in the

Fig. 1. Essential datalogger components to monitor sensors and control devices: embedded computer with a credit-card size and dual-display desktop [Raspberry Pi 4 Model B with 8 Gb of random-access memory (RAM); Raspberry Pi Foundation, Cambridge, United Kingdom] and serial data interface (SDI)-12/analog adaptor with larger terminals [SDI-12 universal serial bus (USB) adaptor; Liudr's Blog, St. Cloud, MN] connected via the USB type-C port without the need of stacking or soldering the boards; DC = direct current.

field to the farmer (Payero et al., 2017). Most of these elements are centered on the microcontroller or datalogger used to process the data, as they are usually proprietary systems. The most common devices cost more than \$1500 and are generally used in scientific research settings. The programming is relatively simple but requires proprietary software. One potential solution is using low-cost open-source platforms such as the Arduino (Arduino, Ivrea, Italy) and Arduino-like boards (Ferrarezi et al., 2015). These prototyping boards are prevalent for inexpensive projects as parts are readily available, and the cost is appealing. For example, Ferrarezi et al. (2015) described an automated system for monitoring soil moisture and controlling irrigation using low-cost open-source microcontrollers. Open-source commonly refers to software that uses an open development process and is licensed to include the source code, which means the source code is freely available to the public. Anyone can modify the code to customize the system based on user needs. The program is written using the Arduino integrated development environment (IDE) free-ware that employs C/C++ programming language (Ferrarezi et al., 2015). However, the Arduino boards are made for small-scale prototyping projects. The addition of multiple accessory boards and algorithms to precisely control the timestamp, save data into a memory card, and transmit the data to the cloud reduces the system performance for

various sensors to handle large, complex systems.

Embedded computers with a credit-card size became popular due to the promotion of teaching introductory computer science. They are used in agriculture to monitor environmental parameters because of their low cost, modularity, and open-source design, and the system is more robust due to the higher processing capacity. There are different brands and models available in the market. To make smart farming more publicly accessible, a reliable, robust, low-cost, open-source IoT device can be developed as a universal-fit and cloud-connected data-collecting system integrated by computer programming.

Our objective is to describe a low-cost open-source IoT intelligent system using digital soil moisture sensors connected to an embedded computer to make moisture monitoring and irrigation control more efficient and affordable in specialty crop applications. This material was presented at the American Society of Horticultural Sciences 2020 Conference Workshop “Using Sensors to Inform and Control Irrigation Research and Management” as part of the Water Utilization and Management Professional Interest Group.

Pilot study: A low-cost open-source IoT system for irrigation control

LOGIC. The intelligent, automated irrigation platform described in

Received for publication 1 Apr. 2021. Accepted for publication 2 Aug. 2021.

Published online 27 October 2021.

¹Horticultural Sciences Department, Indian River Research and Education Center, University of Florida, Fort Pierce, FL 34945

²Department of Horticulture, University of Georgia, Athens, GA 30602

This paper is based on information presented during the “Using Sensors to Inform and Control Irrigation Research and Management” workshop at the ASHS 2020 Virtual Conference as part of the Water Utilization and Management (WUM) Professional Interest Group (PIG).

Funding for this research was provided by the U.S. Department of Agriculture's (USDA) National Institute of Food and Agriculture (NIFA) Hatch/Multi-state W4128 Project #1021596 and USDA-NIFA Emergency Citrus Disease Research and Extension Program (ECDRE) project #2018-70016-27387.

We thank the ASHS WUM PIG for the exciting presentations at the workshop, University of Florida's Institute of Food and Agricultural Sciences (UF/IFAS) Indian River Research and Education Center, and the HortTechnology editorial staff.

Mention of trade names or commercial products in this publication is solely for the purpose of providing specific information and does not imply recommendation or endorsement by the authors.

R.S.F. is the corresponding author. E-mail: ferrarezi@uga.edu.

This is an open access article distributed under the CC BY-NC-ND license (<https://creativecommons.org/licenses/by-nc-nd/4.0/>).

<https://doi.org/10.21273/HORTTECH04860-21>

Table 1. Information of the sensors used in the development of automated irrigation systems. The soil-specific calibrations for mineral soils ranging from 0 to <5 dS·m⁻¹ were obtained from the sensors' user manuals and applied to the code for the datalogger's program code to output specific data accordingly.

