Smart Agriculture Technology: Adapted for Varying Growing Conditions

Simon S. Stroh

College of Information Sciences and Technology, Penn State

Abstract

We created a software suite that serves the purpose of aggregating data across multiple farms by using web and mobile applications. The applications we have created have given farmers the ability to organize the data they gather by crop cycles and farm rooms, or sections.

By completing the software that was needed for recording information about farmers' crops, we were able to understand the system that farmers used for improving the yield of their crops. We built features in a mobile application that would allow farmers to move freely through each room or section of their farm while recording information about the crop's environment.

While this solution added the necessary technology for contributing information about crops to our application, we realized that other solutions were possible that could improve the efficiency of recording certain types of information. The pathway towards a technology-embedded indoors farm is always winding. The problems within agriculture are ever-increasingly complex and require solutions with many different pieces of technology working together.

Smart Agriculture Technology: Adapted for Varying Growing Conditions

We were driven to push the technology we created even further, and implement something that could record information wirelessly, which only needed minor human interventions to synchronize the data recorded wirelessly. This allowed us to research current designs for IoT devices and make a design based on the requirements for our use case.

In order to develop a scalable solution that fit the needs of agriculture, we devised a design that would allow the IoT sensors to operate as a standalone database, as well as a server for HTTP requests. The reason for allowing an IoT device to act as a standalone database is that connection disruptions cannot hinder the saving of data.

One major constraint we faced while designing this device was that sometimes farms do not have a wireless Internet connection where data must be recorded or sent. For this reason, we designed the hardware in such a way that allowed the device to be fully portable. Therefore, the device can be brought to where a wireless Internet connection is stable and synchronize its data with other devices.

Once measurements have been taken, they can be requested by using the REST API at the endpoint hosted on the device. Any device, on a local network or on external networks, can make requests for this type of data if we authorize it. For the purpose of our application, the API can only be accessed by our servers, so that this data remains secure once it has been stored in the database.

Hardware

For this prototype device, we used microcomputers and breadboards in a configuration that extended the I2C bus to multiple sensors. On a perma-proto breadboard from Adafruit, we soldered a Power Boost 500 Charger onto pin headers along the power circuit. The Power

Boost 500 Charger supplies electricity from the lithium battery to the 5V rail, which allows the microcomputer, and multiple sensors, to be powered by an uninterruptable power source.

On the microcomputer itself, only one data pin and one clock pin are available. Therefore, a daisy chain was implemented onto the breadboard by using pin headers and wires. The wires helped to create a bus topology that snakes all the way down the breadboard. The snaking shape allowed us to create a daisy chain spanning horizontally, and vertically, across the entire breadboard. Through this process, we minimized the space used while making as many connections available as possible.

In a bus topology, only a limited number of sensors can be connected. The distance of the main line connecting these sensors is also limited. This is due to the fact that data is transmitted over a single, shared line. The more sensors there are connected to the main line, the more likely that data collisions may occur (Isaac Computer Science). As a result, the daisy chain is not infinitely scalable. However, we do not need to connect an infinite number of sensors to the device.

The main component of the device, the microcomputer, depends on a preconfigured environment. The operating system and a series of executable commands to prepare the device are stored on a flash memory card. When it is initialized, the device has the capability of installing necessary dependencies, running a database, and executing code. It is necessary for these steps to be performed "on boot" so that the device is prepared out-of-box. In order for the booting process to be completed, we created and enabled a few services.

Three separate services must be enabled in order to prepare the device. The first service is a program that brings the database online, exposing it to HTTP requests locally by creating a server. The second service is broken into two consecutive steps, the installation of several Python libraries and the execution of code dependent on these libraries. The third and final service is a program that creates another server. The purpose of the second server is to utilize Bluetooth Low Energy, a wireless protocol that allows the device to communicate with iOS or Android phones. Data such as the permanent location can be written to the device using a string in the same form as URL search parameters. This is an extra measure to ensure that a device can be configured without a digital screen included.

Software

The software on the device is run by using Python code. Each sensor on the device has a unique I2C address, allowing the code to recognize sensors and receive data from them. These measurements need to be sent somewhere for safekeeping. However, sending them to external servers was not in our best interest. When designing the software for this device, we imagined the technology being developed in such a way that it could have the least impact on the environment. These devices already act as their own servers. Thus, relying on a load-balancing network of servers in a warehouse operated by a large corporation would not be necessary.

With the database we chose, designing the software this way was not only possible but also scalable. In a growing operation as large as an industrial farm, these devices could reliably share the same set of data. In addition to acting as standalone databases, these devices could work together to contribute data to one another. The devices in each room or building can contribute the set of data they contain even when experiencing connection disruptions because of the way our database handles the synchronizing of data across devices.

Terminal Scenario

This scenario describes how data can be synchronized and how the device ties together with the software suite we developed.

A central terminal exists that acts as a “target” device. The terminal includes the necessary storage capacity for large amounts of data. With a mass storage device such as this central terminal, it is possible to collect information from numerous devices. The devices in each room or building of an indoors farm would act as "source" devices. The information gathered this way would be localized, on a Machine-to-Machine network. It would only be on the central terminal that a connection is made to a broader network. Therefore, regardless of disruptions or hindrances to their Internet connections, these farms would still collect data through the local area network.

This is made possible by the procedure our database undergoes, called replication. The devices deployed as "source" devices would periodically broadcast whether new information has been added or deleted. These changes are shared directly with the "target" device, which could update its own database to reflect the same changes. This procedure would allow any devices, on a local network or an external network, to replicate themselves to a central repository of data. To protect the data from a man-in-the-middle attack, these devices would only broadcast their changes locally.

A terminal is useful because it can be the sole provider of sensory information to an external server. The terminal's body is encased in a slate form factor, allowing it to resemble, and function similarly to, smartphones. By featuring a touchscreen, it can run several different types of applications without the need for peripheral devices attached to it. In addition to initializing the main database and collecting data from a range of sensors, it can run any of the applications in our suite of software.

One of the applications it can run is the user-friendly, cross-platform, and mobile-first application we developed for web, iOS, and Android. Included in this application is a recursive function that is only operable if it is run in a local environment. The operability of the recursive function is also based on a contingency, whether or not the database has been initialized. This is because the purpose of the function is to await change notifications from the database. Then, by following the sequence that changes are notified in, it sends new information to an external server.

Interoperability

Whether or not these devices have an Internet connection, they can still share data on a local network and make decisions based on live recorded information. The approach used to facilitate interoperability this way was Offline First. The devices need to operate within offline environments first, then to make data available to an external server in a sequence that the central device can handle.

We believe that the solutions this device provides could answer some of the problems experienced by smart agriculture. By handling the issue of connection disruptions, the device we created could be applicable in growing conditions without a wireless Internet connection. By creating events and listening for changes that are broadcasted across a network of databases, the device can also handle the need for larger sets of data being collected. We can see the device being useful for small scale growers, or enthusiasts, as well as large scale operations. All of this is possible because of the options we have available in terms of databases, sensors, and microcomputers.

References

BBC Bitesize (Ed.). (n.d.). The Bus Network - Network Types and Topologies. Retrieved December, 2020, from https://www.bbc.co.uk/bitesize/guides/z36nb9q/revision/4

Isaac Computer Science (Ed.). (n.d.). Topologies - Bus Topology. Retrieved December, 2020, from https://isaaccomputerscience.org/concepts/net_network_topologies