Temperature Excursion Tracker

IoT Ambient Sensing and Data Collection

Joseph D Ramli
Computer Science Masters
SUNY at Binghamton
Binghamton, NY USA
jramli1@binghamton.edu

ABSTRACT

The Internet of Things (IoT) generally refers to the presence of physical objects with embedded sensors which are in some way linked to software. While this is not the official definition, the field continues to evolve and be redefined as more advances in implementations and designs of this nature emerge. It is in this spirit that this project was created. The core focus of this project is to explore and implement a method that could take ambiently sensed environmental data and transmit it to a common data analysis software (in usable form for that respective software). Performing such a task can show the potential availability for these IoT devices for personal use. This may expand the perception of IoT beyond what some may assume is a 'commercially-oriented' field of study. The Temperature Excursion Tracker is an example of a simple home-based application that may be of interest.

1 Introduction

Monitoring and tracking temperature can be a very critical part of many processes or quality assurance protocols. This is not only true for things in the commercial world, but is even important for consumers as well. One example is that a consumer may have a medication that must be stored at a certain temperature in order for that medication to remain potent and effective. In this case, it may be useful for a consumer to have a device with an embedded sensor that can transmit temperature readings to data analysis software over a network.

There are many ways to transmit data from ambient sensors to high-powered computing machines, but this report aims to show a specific implementation using an Arduino Uno and a laptop computer.

2 Design

The design of this project includes both the hardware/software of the ambient-sensing hardware, and the subsequent receiving hardware/software that will have the data analysis tool. In this case, the design includes the Arduino UNO WiFi REV002 hardware, the Arduino IDE, and a Windows 10 laptop that has Microsoft Excel and Python 3.7 installed.

2.1 Arduino

The hardware that was used for acquiring ambient sensor readings was the Arduino Uno WiFi REV002 and the Arduino IDE for Windows 10. Using these systems alone, the data can be read/sensed from the environment and read through a USB serial-monitor directly to the screen, once programmed and set up properly. Additional components and cables are listed here:

1) A-Male to B-Male USB connector for Arduino Serial Monitor readings
2) Breadboard
3) 9V AC/DC power adapter for Arduino power supply
4) Red LED-light
5) Resistors (100k ohm)
6) Gikfun NTC 10K ohm 5% Thermistor Temperature sensor for Arduino
7) Connector cables from Arduino to Breadboard

A major advantage that this particular Arduino board has over many older boards, is that it has built-in WiFi capability. The WiFi component allows the Arduino to send data over a local network to any computer linked to the same network as the Arduino (properly set up with server software) without being USB-connected.

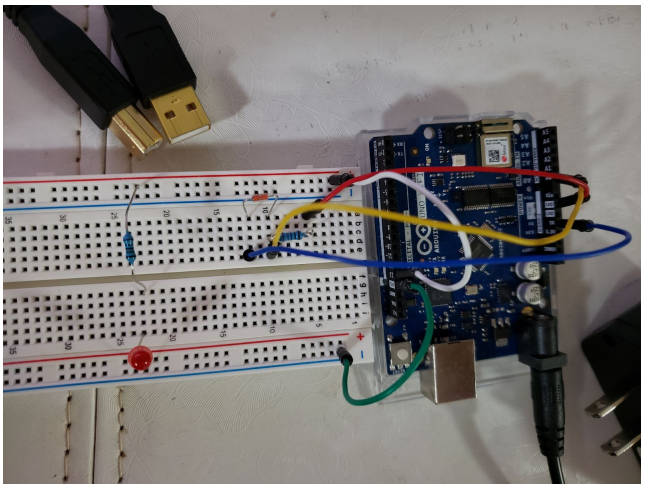

Figure 1: **Full Arduino hardware setup. Disconnected USB and power supply to show all components in this photo**