Manufacturer and sensor model	Measuring technique ^z	Response variables ^z	Factory calibration ^z	
			VWC	ε
Meter Group, Pullman, WA				
10HS	FDR	Voltage	$2.97 \times 10^{-9} mV^3 - 7.37 \times 10^{-6} mV^2 + 6.69 \times 10^{-3} mV - 1.92$	$2.589 \times 10^{-10} mV^4 - 5.010 \times 10^{-7} mV^3 + 3.523 \times 10^{-4} mV^2 - 9.135 \times 10^{-2} mV + 7.457$
GS3	FDR	ε, EC _b , T	$5.89 \times 10^{-6} \epsilon_a^3 - 7.62 \times 10^{-4} \epsilon_a^2 + 3.67 \times 10^{-2} \epsilon_a - 5.3 \times 10^{-2}$	
Acclima, Meridian, ID				
ACC-SEN-SDI	TDT	ε, EC _b , T	$4.3 \times 10^{-6} \epsilon_a^3 - 5.5 \times 10^{-4} \epsilon_a^2 \div 2.92 \times 10^{-2} \epsilon_a - 5.3 \times 10^{-2}$	

^zVWC = volumetric water content, ε = dielectric permittivity, FDR = frequency domain reflectometry, EC_b = bulk EC, T = temperature, TDT = time-domain transmission.

this paper integrates open-source software and hardware, Internet cloud components to allow remote data transfer and access, soil moisture sensors, and an irrigation manifold with a solenoid valve. A program was written and uploaded to a programmable single-board computer to perform data collection and transmission functions to an Internet cloud-based service.

HARDWARE. The low-cost open-source automated irrigation control system consisted of three major components: a datalogger, soil moisture sensors, and an irrigation manifold. Three soil moisture sensors were wired together and transmitted the data to the datalogger. The system stored the information locally in a microsecure digital (SD) card, uploaded it in real-time to the cloud wirelessly, and controlled the irrigation based on soil moisture data. This last portion was not tested in this pilot trial as it had been validated before (Ferrarezi et al., 2015; Nemali and van Iersel, 2006). In addition, data were also available remotely through a website that can be fully customized on the user backend.

DATALOGGER. We used an embedded computer with a credit-card size and dual-display desktop [Raspberry Pi 4 Model B with 8 GB of random-access memory (RAM); Raspberry Pi Foundation, Cambridge, UK] and an SDI-12/analog adaptor with larger terminals [SDI-12 universal serial bus (USB) adaptor; Liudr's Blog, St. Cloud, MN] to read both SDI-12 and analog sensors (Fig. 1). More info about the board and compatible soil moisture sensors are available at Liu (2021). The system was housed in a 20 × 12 × 5.6-cm waterproof and dustproof enclosure (LeMotech ABS,

Shenzhen, China). The desktop microcomputer is an open-source single-board embedded computer used as both a microcontroller and micro-processor. The model used features Gigabit Ethernet, onboard wireless networking, Bluetooth, four USB ports, a micro-SD card slot for data storage, and 40 physical general-purpose input/output (GPIO) pins to interact with different components. The device consumes low power and requires a standard USB Type-C (USB-C) 5-V 3-A

DC power supply and has an internal 5-V battery to power the SDI-12/analog adaptor via USB port connection without the need of stacking or soldering the boards.

SENSORS. We used three sensors: an analog (10HS; Meter Group, Pullman, WA) and two SDI-12 digital sensors [GS3 (Meter Group) and ACC-SEN-SDI (Acclima, Meridian, ID)]. The 10HS analog sensor was assigned to address z, GS3 digital sensor to address 1, and ACC-SEN-SDI digital

Fig. 2. Basic wiring diagram of automated irrigation controller with embedded computer with a credit-card size and dual-display desktop [Raspberry Pi 4 Model B with 8 Gb of random-access memory (RAM); Raspberry Pi Foundation, Cambridge, United Kingdom], relay module [one channel 5-V direct current (DC); SaintSmart, Las Vegas, NV], and 1-inch (2.5 cm) 24-V alternate current (AC) solenoid valve (Orbit, North Salt Lake, UT) (Raspberry Pi Foundation, 2021); GPIO = general purpose input/output.

Total number of data points: 300
 Delay between data points (seconds): 600
 Time stamps are generated with:
 0) GMT/UTC
 1) Local
 Select time zone: 1
 Enter all SDI-12 sensor addresses, such as 1234: 12
 Collect analog inputs (requires SDI-12 USB + Analog adaptor)? (Y?N): Y
 Answers

Sensor address: 1 Sensor info: 113DECAGON GS3 402
 Sensor address: 2 Sensor info: 214ACCLIMA TDT 0.76013451

Fig. 3. Example of the computer code used to run the automated system. Items with an orange box are direct outputs of the computer instruction; GMT/UTC = Greenwich mean time and coordinated universal time, SDI = serial data interface, USB = universal serial bus.

sensor to address 2. The 10HS analog sensor uses frequency domain reflectometry (FDR) for the measurements. The principle is based on the electrical capacitance that uses the soil as a dielectric pole to measure the soil VWC. When connecting the capacitor with an oscillator to form an electrical circuit, changes in soil moisture can be detected by changes in the operating frequency (Ferrarezi et al., 2020). The analog sensor determines voltage, which is manually converted into VWC and ϵ by using the sensor manufacturer's calibration equations. The calibration was added to the datalogger to calculate the values automatically (Table 1).

The GS3 digital sensor also uses the same FDR technology by sending out a 70-MHz frequency pulse to the soil. The difference between the output wave and the return wave frequency is measured to determine the soil moisture. In addition, the GS3 sensor measures ϵ , bulk EC, and temperature.

