Wireless Sensors Network for Measuring Temperature and Humidity in Historic Structures

Alexander Martin 12/2/2024

Introduction:

Historic structures are often maintained in the same nature that they were initially built, when materials and methods were not the most modern. Because conservation tends to be conducted with historically accurate materials and methods, historical structures are often more susceptible to the effects of weather than more modern structures (especially to temperature and humidity). It is therefore important that the response of historical structures to weather is monitored, especially the effects pertaining to temperature and humidity inside these structures, so that these buildings are best taken care of.

Mission:

Explore the methods, practicality and scalability of wireless sensor networks for monitoring the humidity and temperature of historic structures.

Method:

As historical structures tend to be strictly managed by building owners or conservators, it is extremely unlikely that any undergraduate student experimentation about these structures would be allowed. Therefore, experimentation was conducted on a simulated historical structure, which exhibited similar responses to weather as a true historical structure. A wireless sensor network consisting of a combined humidity and temperature sensor, a Wi-Fi router, and a base station computer was set up to monitor the response of the simulated historical structure to weather. A workstation computer was used to analyze the recorded data. The sensor network was designed to be scalable so that more sensors could be used simultaneously to increase its practicality (although only one sensor was used in this experiment).

Simulated Historical Structure:

Due to the constraints for experimenting on historical structures, a simulated historical structure was utilized. This environment is part of a 44 year old building which could be legitimately accessed and experimented on, has been maintained (albeit somewhat poorly) with similar methods and materials as when it was constructed, and is greatly susceptible to the effects of temperature and humidity. In addition, it had access to reliable and steady electrical power for the sensor network to run and was likely to not be disturbed by unknowing individuals. Overall, it is very similar (if not identical) to a building classified as a historical structure and is suitable for experimentation to occur.

Parts and pieces:

The designed sensor network consists of a combined humidity and temperature sensor, a Wi-Fi router, and a base station computer. The router provides a means for the sensor computer and main computer to communicate locally over Wi-Fi without necessitating an actual connection to the internet. A typical exchange between the two computers involves the main computer requesting a reading from the sensor, and the sensor then measuring the temperature and humidity and transmitting it as a response to the main computer. The method of communication between the main computer and sensor utilizes Berkeley Socket technology to create a temporary, local connection between the two computers for simple and quick communication.

Each sensor is designed with a Raspberry Pi Zero W computer board and a DHT-11 environment sensor to measure the temperature and humidity of its nearby surroundings. The Raspberry Pi Zero W board runs a Python script and interfaces with the DHT 11 sensor via onboard GPIO pins. The Python program is responsible for receiving data requests from the main computer, carrying out measurements, and then transmitting the measurement data to the main computer. This entire process of taking and communicating measurements takes approximately between 1-2 seconds.

A Raspberry Pi 3B+ computer board is used as the main computer. This board runs a Python program which is responsible for transmitting data requests and receiving data measurements wirelessly from each sensor. Temperature and humidity data received from a sensor is consequently time stamped and saved to a CSV file for later analysis. The Raspberry Pi 3B+ board offers more processing power than the Raspberry Pi Zero W board, and is a more suitable option for requesting and processing the data from a sensor.

Both types of computer boards operate on Debian-based Linux software, which supports GPIO interfacing with Python by default, can manage long-term use without the need for system restarts, is skilled at handling internet

communications, and is easy to set up over SSH connections. In this experiment, the base station computer was connected to a monitor and keyboard for setup while the sensor computer was controlled with SSH from the base station computer. This simplified setup by eliminating the need to connect an additional monitor and keyboard to the sensor computer and showing all processes of the sensor computer on the base station computer in case of the need for troubleshooting.

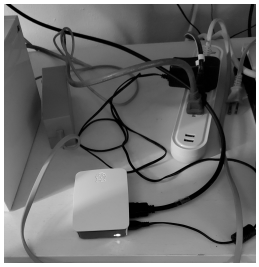

Figure 1: The setup of the Raspberry 3B+ computer used as the base station computer. The computer is plugged in to power, as well as to a monitor to easily observe the exchange of temperature and humidity data between this computer and the sensor.

