Air Quality Measurement Device

A Python and Raspberry Pi Implementation

Wesley Romary, Brandon Percin

CS370 Colorado State University

Fort Collins, CO, USA

I. Introduction

This project consists of an air quality monitoring sensor array, screen for display readout, and a backend server for safe offsite storage and retrieval. This device is designed to be portable and easy to set up and in order to test the air quality in different rooms or areas while requiring relatively low amounts of power and a minimal cost for production. Air quality is, of course, not only a quality of life issue but a safety concern as well, and as such, the existence of a cost-effective and low-power device is very much welcomed. This project is simply titled the *Air Quality Measurement Device* in order to make its goal self-evident.

II. Problem characterization

The purpose of this device and set of programs is to provide an easy method to measure air quality through the use of four sensors: one for temperature, dust particles, relative air purity, and for potentially poisonous gas concentrations. In order to do this, a board must be able to retrieve data from four sensors, make note of the time of its observations, and write to the offsite database. Another board or computer must then process the requests and store the uploaded data. All of this must be done while simultaneously minimizing cost and power consumption.

III. Proposed solution and implementation strategy

The chosen solution was to one Raspberry Pi 3 model B+ for capturing and transmitting sensor data and a computer for displaying data in a portable manner (although the display program is portable and can run on any hardware capable to run Python3, including the sensing Raspberry Pi). The Raspberry Pi runs the stock Raspbian operating system, a derivative of Debian. Both the display and sensor applications use Python3. The sensing Raspberry Pi required four sensors in order to measure all of the intended aspects of the air: the Grove Gas Sensor (MQ5), the Grove Dust Sensor（PPD42NS), the Grove Air quality sensor (v1.3), and the Temperature Sensor (TMP36).

The storage server is running Fedora as that is known to be a stable and secure operating system for servers. All of the software on the server is run in Docker for the container orchestration. Docker allows this project to be run on potentially any major operating system with no alterations to the code. There are two containers relevant to this project for data storage: PostgreSQL and Flask/uWSGI. PostgreSQL is a relational database that also allows the user to have JSON column which is necessary for the kind of data that is required to store. This project utilized the “stock” PostgreSQL docker image with no modifications. Flask running with uWSGI instances is how the server is able to run multiple instances of Flask seamlessly. Flask itself is a lightweight Python microframework with our code added to perform database operations over a REST API.

A. Costs and Limitations

These devices, too, had their costs and limitations. As everything is powered through the Raspberry Pi USB power supply, there is a clear upper limit to power consumption. USB standards state that a constant supply of 5.0 V will be supplied to the Pi , and Pi 3b+ standards state that the current within the power supply has a bottleneck at 2.0 A, meaning the upper bound of power used by the Raspberry Pi with all components attached is 5 V x 2 A, or 10W of power [1]. Particularly, the devices have the following specifications: the Raspberry Pi 3b+ ranges from 1.9W at 350 mA to 5.0W at 950 mA depending on load [2]. Total load (when not powering a display) is approximately 3 W. The 7 Inch Touchscreen Display (which is used to mitigate power consumption, rather than using a typical monitor) has extremely

variable power drawing, but an estimate can be calculated. It is connected to a Raspberry Pi 3 B+ power supply, and therefore limited to a maximum of 2 A. Therefore, at most, the display current draw is 2,000 mA minus the maximum draw of the board, or about 1000 mA, 1 A. The Grove Gas Sensor requires approximately 0.1V from the Raspberry Pi, with an amperage of at least 300 mW to properly heat the gasses. It can measure gas concentration ranging from 200 ppm to 10000 ppm, but this is returned relating to a mixture of gasses and is difficult to work with. Despite this, it is accurate to within 0.02% - 1% of the actual value. The sensor itself costs $7.90 [3], making it fairly cheap. The dust sensor requires approximately 5.0 V to properly function. It detects particles larger than 1 um and costs $11.50 [4]. The air quality sensor requires 5V to function. It works by measuring gas concentration relative to a starting concentration. It is accurate to within 1 ppm for ranges 10 ppm - 1000 ppm. The sensor costs $9.90 [5]. The temperature sensor requires approximately 3.3 V of DC current to function. The sensor performs best within the range of -40°C to 125°C and is accurate to 1.5°C of the actual temperature, costing only $1.50 [6]. Despite these limitations, all of these components are both functional and cost-effective enough to warrant the production of this air quality sensor.

B. Methodology

The sensor software has been broken up into three parts, the device's code to poll sensor data and pass it on to the server, the code on the server itself to manage the database and other functions, and the code for the display graphics. The Raspberry Pi collects sensor data for a set amount of time and then uploads that data to the server in a JSON payload. That payload is then retrieved by the Flask server and put into the PostgreSQL database for storage. Later on, if the user would like to view historical data the Raspberry Pi sends a request for a certain time period of data. The server processes that request and returns all of the table rows that potentially contain that time period back to the Raspberry Pi as a list of JSON payloads.

The display software relies completely on the sensor software and its corresponding server. The display device (typically another Raspberry Pi) runs display.py and is presented with a graphical display of four graphs, each of which plot values retrieved from the sensors over time. Many aspects, such as background image, resolution, number of data points per graph, graph color, graph title, graph size, and graph position, are all open to user customization through the provided config.json. This allows devices with higher display resolutions to benefit from increased accuracy. Each graph also displays a corresponding mean and running mean in order to visualize trends in the data.