The ACC-SEN-SDI digital sensor uses a different operation principle known as time-domain transmission (TDT), which measures the speed from the start to the end of the loop to propagate an electrical pulse. A pulse emitted by the ACC-SEN-SDI sensor is slower in wetter soils than drier soils. This sensor measures ϵ , EC, and temperature. Both sensors, the GS3 and ACC-SEN-SDI, do not require previous conversions since they output all the data as a string that quickly allows users to obtain the data microprocessed by the sensors. ϵ , EC, and temperature

are generated directly using a generic equation built into the sensors.

Soil moisture sensors were wired to the SDI-12/analog adaptor. The 10HS sensor was powered by the internal 5-V DC power supply, and an external 12-V DC power supply was used to power the ACC-SEN-SDI and GS3 sensors as they require more than 5 V to function. Analog sensors were assigned to address z, and SDI-12 sensors to addresses starting from number 1 on the SDI-12/analog adaptor. The data collected from the sensors were stored in the desktop microcomputer as comma-separated values (CSV) file for local backup and also uploaded to the cloud for real-time monitoring.

Fig. 4. Diagram of a smart system for automated irrigation using Internet of things devices: embedded computer with a credit-card size and dual-display desktop [Raspberry Pi 4 Model B with 8 Gb of random-access memory (RAM); Raspberry Pi Foundation, Cambridge, United Kingdom] and serial data interface (SDI)-12/analog adaptor with larger terminals [SDI-12 universal serial bus (USB) adaptor; Liudr's Blog, St. Cloud, MN] connected to three soil moisture sensors [an analog (10HS; Meter Group, Pullman, WA) and two SDI-12 digital sensors [GS3 (Meter Group) and ACC-SEN-SDI (Acclima, Meridian, ID)]]; DC = direct current.

IRRIGATION MANIFOLD. The ability of microcontrollers to successfully turn solenoid valves on and off has been previously described (Ferrarezi et al., 2015; Nemali and van Iersel, 2006). For that reason, we decided not to trigger the irrigation automatically in this study. Instead, we built a manifold consisting of one 5-V DC-operated relay module (SainSmart, Lenexa, KS) and a 24-V alternate current (AC) solenoid valve (Orbit, North Salt Lake, UT). The 5-V DC relay module was wired to the desktop microcomputer's GPIO pins #2, #6, and #23; GPIO pin #2 acts as the internal 5-V power supply to operate the relay, pin #6 is the ground, and pin #23 is the signal output to control the relay. The 5-V DC relay module was equipped with a high-current AC relay up to 30 V and 10 A, allowing to connect the 24-V AC external power supply to trigger the solenoid valve. The solenoid valve was not connected to a water source (Fig. 2).

SOFTWARE. The desktop microcomputer runs its unique operating system (Debian GNU/Linux version 10.8; Debian Project, San Francisco, CA) based on the Linux kernel, a free and open-source community-developed operating system. The program was built using the desktop microcomputer's built-in open-source software IDE (Thonny version 3.3.6; University

of Tartu, Tartu, Estonia), which is available for Python programming language to write codes and upload the compiled data program into the desktop microcomputer. Our automated system was designed to run the script with Python version 3.4 or higher. We used a built-in virtual networking computing software (RealVNC Viewer version 6.20.817; RealVNC, Cambridge, UK) to access the desktop microcomputer for remote control and troubleshooting. Additional instructions for setting up the desktop microcomputer and its software are accessible on the manufacturer's website. We used a cloud server to store the data online and allow real-time monitoring (ThingSpeak, 2020), which is an open-source IoT analytics platform that enables users to aggregate, visualize, and analyze live data streams in the cloud. Users have a unique channel identification and key code through the application programming interface provided by the website after creating an account. The website also provides instructions to assist in setting up and configuring the channel's webpage. Users can decide to set the streaming channel in public or private mode, and the webpage could be viewed at any time from any place via a web browser or an app on a computer or a mobile device.

CODE. The Python script running the system is exemplified in Fig. 3

and fully accessible as Supplemental Material. The code was modified from the open-source script available at Liu (2021).

EXPERIMENTAL SETUP. To test the power consumption of the system in a field where an outlet is not accessible, we used a 12-V 7-A DC rechargeable battery, solar panel, charge controller (CMP-12; Unique Bargains, San Francisco, CA), and a DC/DC 12-V to 5-V adaptor (Homree, Hong Kong, China). The solar panel was connected to the charge controller to regulate the incoming voltage, and the charge controller was wired to the 12-V DC rechargeable battery, which powered the entire system. Finally, the DC/DC adaptor was used to convert the 12-V output from the battery to 5 V to power the desktop microcomputer.

The desktop microcontroller, charge controller, and the DC/DC adaptor were installed in a waterproof enclosure, and the system was placed in an outdoor garden in central Florida. We used an all-natural potting mix for outdoor containers with pH ranging from 5.8 to 7.5 (All-Natural Potting Mix Premium for Outdoor Container; Kellogg Garden Organics, Carson, CA).