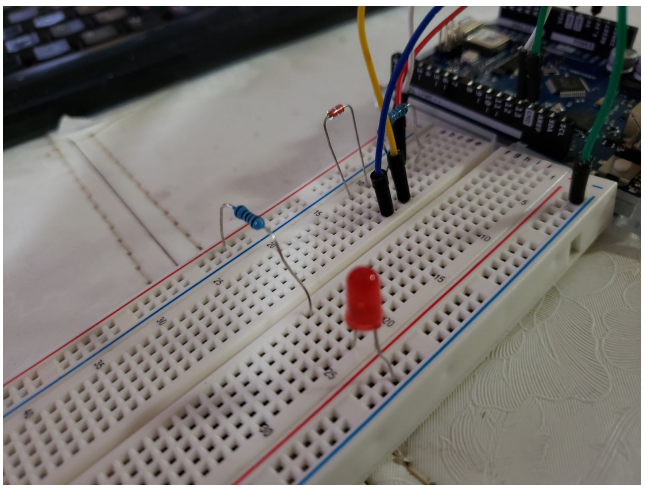

Figure 2: Angled to show LED setup and connections

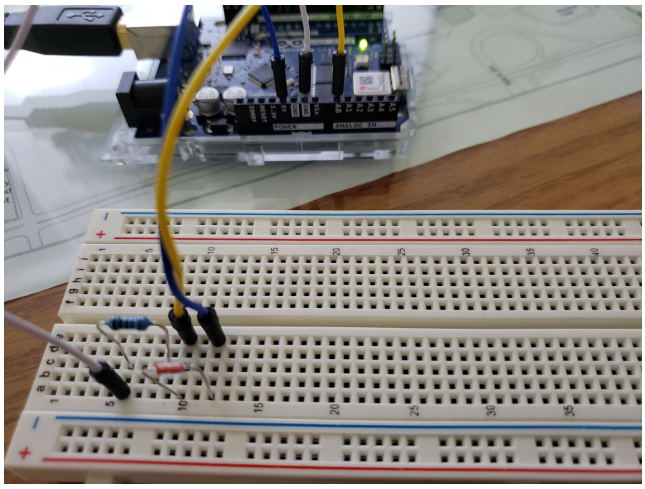

Figure 3: Angled to show thermistor setup

2.1.1 Arduino sensing - The software to program the Arduino setup is the Arduino IDE version 1.8.13 for Windows 10. Using the appropriate resistor, thermistor, and LED setups, various calls to the "analogRead()" function were made to read the sensors. There is an onboard temperature sensor that was read as well using the SparkFunLSM6DS3 module reference guide.

2.1.2 Arduino alert LED – There is a red LED that is lit up if the temperature exceeds a certain limit. This is not required for radio-transmission of the data. That LED-alert is an added feature for the user to be able to see when a temperature may be exceeding the set threshold.

2.1.3 Arduino transmission of results - Following the examples from Arduino.cc and the WIFI NINA library, code was generated and adapted to Send/Receive UDP strings that are populated with the current temperature reading along with a time-stamp. The Arduino side is programmed to wait for a signal from a server, and reply with a string of its own (attached to the received string) when that UDP signal is received. In this case, when it receives the time-stamp from the server, it attaches the current sensor reading and returns the UDP message to the sending server IP address.

2.1.4 Sensor Calibration - Unfortunately, this particular setup does not demonstrate consistent accuracy and precision. Attempts were made using the Steinhart-Hart equation alone, but this was not sufficient by itself without the ability to read/calibrate the hardware. Additional calibration and troubleshooting with voltage equipment is likely necessary to get a fully accurate auxiliary thermistor reading. The existing setup was used to get close to room temperature readings without the use of the Steinhart-Hart equation, to at least demonstrate the data-flow with values within about a 10% margin of error.

2.2 Windows 10 laptop

2.2.1 Python storage of data - The first critical software piece is the Python 3.7 server code. This code is what broadcasts the initial timestamp message to the Arduino board's IP address, allowing for the receipt of the sensed-data response. This particular implementation is set to ping the waiting-Arduino a user-specified number of times, with a user-specified number of seconds of delay between each ping. e.g. The user can enter at each prompt "100" and "5", and this will give 100 temperature readings at 5-second delays apart from each other. To get a reading of the temperature every 10 seconds over a 1-hour span, the user could simply enter "360" and then "10" at the first 2 prompts respectively. The data is stored progressively into a list data-structure. The final user-prompt will accept a temperature value that will be used to print a 'HIGH' string next to the reading, if the reading exceeds that user-input value.