C. Libraries and Creations

The sensors utilized the "grove", and "RPi.GPIO" libraries. The grove library was created to interface with Grove sensors and shield which were used to connect our sensors to the Raspberry Pi. This library is created by Seeed which sells the groove sensors. The RPi.GPIO library is the standard GPIO library on the Raspberry Pi and it is used to interface with the dust sensor over the on board pins as the recommended grovepi library does not work. The grove library is open sourced free software under the MIT license. The RPi.GPIO library has a GPL-like free software license[7]. The Raspberry Pi also utilizes the "requests" module for POST and GET requests to the server as well as "datetime" for timestamp support and "json" for json support.

The software stack on the server is much more considerable. There are a total of 2 containers, one for the flask server and uWSGI, and one for a PostgreSQL database. Docker is open source under the Apache license.

Flask is a micro web framework written for python. Our flask container is responsible for handling requests between the Raspberry Pi and the database through REST API endpoints. This secures our database from unwanted queries as well as abstracts the storage logic from the sensor logic. Many libraries in Flask were utilized including: sqlalchemy, flask, datetime, json, and os. Flask is also open source under the BSD license.

Finally, the server also has a container for PostgreSQL. This is a relational database management system which allows us to store not only relational data but also json data which is useful when working with large datasets. PostgreSQL is also open source under its own, self-titled, license.

The display program relies heavily upon several libraries. The libraries time, datetime, random, math, and json require no installation as they are built into Python and require no further explanation. The library requests is used to make specific requests to the server regarding the search interval for data points on the server.

The library graphics was the most important library. The library graphics.py was developed by John M. Zelle,

Ph.D., in order to create a simple and lightweight method for drawing polygons, text, points, and lines to a window [8]. The library boasts many internal classes that are useful for creating shapes while also boasting dynamic redrawing based on the modification of shape objects. Despite this, the library is unable to create graphs. This was settled through the creation of another library.

The name of the library developed for the display program is graphs.py, which comes included with the display program. The library itself uses the time, math, and graphics libraries. The library includes a single instantiable class called Graph that can be initialized given a display object, center point, width, height, range of data values, number of points to show, title, color, and units. After it is initialized, the graph draws itself to the screen and awaits the addPoint() method to be called on it with a value. Points on the graph are stored in a list of size N, popping the first point off of the list when N is surpassed. The mean is calculated by taking the average value of each of the points currently displayed on the graph. The total number of points processed is kept in order to keep a running mean of these smaller means. The two together are drawn to the screen in order to help the end user view trends in data. The library is not limited to the four graphs utilized in this case and can be used in any program that uses the graphics library.

The actual display program, display.py, works by reading in the config.json, opening a display window, creating four graph objects from graphs.py, and periodically checking the server to see if new data points have been produced. If so, addPoint() is called for each of the graph objects and the graphs are each updated. The program determines whether a point has already been added by the timestamp on the foot of each data block. The display program runs continuously until the display window is manually closed, allowing it to run in the background quite easily.

IV. Conclusions

The *Air Quality Measurement Device* that has been produced is undoubtedly a cost-effective and energy-efficient device for its goal, as evidenced by the data in the costs and limitations subsection. In all, the device costs $30 for the sensors and wiring and $40 for the Raspberry Pi for the sensor model, for a total of $70. Only one sensor module must be purchased. Optionally, a display for the Raspberry Pi can be purchased for ~$50, which can then be attached either to the sensing Raspberry Pi or another Raspberry Pi that solely runs the display program. In its place, any computer with a display capable of running Python3 can also be used at no additional cost. At any time, the sensor Raspberry Pi will not draw more than 10W of power, even when attached to a display. This data and the customizability provided in the display program provide a user-centered, cheap, and low-powered solution to monitoring air quality for user safety.

References

[1] “Introduction to USB Power Delivery” - A. Sengupta. [Online]. Available at: https://www.electronicdesign.com/interconnects/introduction-usb-power-delivery. [Accessed: April-2019].

[2] “Power Consumption Benchmarks” - Raspberry Pi Dramble. [Online]. Available at: https://www.pidramble.com/wiki/benchmarks/power-consumption. [Accessed: April-2019].

[3] “Grove - Gas Sensor(MQ5)” - Seeed Technology Co. [Online]. Available at: https://www.seeedstudio.com/Grove-Gas-Sensor-MQ5.html. [Accessed: April-2019].

[4] “Grove - Dust Sensor（PPD42NS)” - Seeed Technology Co. [Online]. Available at: https://www.seeedstudio.com/Grove-Dust-Sensor-PPD42NS.html. [Accessed: April-2019].

[5] “Temperature Sensor - TMP36” - SparkFun Electronics. [Online]. Available at: https://www.sparkfun.com/products/10988. [Accessed: April-2019].

[6] “Grove - Air quality sensor v1.3” - Seeed Technology Co. [Online]. Available at: https://www.seeedstudio.com/Grove-Air-quality-sensor-v1-3-p-2439.html. [Accessed: April-2019].

[7] “License.txt” - B. Croston [Online]. Available at: https://sourceforge.net/p/raspberry-gpio-python/code/ci/default/tree/LICENCE.txt. [Accessed: April-2019].

[8] “Simple Graphics Library: graphics.py” - J. M. Zelle, Ph.D. [Online]. Available at: https://mcsp.wartburg.edu//zelle/python/. [Accessed: April-2019].