The system diagram is shown in Fig. 4, and the hardware components and the approximate cost are listed in Table 2. If users decide on using a regular 110-V AC outlet as a power

Table 2. List of materials and approximate costs for the automated irrigation system.

Main components	Part model ^z	Source	Cost
Desktop microcomputer or datalogger	Raspberry Pi 4 Model B	Raspberry Pi Foundation, Cambridge, UK	\$35.00
	Serial data interface (SDI)-12/analog adaptor	Liudr's Blog, St. Cloud, MN	\$79.90
Sensors (3)	10HS analog sensor	Meter Group, Pullman, WA	≈\$200.00 each (\$600.00)
	GS3 SDI-12 digital sensor	Meter Group	
	ACC-SEN-SDI SDI-12 digital sensor	Acclima, Meridian, ID	
Irrigation manifold	One channel 5-V direct current (DC) relay module	SaintSmart, Las Vegas, NV	\$3.70
	1-inch 24-V alternate current (AC) female to female solenoid valve	Orbit, Bountiful, UT	\$11.00
	Dual spray half-circle sprinkler	Rain Bird, Azusa, CA	\$11.40
Power supply	12-V 7-A DC rechargeable battery	Generic Power System, Waukesha, WI	\$30.00 ^y
	20-W solar panel	Generic Power System	\$25.00 ^y
	CMP-12 5-A solar charge controller	SaintSmart	\$18.00 ^y
	DC/DC 12-V to 5-V adaptor	SSLHONG, Shenzhen, China	\$14.00
Others	20 × 12 × 5.6-cm waterproof enclosure ^z	LeMotech ABS, Shenzhen, China	\$30.00
	SanDisk 32 GigaByte (GB) microsecure digital (SD) card	Western Digital, San Jose, CA	\$8.00
Total			\$866.00

^z1 inch = 2.54 cm, 1 cm = 0.3937 inch.

^yItems not needed if the system is powered using electricity.

Fig. 5. Snapshot of cloud server storing the data and allowing real-time data monitoring for on demand irrigation in June 2020 (ThingSpeak, 2020); $1 \text{ m}^3 \cdot \text{m}^{-3} = 100\%$, $(1.8 \times ^\circ\text{C}) + 32 = ^\circ\text{F}$, $1 \mu\text{S} \cdot \text{cm}^{-1} = 1 \mu\text{mho}/\text{cm}$, $1 \text{ dS} \cdot \text{m}^{-1} = 1 \text{ mmho}/\text{cm}$, $1 \text{ cm} = 0.3937 \text{ inch}$.

supply, the price for the rechargeable battery set up can be reduced. The cost of soil moisture sensors and the irrigation manifold can vary depending on the users' choice. For this experiment, a one-channel 5-V DC-operated relay was used. We

programmed the relay to be controlled based on the sensor readings. If the VWC values dropped below 20%, the desktop microcomputer output 5 V DC to turn the relay on for 10 s. When the relay is on, it opened the solenoid valve to simulate letting the

water flow through the system. At the same time, the system updated its irrigation status to the live streaming platform.

Results

Our goal was to search for a robust, low-cost open-source setup to read SDI-12 digital sensors and test if the system can run with the rechargeable solar power supply system for 1 week without maintenance.

SENSOR READINGS. The ϵ expresses a substance's ability to hold an electrical charge and is represented as a substance's permittivity ratio by the vacuum permittivity. The ϵ equals 1 in air and ≈ 80 in water (Ferrarezi et al., 2020), easily discriminating moisture in different soil types and soilless growing media. We tested how the sensor readings responded in water and air and an open outdoor setting, and the embedded system successfully recorded the sensor readings. The ϵ readings in water for 10HS analog sensor = 81.4, GS3 digital sensor = 76.0, and ACC-SEN-SDI digital sensor = 73.3. In air, the ϵ reading for 10HS = 2.34, GS3 = 1.42, and ACC-SEN-SDI = 1 (Table 3). VWC is the ratio of water volume to soil volume. The 10HS sensor's reading is about 65% in water, GS3 equals 90%, and ACC-SEN-SDI 89%. Typically, the VWC ranges from 0% to 100% or 0 to 1 m^3 of water per cubic meter of soil. However, the range for the 10HS sensor is from 0% to 69% in soilless media (Meter Group, 2021). The VWC reading of sensors in the air all resulted in 0. Some sensors outputted negative values, but that results from the calibration equation terms since there is no negative water volume in soils.