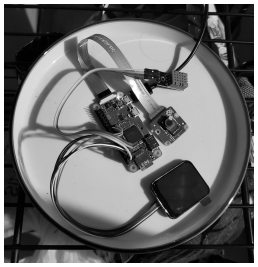

Figure 2: The setup of the sensor. The sensor consists of a DHT-11 humidity and temperature sensor connected to a Raspberry Pi Zero W computer board. This sensor takes measurements with the DHT-11 sensor and communicates them to the base station computer.

Figure 3: The display of the base station computer (with private information redacted). The display shows the programs running on both the base station computer and the sensor computer to easily observe the exchange of data between them. Each program displays the end result of each data exchange (the base station program is on the left, and the sensor program is on the right).

It was decided that electrical power from the simulated historical structure's electrical outlets would be sufficient for this experiment due to concerns about the reliability of using generators or batteries instead. The electrical outlets carried far less risk of losing power than the generator or batteries. For this reason, the electrical outlets were ultimately chosen because of the risk of losing data to power losses in the generator or batteries should they not be serviced appropriately.

Code:
Both the base station computer's and sensor computer's programs communicate with each other via Wi-Fi Berkeley Socket technology. Berkeley Sockets are used on either a local or global network to enable two devices to communicate wirelessly. To communicate, each device must know the IP address and port number that the other device is using. In a basic exchange, one device sends a message to the second device's IP address and port number; the second device then receives the message. For communication to occur within the sensor network, the base station computer similarly must know the IP address of each sensor, and a socket port number must be established between the two computers. For these programs, the internet port 31450 was established. Using such a high port number reduces the risk of internet interference and data collision as commonly used port numbers range from 0-1023 (Wikipedia Contributors, 2024).

The Python program for the base station computer first executes the "send()" function. This function uses the "send_over_net()" function to send a request to the sensor at the IP address 192.168.1.24 over the internet port 31450 and ask for the temperature and humidity data. The requested data is received as binary data, also referred to as a bytes literal string (Robinson, 2023). The binary data is then converted to a list container with regular string expressions, parsed for the numerical values of the measurements, and logged to a CSV file. The program repeats indefinitely, waiting 60 seconds in between each data request (each recorded measurement is close to 1 minute apart because requests are made 60 seconds apart from each other with a delay of about 1-2 seconds).

```
import datetime
import time
import socket

HOST = "192.168.1.24"
PORT = 31450
result = [0,0]

def send_over_net(bean):
        global HOST
        global PORT
        global data

        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                s.connect((HOST, PORT))
                s.sendall(bytes(bean, 'utf-8'))
                data = s.recv(19920000)
                return str(data)

def send():
        global result
        try:
                while True:
                        result = send_over_net("get humidity")
                        print(result)
                        result = str(result).replace("'b", "").replace("'", "").split(" ")

        except:
                print("error")

while True:
        send()
        humidity = result [1]
        temperature = result [0]
        logTime = datetime.datetime.now()
        with open("humiditydata.csv", "a") as humidityLog:
                humidityLog.write(str(temperature) + "," + str(humidity) + "," + str(logTime) + "\n")
                print(str(temperature) + "," + str(humidity) + "," + str(logTime) + "\n")
        time.sleep(60)
```

Figure 4: The Python program running on the base station computer, which receives temperature humidity data and saves it to a CSV file “humiditydata.csv”. Adapted from Martin, 2023.

