Rice University
ELEC 327: Digital Systems Implementation
AirPatrol

Date Submitted: May 2, 2017

Fasai Phuathavornskul
Tianyi Zhang

Contents

1 Introduction

This past decade, we have seen a huge shift in environmental consciousness in the general population. We have started to see more investments go into sustainable energy research, and the admiration of electric cars, reusable rockets, and homes and buildings that generate more power than they expend has become mainstream. AirPatrol seeks to join and continue this movement towards a sustainable world by providing a means for users to collect their own temperature, Carbon-Dioxide (CO2), and Total Volatile Organic Compound (TVOC) measurements in the atmosphere with a quadcopter. We hope to familiarize the world with metrics used to measure air quality and seek to establish intuitions about what baseline measurements look like such that the statement "CO2 has increased to 550 ppm" is a call to action.

2 Overview

AirPatrol is an RC drone that carries a TVOC, CO2, and Temperature sensor and sends the measurements to its web interface. Its main frame and control is based on a Holy Stone drone. The user can fly the drone to areas of interest and monitor air quality measurements on the web interface. At night, the user can also turn on LED indicators to display beautiful colors.

Figure 1. Final System

3 Materials

3.1 Hardware

1. **ESP8266 Breakout Board with ESP12**: 2.4 GHz WiFi Board with microcontroller unit (MCU)
2. **CCS811 Air Quality Sensor with Breakout Board**: TVOC, CO2, and Temperature Sensor
3. **MSP430g2553**: Low-Power Microcontroller
4. **APA102 LEDs**
5. **3.7V Li-Po Battery**
6. **Holy Stone F181C RC Quadcopter and Controller**: RF-Controlled Drone with HD Camera

3.2 Software

1. **Arduino IDE**: For the development of ESP8266 Firmware
2. **Code Composer Studio**: For the development of the MSP430 Firmware (Windows)
3. **JSFiddle**: For the Javascript & HTML development of the Web Interface

4 Methods

The web interface is hosted on the ESP8266's server on a personal hotspot WPA network. The sensor communicates with the ESP8266's MCU through I2C, and the the ESP8266 communicates with the MSP430 through SPI. The MSP430 controls the indicator LEDs through SPI. This communication scheme is outlined in **Figure 2.**

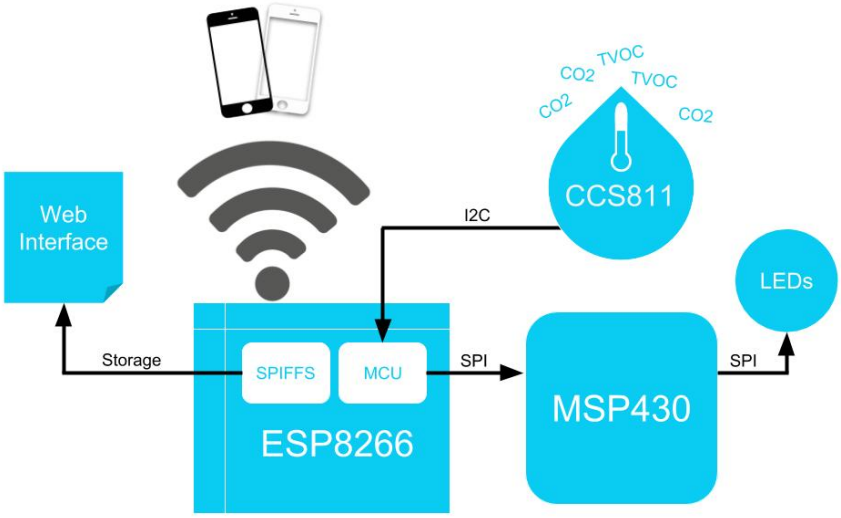

Figure 2. Communication Module

4.1 Front-end: Web Interface

The contents of the Web Interface are written in Javascript and HTML. The file fetches CSS and Bootstrap files from the internet so that the formatting files do not have to be stored in memory. The index.html file itself is stored in the SPI Flash Filing System (SPIFFS) area of the ESP8266, which is a partition of the system's Flash memory that can act as a file system. Upload to the SPIFFS requires an additional ESP8266 Plug-in, referred to in (2).

The ESP8266 is set to operate in server mode, in which it hosts the web interface, accessible on any browser on mobile or PC through its IP address. The bulk of the web interface was taken from (3) and adapted to work with AirPatrol's LEDs and sensor data.