DATA COLLECTION AND MONITORING. The experiment was conducted from Mar. to June 2020. All data were successfully collected and uploaded to the cloud every 20 min, allowing users to monitor the data through the ThingSpeak channel remotely. Sample results of the sensors installed in potting soils in June 2020 on display at the ThingSpeak channel are shown in Fig. 5, highlighting the system responded as expected with the soil moisture conditions (dry-down periods followed by sharp increases after a rainfall event). ThingSpeak created a status window to show each sensor's

Table 3. Sensor voltage, dielectric permittivity (ϵ), and volumetric water content (VWC) readings in completely dry (air) and wet (water) conditions to test sensors' accuracy. The negative values of VWC obtained from sensors in the air represent the dehydrated condition, which could round up to zero.

Sensor	Address	Location	Voltage (mV)	ϵ (unitless)	Reading value	
					VWC	
					($\text{m}^3 \cdot \text{m}^{-3}$) ^z	(%)
10HS analog	z	In air	430.27	2.34	−0.169	−16.9
		In water	1213.2	81.4	0.652	65.2
GS3 serial data interface (SDI)-12 digital	1	In air	—	1.42	−0.025	−2.50
		In water	—	76.0	0.899	89.9
ACC-SEN-SDI SDI-12 digital	2	In air	—	1.00	0.000	0.00
		In water	—	73.3	0.893	89.3

^z1 $\text{m}^3 \cdot \text{m}^{-3}$ = 100%.

VWC reading and irrigation status every 20 min (Fig. 6). The status window is helpful for quickly checking if sensors are working correctly after adding water or moving the sensors for troubleshooting or other reasons. The data collected were analyzed from the CSV files stored in the desktop microcomputer. Fig. 7 shows the three sensors' graphic charts in June 2020 made from data stored locally in the desktop microcomputer. The peaks indicate the quick response of the sensors to an increase in soil moisture due to rainfall. The data shows the soil temperature readings from GS3 and ACC-SEN-SDI sensors were consistent, and the ACC-SEN-SDI sensor registered a higher EC value than the GS3 sensor. Sensor differences are expected since each one has a different measurement and processing technology. Several studies have been conducted to test other sensors in various soil types and conditions, indicating the need for soil-specific calibrations to increase accuracy

(Ferrarezi et al., 2020). Furthermore, the wireless communication between the desktop microcomputer and the ThingSpeak website proved reliable and never failed to report data to the cloud.

SYSTEM PERFORMANCE. We set up the system in a waterproof hard case and tested it outdoor for 3 months. The system experienced the weather variability of subtropical conditions in Florida without significant maintenance requirements. The only issue that occurred was the overheating of the desktop microcomputer due to direct exposure to sunlight, which caused the power consumption to go up. The increased power consumption drained the rechargeable battery faster than the solar panel charging speed, which resulted in the system running out of power in 3 d. After the first attempt, we moved the system to a shaded area to test the power supply performance. The system ran longer than the first test, but the solar panel's charging speed still

could not keep up with the entire system's power consumption. We found the system depleted power quickly at night when the solar panel could not be charged. That was caused by the frequent data collection and high power consumption of the desktop microcomputer to collect data and send it in real-time to the cloud. The data collection and solenoid valve activation ran smoothly after switching the rechargeable solar power supply to a stable 110-V power outlet.

Conclusion

In this study, a system was developed to periodically collect data using three commercial sensors installed in an outdoor garden with potting soil and automatically send the data to the cloud via Wi-Fi. The system was developed using open-source devices (desktop microcomputer and SDI-12/analog adaptor) with Python script and an IoT platform to live streaming the data (ThingSpeak, 2021). Before installing the system to collect data, laboratory tests were conducted to check if the microcontroller could respond to the sensor readings in air and water based on the manufacturers' calibrations. The system described in this study can be built for \$198 to \$266 without considering the sensors, and every component of the system is commercially available (Table 2). Heavy-duty and waterproof protection cases are required if the system is running outdoor. Potential future improvements for the system could include more power-efficient instructions to save battery. In addition, the system can become more field-friendly where a stable power outlet is not available or difficult to reach. Making the system more power

Fig. 6. Status window sample of cloud server platform to store the data and allow real-time volumetric water content (VWC) monitoring (ThingSpeak, 2020); TDT = time-domain transmission.

Fig. 7. Volumetric water content, dielectric permittivity, temperature, and electrical conductivity daily sample data extracted from the local comma-separated values (CSV) file; $1 \text{ m}^3 \cdot \text{m}^{-3} = 100\%$, $(1.8 \times ^\circ\text{C}) + 32 = ^\circ\text{F}$, $1 \text{ dS} \cdot \text{m}^{-1} = 1 \text{ mmho/cm}$, $1 \text{ cm} = 0.3937 \text{ inch}$.

efficient can be accomplished by programming the system only to turn on when data collection is required or using a microcontroller that uses less energy. The system's development and improvement can increase smart farming's popularity for scientists, home users, hobbyists, and small growers.

Literature cited

Ferrarezi, R.S., S.K. Dove, and M.W. van Iersel. 2015. An automated system for monitoring soil moisture and controlling irrigation using low-cost open-source microcontrollers. *HortTechnology* 25(1):

110–118, <https://doi.org/10.21273/HORTTECH.25.1.110>.