The Python program for the sensor computer runs a serving loop (function “serve()”) in which it waits for a request from the base station computer. When a connection is made and a request is received, the program takes humidity and temperature measurements and sends them to the base station computer in a string. This string is packaged with the temperature and humidity values separated by a space character (If the temperature is 40 degrees Celcius and the humidity is 50%, the string is arranged as “40 50”). The process of receiving a request, measuring the environment, and sending the reply takes about 1-2 seconds. An initial measurement is taken before the serving loop, to doubly ensure that the first measurement is accurate.

```
import Adafruit_DHT

sensorClass = Adafruit_DHT.DHT11
GPIOPin = 4

humidity, temperature = Adafruit_DHT.read_retry(sensorClass, GPIOPin)
print(f"temperature:{temperature}*C humidity:{humidity}%")

import socket

HOST = "" #Standard loopback interface address (localhost)
PORT = 31450  # Port to listen on (non-privileged ports are > 1024)
REPLY = f"{temperature} {humidity}" #My variable, what gets sent back to the client, the requested data

def serve(): #Communicate with the Client, receive the transmission, send the requested data
    global HOST
    global PORT
    global REPLY
    global temperature
    global humidity

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        conn, addr = s.accept()
        with conn:
            print(f"Connected to the client at {addr} ")
            while True: #Action starts here ----------------------------------------!!!
                #Receive transmission
                humidity, temperature = Adafruit_DHT.read_retry(sensorClass, GPIOPin)
                REPLY = f"{temperature} {humidity}"
                data = conn.recv(1024) #Receives a max of 1024 bytes
                message = data
                print(message)


                conn.sendall(bytes(f'{REPLY}', 'utf-8')) #Returns the amount of data left unsent
                #Use only one 'conn.sendall'


while True: #Maybe I can close server with ctrl+c
    try:
        humidity, temperature = Adafruit_DHT.read_retry(sensorClass, GPIOPin)
        serve()
    except:
        pass #Use try/except to bypass connection technicality
```

Figure 5: The Python program running on the sensor computer, which takes measurements of the temperature and humidity of the simulated historical structure and broadcasts them to the base station computer. Adapted from Averous, 2021 and Martin, 2023.

Due to difficulties with using the SSH tools and applications installed on the workstation, it was decided to instead set up a small web server on the base station computer and access the CSV file over the local network with HTTP protocol (see ***Figure 6***). The web server used for this task runs on NodeJS, an industry leading language for programming web applications that is a subset of the JavaScript programming language. Any content served with a NodeJS web server can be accessed from an internet browser (or any internet-enabled program). Similar to the program for the sensor computer, a serving loop is first initiated. This causes the program to wait for an incoming connection from a workstation. When a connection is established, “humiditydata.csv” is read by the program and is sent to the workstation (along with other HTTP details, such as the mime type of the resource).

```
var http = require('http').createServer(handler); //require http server
var fs = require('fs'); //import filesystem module
http.listen(80); //bind to port 80
var mimes = [
   'text/javascript', //JS
   'text/html',       //HTML
   'text/css',        //CSS
   'image/png',       //png
   'image/jpeg',      //jpeg, jpg
   'image/gif'        //gif
]
function handler (req, response) { //create server
   reply = 'no reply'
   console.log(req.url);
   var path = new String(req.url);
   if(path == '/') {
      reply = 'humiditydata.csv'
      mimetype = mimes[1]
   }
   fs.readFile(reply, function(err, data){
      if (err){
         console.log('404 - Site or resource not found');
         console.log();
         response.writeHead(404, {'Content-Type': 'text/html'});
         response.end('404 - Site or resource not found');
      } else {
         console.log(reply + ' [' + mimetype + ']');
         console.log();
         response.writeHead(200, {'Content-Type': mimetype});
         response.write(data);
         response.end();
      }
   })
}
```

Figure 6: The NodeJS program running on the base station computer, used to access the CSV file from a workstation. Adapted from At216, 2023.

The setup of this sensor network is conducive for expansion. Multiple sensors can be connected to the base station computer in at least two ways. First, the base station computer's program could be run multiple times for each sensor added to the network as long as each subsequent program acknowledges the unique IP address pertaining to the sensor it communicates with. This would be the easiest method for scaling up the sensor network, but could encounter issues when multiple programs try to access and write to the same CSV file. To fix this problem, a separate CSV file could be created for each sensor and sensor's program so that no issues occur. These CSV files could be combined for analysis (if desired). A different approach to scaling up the sensor network would involve editing the base station computer program to connect to a series of IP addresses asynchronously. This could be accomplished by writing all IP addresses into a list, where the program could iterate through each IP address with the "send()" function. This approach is similar to the first, but requires a little more setup. However, it necessitates only one program, never encounters errors when writing to the CSV file, and potentially has no need for multiple CSV files.