2.2.2 Python export to Excel - Once the Python code finishes its loop iterations to get the data, the list is used to export the data to an Excel file using the xlwt Python package from the PyPI (Python Package Index). This file is then named/created and the results are exported and printed via loop through the data-structure. The resulting file is now ready for traditional use by the individual or professional that may be familiar with Excel. Sample output is shown below.

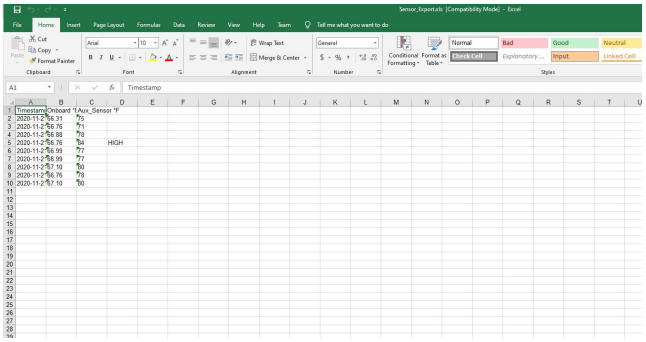

Figure 4: Sample output of timestamp and readings

There is exception handling to create a backup file with the word 'Alternate' at the end, to indicate that there was an issue in saving the first file (i.e. if the other file was left open or not accessible for some reason).

3 Discussion/Design Rationale

3.0.1 Arduino Choice Rationale: The Arduino hardware/software is a highly accessible and growing project-base, that is user-friendly for individuals doing small-scale IoT projects. Being able to handle and experiment with such compact and succinct hardware can be very beneficial for understanding the flow and appreciating the design of IoT applications. Most anyone can create and implement a design using an Arduino at very low cost, therefore using this design seemed most pertinent for a non-commercial project.

3.0.2 Python Choice Rationale - Python is a very flexible and succinctly written programming language, that allows for very powerful interfacing with other file applications. Having the data in a python data structure is very effective for making a simple export of that data to a Microsoft Excel file. In addition, the versatility of storing the data in a python data structure allows access to PyPI packages that can rapidly aid in exporting the data to other programs besides Microsoft Excel.

3.0.3 Excel Choice Rationale - The Excel software was chosen because it still seems to be a widely-used and familiar software for collecting and analyzing data sets among businesses and professionals. Despite the rise of so many new data collection and analysis systems/tools, this has been a popular and familiar software that almost any individual will have some experience using.

3.0.4 Synergy of sensor and server - The Arduino is programmed in a way that it can wait and take UDP cues from a server that is coded in a common language such as Python, C++, or Java. Python was chosen for this assignment because of its notorious ability to perform complex tasks with brief and often easy-to-read lines of code. The flexibility provided to the user when running the Python program synergizes with the 'wait and reply' format of the Arduino's sensing/sending design.

4 Conclusion

The Temperature Excursion Tracker built here is a good example of how much can be accomplished with commonly accessible hardware. This particular design is somewhat limited in practicality due to the sensors being attached to and powered by a breadboard. However, any long-durable-cord or cordless sensors can change the practical applicability of this general design. By example, a remotely powered and compact sensor that can transmit its readings wirelessly (perhaps via Bluetooth) to a receiver may have a connector on the receiving end to plug into an Arduino pin for analog-reading. If something like this is developed, then the temperature sensor could be placed in more remote or extreme conditions that would otherwise damage the Arduino, breadboard, or connector cables. While this is not hardware that I found for this project, it seems like something worth exploring for anyone using this sort of template, as it would open up a myriad of possibilities for remote sensor placement and data-transmission. The practical applications and usability of remote sensors of this nature would likely be greatly augmented.

REFERENCES

[1] https://www.arduino.cc/
[2] https://www.arduino.cc/en/Reference/WiFiNINA
[3] https://www.python.org/
[4] https://www.geeksforgeeks.org/socket-programming-python/
[5] https://www.geeksforgeeks.org/sockets-python/
[6] https://www.geeksforgeeks.org/datagram-in-python/
[7] https://create.arduino.cc/projecthub/iasonas-christoulakis/make-an-arduino-temperature-sensor-thermistor-tutorial-b26ed3
[8] https://www.geeksforgeeks.org/writing-excel-sheet-using-python/
[9] https://forum.arduino.cc/index.php?topic=87329.0
[10] https://wiki.python.org/moin/UdpCommunication