Ferrarezi, R.S., T.A.R. Nogueira, and S.G.C. Zepeda. 2020. Performance of soil moisture sensors in Florida sandy soils. *Water* 12(358):1–20, <https://doi.org/10.3390/w12020358>.

Irmak, S., M.J. Burgert, H.S. Yang, K.G. Cassman, D.T. Walters, W.R. Rathje, J.O. Payero, P. Grassini, M.S. Kuzila, K.J. Brunkhorst, D.E. Eisenhauer, W.L. Kranz, W.B. Van, J.M. Rees, G.L. Zoubek, C.A. Shapiro, and G.J. Teichmeier. 2012. Large-scale on-farm implementation of soil moisture-based irrigation management strategies for increasing maize

water productivity. *Trans. ASABE* 55(3): 881–894, <https://doi.org/10.13031/2013.41521>.

Liu, J. 2021. SDI-12 USB adapter. 21 June 2021. <<https://liudr.wordpress.com/gadget/sdi-12-usb-adapter>>.

Meter Group. 2021. 10HS manual. 20 June 2021. <http://publications.metergroup.com/Manuals/20426_10HS_Manual_Web.pdf>.

Nemali, K.S. and M.W. van Iersel. 2006. An automated system for controlling drought stress and irrigation in potted plants. *Scientia Hort.* 110(3):292–297, <https://doi.org/10.1016/j.scienta.2006.07.009>.

Payero, J.O., A. Mirzakhani-Nafchi, A. Khalilian, X. Qiao, and R. Davis. 2017. Development of a low-cost Internet-of-Things (IoT) system for monitoring soil water potential using Watermark 200SS Sensors. *Adv. Internet of Things* 7(3):71–86, <https://doi.org/10.4236/ait.2017.73005>.

Raspberry Pi Foundation. 2021. GPIO. 29 July 2021. <<https://www.raspberrypi.org/documentation/usage/gpio/>>.

Ravindra, S. 2020. IoT applications in agriculture. 3 Mar. 2021. <<https://www.iotforall.com/iot-applications-in-agriculture>>.

ThingSpeak. 2020. ThingSpeak for IoT projects. 20 July 2020. <<https://thingspeak.com>>.

ThingSpeak. 2021. ThingSpeak for smart farming. 20 Mar. 2021. <http://54.88.253.164/pages/smart_farming>.

Tilman, D., C. Balzer, J. Hill, and B. Befort. 2011. Global food demand and the sustainable intensification of agriculture. *Proc. Natl. Acad. Sci. USA* 108(50): 20260–20264, <https://doi.org/10.1073/pnas.1116437108>.

United Nations Department of Economic and Social Affairs. 2015. World population projected to reach 9.7 billion by 2050. 20 Mar. 2021. <<https://www.un.org/en/development/desa/news/population/2015-report.html>>.