Results:

Data was recorded automatically by the base station computer in a CSV type file. The CSV file is formatted so that each measurement is recorded on its own line. Each line shows the temperature, humidity, and date, all delimited by commas. It seemed that this formatting method would be the best for exporting the data to a workstation for analysis and visualization. Difficulties arose when it came to wirelessly accessing the base station computer (the base station computer was considered to be physically inaccessible and therefore only accessible over the network) and copying the CSV file to a workstation. It proved to be somewhat difficult to download the CSV file to the workstation over a SSH connection because of troubles with SSH tools and applications; it was consequently decided to set up a HTTP NodeJS web server on the base station computer and serve the CSV file to the workstation's internet browser application as a web page (see At216, 2023). Once accessed and downloaded to the workstation, another problem was found: formatting between the CSV file on the base station computer and the web page downloaded to the workstation had not been preserved. HTML syntax dictates that lines are broken with a "
" tag (not the standard comma delimiter or physical line break used by CSV formatting, see At216, 2024); the CSV inspired web page displayed one long stream of data separated by tab indents instead of line breaks. The web page data had to be downloaded, parsed for tab indents, and saved with all tab indents replaced with "
" tags. The web page data was then reopened in the internet browser application and downloaded again, this time with the line breaks present. The web page data could then be saved as a CSV file and loaded into a spreadsheet application.

Data was continuously collected by the base station computer in near-minute increments for the entire time the network was active. As thus, a considerably immense amount of data was collected from the sensor network (over 5000 data points filling 15000 spreadsheet cells). The final version of the CSV file saved on the workstation was loaded into a spreadsheet application, and all data was grouped by the day each was measured. The data was too extensive to put all of it into one figure (as determined by the limitations of the spreadsheet application), so the

average humidity and temperature per day was calculated and depicted in ***Figure 7***. Other types of figures and statistics might prove to be more helpful, but for the purposes of the experiment ***Figure 7*** will suffice. The sensor network could accurately depict weather trends in its data, especially weather trends that relate to changes in an environment's temperature and humidity. For example, per ***Figure 7***, days 4 and 5 depict elevated humidity levels with temperatures similar to those of days 1, 2, and 3. In comparison to the actual weather that occurred, days 4 and 5 both experienced moderate to heavy rain (hence the elevated humidity in these two days), while days 1, 2, and 3 were moderately to mostly cloudy without any precipitation. All five days experienced a cool front (hence the lower humidity in all five days.

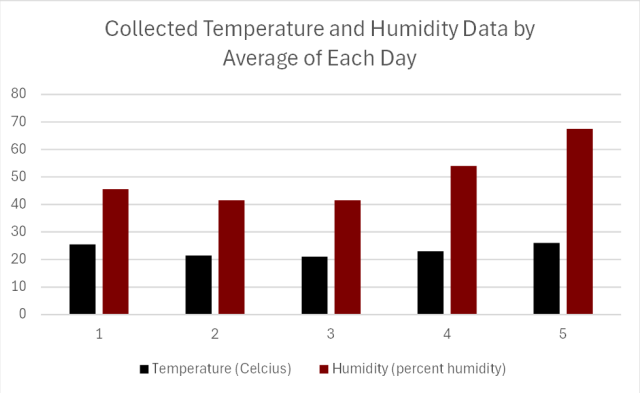

Figure 7: A plot of temperature and humidity statistics retrieved from “humiditydata.csv”.

Conclusion:

As historic structures tend to be very susceptible to the effects of weather, especially the effects of temperature and humidity, it is important that they are monitored to determine the magnitude of these effects and help building owners or conservators better preserve them. Temperature and humidity sensor networks can play a large role in helping building owners or conservators in this way, and are a practical system to actually do so. As tested with a small temperature and humidity sensor network in a simulated historical structure environment, the relevant data can be accurately collected and depict trends between weather and the response inside historical structures. For a sensor network, the base station computer should be able to easily process data and handle large files. For this reason, the Raspberry Pi 3B+ computer board is a great option to use as the base station computer. Any programmable computer with similar capabilities as the Raspberry Pi 3B+ should work similarly well. The sensor computers should be able to communicate wirelessly with the base station computer, be programmable, and have some technology to interface with temperature and humidity sensors. The Raspberry Pi Zero W is a suitable board for this application, but any similar computer board or microcontroller with internet capabilities should also work well. Using Python programs to take measurements of the humidity and temperature, transmit these measurements over a Wi-Fi network, and save these measurements to a file proved to be suitable for sensor networks. In addition, should it be difficult to access a file on the base station computer via SSH tools, a NodeJS web server could be used instead. The sensor network was adept at depicting weather events,especially those relating to changes in environmental temperature and humidity. For example, the sensor network was able to portray weather events such as cool fonts, cloudy skies, and rain when compared to the seasonal and forecasted weather patterns. Depicting weather trends in the data is helpful to building owners or conservators when determining how to best protect their historical structures, and is an important trait of the sensor network.