Supplemental material

Computer code used to automate the sensor measurement, irrigation control, and data uploading to an online Cloud computing service. Text in black represents the actual code, and text in blue/italics after # sign denotes explanation.

```
#!/usr/local/opt/python-3.5.1/bin/python3.5
# Serial data interface (SDI)-12 sensor
data logger copyright Dr. John Liu 2017-
11-06, modified by Rhuanito Ferrarezi &
Tzu-Wei Peng
```

```
import os # For running command line
commands
import time # For delaying in seconds
import datetime # For finding the system's
real-time
import serial.tools.list_ports # For listing
available serial ports
import serial # For serial communication
support
import re # For regular expression
system
import urllib.parse # For encoding data to
be uniform resource locator (URL) safe
import signal # For trapping Ctrl+c or
interrupt signal (SIGINT)
import sys # For exiting the program with
exit code
import RPi.GPIO as GPIO # For control-
ling relays
import math # For sensor calibration
```

```
GPIO.setmode(GPIO.BOARD) # General
purpose Input/Output (GPIO) output for
relays
```

```
GPIO.setup(23,GPIO.OUT)
GPIO.output(23,GPIO.LOW)
time.sleep(3)
```

```
def SIGINT_handler(signal, frame):
    ser.close()
    data_file.close()
    GPIO.output(23,GPIO.LOW)
    print('Quitting program!')
    sys.exit(0)
signal.signal(signal.SIGINT, SIGINT_handler)
```

```
channelID = "1042887" # User Think-
Speak Channel Identification
api_key = "4VGNVVW32D5JMWBG"
# User ThingSpeak channel write applica-
tion programming interface (API) key
curl_command_format='curl "https://api.
thingspeak.com/update/?api_key=%s%s%s"'
# This is the client URL (cURL) upload
command to thingspeak.com
RasPi_A=platform.node() # Use computer
name as unit_id. For a Raspberry Pi,
change its name from raspberrypi to some-
thing else to avoid confusion
```

```
curl_exists=False # The code will test
whether cURL exists. If yes, it will be
used to upload data
adapter_sdi_12_address='z'
no_data=False # This is the flag to break
out of the inner loops and continue the
next data point loop in case no data is
received from a sensor such as the Global
Positioning System (GPS)
```

```
if (os.system('curl -V')==0):
    curl_exists=True
```

```
print('+-*40')
print('SDI-12 Sensor and Analog Sensor
Python Data Logger with Telemetry V1.5.0')
print('Designed for Dr. Liu's family of
SDI-12 USB adapters (standard,ana-
log,GPS)\nDr. John Liu Saint Cloud MN
USA 2017-11-06\nFree software GNU
GPL V3.0\nModified by Rhuanito Fer-
rarezzi and Tzu-Wei Peng')
print("\nCompatible with Windows, GNU/
Linux, Mac OSX, and Raspberry PI")
print("\nThis program requires Python 3.4,
Pyserial 3.0, and cURL (data upload)")
print("\nData is logged to YYYYMMDD.CVS
in the Python code's folder")
print("\nVisit https://thingspeak.com/channels/
%s to inspect or retrieve data' %(channelID))
print ("\nFor assistance with customization,
telemetry etc., contact Dr. Liu.\nthttps://
liudr.wordpress.com/gadget/
sdi-12-usb-adapter/")
print('+-*40')
ports=[]
```

```
VID_FTDI=0x0403;
```

```
a=serial.tools.list_ports.comports()
for w in a:
    ports.append((w.vid,w.device))
```

```
ports.sort(key= lambda ports: ports[1])
```

```
print("\nDetected the following serial
ports:")
i=0
```

```
for w in ports:
    print("%d)\t%s\t(USB VID=%04X)'
    %(i, w[1], w[0] if (type(w[0]) is int)
    else 0))
    i=i+1
```

```
total_ports=i # Now i= total ports
```

```
user_port_selection=input("\nSelect port
from list (0,1,2...). SDI-12 adapter has
USB VID=0403:')
if (int(user_port_selection)>=total_ports):
    exit(1) # Port selection out of range
```

```
ser=serial.Serial(port=(ports[int(user_por-
t_selection)])[1],baudrate=9600,timeout=10)
time.sleep(2.5) # Delay for Arduino boot-
loader and the 1 second delay of the
adapter
```

```
total_data_count=int(input("Total number
of data points:"))
delay_between_pts=int(input("Delay between
data points (second):"))
```

```
print("Time stamps are generated with:\n0)
GMT/UTC\n1) Local\n")
time_zone_choice=int(input('Select
time zone:'))
if time_zone_choice==0:
```

```
    now=datetime.datetime.utcnow() # Use
coordinated universal time (UTC) time
instead of local time
```

```
elif time_zone_choice==1:
    now=datetime.datetime.now() # Use
local time, not recommended for multiple
data loggers in different time zones
```

```
data_file_name="%04d%02d%02d.csv"
%(now.year,now.month,now.day)
data_file = open(data_file_name, 'a')
# Open yyyyymmdd.csv for appending
```

```
sdi_12_address=""
user_sdi_12_address=input("Enter all SDI-
12 sensor addresses, such as 1234:')
user_sdi_12_address=user_sdi_12_address.-
strip() # Remove any \r from an input file typed
in windows
```

```
analog_inputs=input('Collect analog inputs
(requires SDI12-USB + Analog adapter)?
(Y/N)')
analog_inputs=(analog_inputs.strip()).capi-
talize() # Remove any \r from an input file
typed in windows and capitalize answer
```

```
for an_address in user_sdi_12_address:
    if ((an_address>="0") and (an_address
    <="9")) or ((an_address>="A") and
    (an_address<="Z")) or ((an_address>="a")
    and (an_address<="z")):
```

```
        print("Using address:",an_address);
        sdi_12_address=sdi_12_address+an_ad-
        dress
```

```
    else:
        print('Invalid address:',an_address)
```

```
if analog_inputs=='Y':
    sdi_12_address=sdi_12_ad-
    dress+adapter_sdi_12_address
if len(sdi_12_address)==0:
    sdi_12_address=adapter_sdi_12_
    address # Use default address
```

```
for an_address in sdi_12_address:
    ser.write(an_address.encode()+b'I') #
Send identification
```

```
    sdi_12_line=ser.readline()
    print('Sensor address:',an_address,' Sen-
    sor info:',sdi_12_line.decode('utf-8').strip())
print('Saving to %s' %data_file_name)
print('='*40)
```

```
for j in range(total_data_count):
```

```

i=0 # This counts to 6 to truncate all
data to the 6 values set up in Sparkfun's
Phant server upload
value_str="" # This stores
&value0=xxx&value1=xxx&value2=xxx-
&value3=xxx&value4=xxx&value5=xxx
and is only reset after all sensors are read
if time_zone_choice==0:
    now=datetime.datetime.utcnow()
elif time_zone_choice==1:
    now=datetime.datetime.now()
output_str="%04d/%02d/%02d
%02d:%02d:%02d%s" %(now.year,now.-
month,now.day,now.hour,now.minute,-
now.second,' GMT' if
time_zone_choice==0 else ")
# Formatting date and time
for an_address in sdi_12_address:
    ser.write(an_address.encode()+b'M!');
# Start the SDI-12 sensor measurement
# print(an_address.encode()+
b'M!'); # Start the SDI-12 sensor
measurement
sdi_12_line=ser.readline()
# print(sdi_12_line)
sdi_12_line=sdi_12_line[:-2] # Remove
\r and \n since [0-9]$ has trouble with \r
m=re.search(b'[0-9]$',sdi_12_line) #
Having trouble with the \r
total_returned_values=int(m.
group(0)) # Find how many values are
returned
sdi_12_line=ser.readline()
# Read the service request line
ser.write(an_address.encode()+
b'D0!') # Request data
# print(an_address.encode()+
b'D0!') # Request data
sdi_12_line=ser.readline()
# Read the data line
# print(sdi_12_line)
sdi_12_line=sdi_12_line[1:-2]
# Remove address, \r and \n since [0-9]$
has trouble with \r
values=[] # Clear cache before each sensor
for iterator in range(total_returned_val-
ues): # Extract the returned values from
SDI-12 sensor and append to values[]
    m=re.search(b'[+-][0-9.]+' ,sdi_12_-
line) # Match a number string
    try: # If values found is less than val-
ues indicated by return from M, report no

```

```

data found. This is a simple solution to
GPS sensors before they acquire lock. For
sensors that have lots of values to return,
you need to find a better solution
values.append(float(m.group(0))) #
Convert into a number
sdi_12_line=sdi_12_line[len(m.
group(0)):]
except AttributeError:
    print("No data received from sensor at
address %c\n" %(an_address))
    time.sleep(delay_between_pts)
    no_data=True
    break
if (no_data==True):
    break;
output_str=output_str+'\n'+an_address
for value_i in values:
    output_str=output_str+",%s" %(val-
ue_i) # Output returned values depends on
your sensor choices and its calibration
    if (i==0): # GS3 dielectric value
        GS3_VWC=round(((0.118* ((val-
ue_i)**0.5))-0.117)*100,1) # GS3 sensor
volumetric water content (VWC) mineral
soil calibration
    if (i==3): # ACC-SEN-SDI sensor
VWC value in percent
        TDT_VWC=value_i
    if (i==7 and i<8): # 10HS analog
value
        mV=(value_i)*1000
        ANLG_VWC=round((((7.05*(10*-
*-7))*((mV)**2))-((1.23*(10**-4))*(mV))-
(3.03*(10**-2)))*100,1) # 10HS sensor
VWC mineral soil calibration
    if (i<8):
        value_str=value_str+"&field%d=
%s" %(i+1,value_i) # Format values for
posting. Field starts with field1
        i=i+1
if (no_data==True):
    no_data=False
    continue;
while (i<8): # Pad with zeros in case
we do not have six characters. This is
only necessary for certain servers
    value_str=value_str+"&field%d=0"
%(i+1) # Format values for posting. Field starts
with field1, not field0
    i=i+1

```

```

status_str='Current Status:'
if (GS3_VWC<20 or TDT_VWC
<20 or ANLG_VWC<20): # Put your own
desire irrigation status
    irrigation_status="Too Dry, Irrigation
Started! VWC values(percentage): Sen-
sor1(GS3): %s, Sensor2(TDT): %s, Sen-
sor3(10HS): %s" %(GS3_VWC, TDT_VWC,
ANLG_VWC)
    irrigation_counts=1
    GPIO.output(23,GPIO.HIGH)
    time.sleep(10)
    GPIO.output(23, GPIO.LOW)
else:
    irrigation_status='Do Not Need Irriga-
tion. VWC values: Sensor1(GS3): %s, Sen-
sor2(TDT): %s, Sensor3(10HS): %s'
%(GS3_VWC, TDT_VWC, ANLG_VWC)
    irrigation_counts=0
    GPIO.output(23, GPIO.LOW)
    status_str_curl="&status=Current
Status: %s" %(irrigation_status)
    status_str=status_str+"%s"
%(irrigation_status)
    irrigation_system=output_str+"\n %s"
%(irrigation_counts)

print(output_str)
print(status_str)
print(irrigation_counts)
output_str=output_str+'\n'
irrigation_system=irrigation_system-
+'\n'
data_file.write(irrigation_system)
if (curl_exists==True):
    curl_command=curl_command_
format %(api_key,value_str,status_str_curl)
# Format cURL command

print('='*40)

print(curl_command) # Debug information
print(os.system(curl_command)) # Send
data to ThinkSpeak.com using cURL

values=[] # Clear values for the next iter-
ation (ThinkSpeak 3.2.3 does not support
clear and 3.4.3 and 3.5.1 does)
data_file.flush() # Make sure data is writ-
ten to the disk so stopping the script with
ctrl+C will not cause data loss
time.sleep(delay_between_pts)
ser.close()
data_file.close()

```