For purposes of scalability, the base station computer could run a separate program for each sensor as long as the program is edited to use the correct IP address for its sensor. In addition, it is advisable that each program uses a separate CSV file. A more complex method would be making the base station computer’s program aware of the IP address of all sensors, and could be set to connect to each sensor asynchronously. Depending on preference, a different CSV file could be created for each sensor. This sensor network makes use of open-source software,

particularly in the Python and NodeJS programming languages and the Debian based operating systems. This network is scalable in the sense that it supports the connection of multiple sensors to one base station computer, but it is not scalable in the sense of manufacturing and selling this network without any restrictions. Per the open-source licenses on each of these tools, there exist liability waivers and limitations (among other items) beyond only acknowledging the initial creators. In addition, some of the programs used in this sensor network are of mixed-origin or are proprietary to their authors. To profit from this sensor network, one might have to apply the concepts of this sensor network to build their own network or possess this sensor network and market its use as a service.

Overall, this sensor network was very practical for monitoring the humidity and temperature inside of a historic structure. The base station computer was able to communicate effectively with the sensor and record thousands of temperature and humidity measurements across the span of the experiment. These measurements were later helpful in determining statistics for the temperature and humidity and the response of the simulated historical structure to weather. The scalability of this network definitely enhances its versatility in other scenarios that necessitate the use of additional sensors. Although scaling the network takes additional work in modifying programs on the base station computer, it is a simple and quick task requiring little technical knowledge of programming. As a whole, sensor networks are very practical for monitoring the humidity and temperature inside of a historic structure. A sensor network that undergoes more than a few days of development, with higher quality components and more powerful programming, would be more than sufficient. In addition, such a network would most likely be more easily scalable. With this sensor network (and often most sensor networks), the need for electrical power is substantial. For the two computer boards used in the experiment, each board requires around 5 Volts and 2 Amps to function well. During measurements and measurement recording, especially as the computer boards accumulate heat, the power consumption could potentially increase. In lieu of a constant electrical source (such as from a wall outlet), generators or large batteries compatible with these power requirements should be used to power these boards. In addition to these boards, the Wi-Fi router also necessitates its own power and may not be satisfied with a battery. Any historical structures without their own electrical power sources would be more difficult to monitor, as generators would require refueling and batteries would have to be changed or charged from an outside source of electricity. Any historical structures with intermittent electrical power could experience losses of measurements during times of low power. Any situation where the boards don't receive enough power to run could risk the corruption of the CSV files or the boards' operating systems as a whole. In addition to power concerns, the size or design of the historical structure should be considered. Any structures that require sensors to be placed outside of the range of the Wi-Fi router would require W-Fi extenders to be installed in the network. In addition, any structures that require sensors to be placed beyond walls or objects that block electrical signals would require Wi-Fi extenders to map the Wi-Fi network around obstacles. Additional Wi-Fi extenders would increase the need for power, and could become a problem if the historical structure has intermittent or no electrical power. As previously discussed, such structures might require the use of generators or batteries that are constrained by their own sources of power.

References:

Averous, Pierre 2021. "Setting Up A DHT11 Sensor On Your RASPBERRY PI — The Easy Way." Medium. https://piaverous.medium.com/setting-up-dht11-temp-hum-sensor-with-your-raspberry-pi-b408c0d4490f.

Alexander Martin, 2023. "Python Servers." Python Servers. https://servers-python.web.app/

At216, 2023. "Welcome to ServersNJS - Server Maker." ServersNJS. https://serversnjs.w3spaces.com/

At216, 2024. "Line break standards between CSV and HTML". At216's Web Blog. https://at216.github.io/blog/linebreaks.html

Wikipedia Contributors, 2024, "List of TCP and UDP port numbers." Wikipedia. https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

Robinson, Scott, 2023, "The 'B' Prefix in Python String Literals." Stack Abuse. https://stackabuse.com/bytes/the-b-prefix-in-python-string-literals/