When the web interface's client sends a GET request to the IP address of the ESP8266, the webpage is loaded, and a GET request to /chart.json is sent every 10 seconds to retrieve new sensor data. The sensor data is obtained by the ESP8266 MCU, formatted into a JSON, and stored at /chart.json for the webpage to retrieve from.

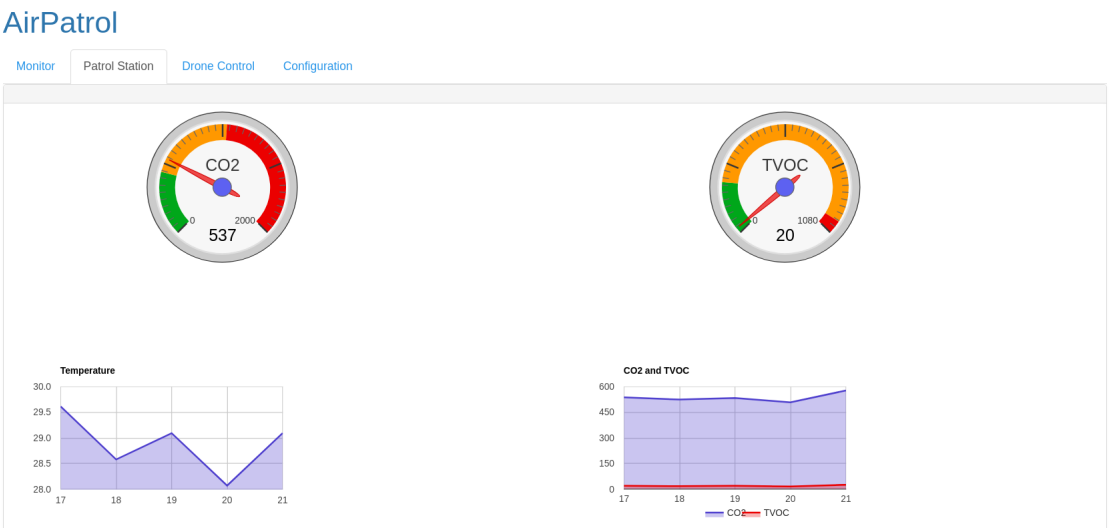

Figure 3. Front-End Charts and Gauges

In order to control the indicator LEDs from the web interface, the client sends a POST request to /gpio with parameters *id*, which is the selected LED mode, and *state*, which is whether the mode is selected or not. The ESP8266 MCU reads from /gpio and sends the LED Mode variable over to the MSP430 to light the LEDs in the requested mode.

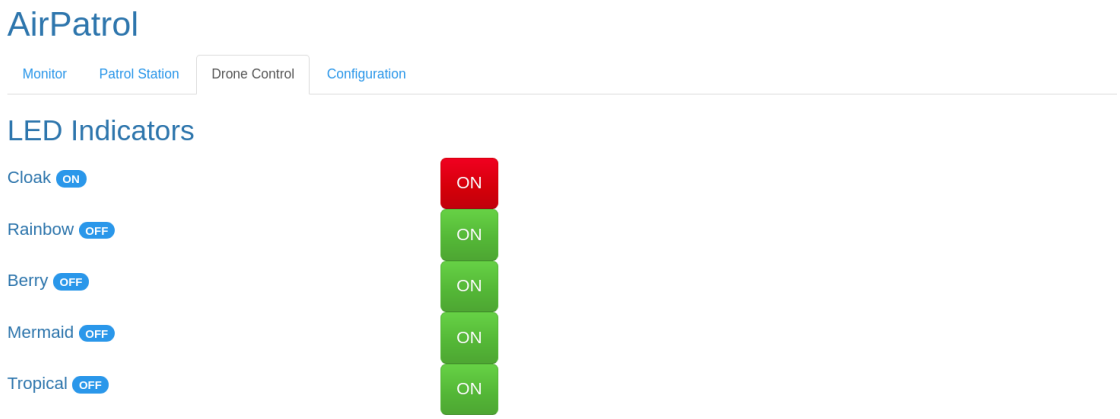

Figure 4. Front-end User LED Control

Table 1. Endpoints and Data Stored

Endpoint	Data Stored	Requests
/chart.json	Sensor Data from the CCS811	GET
/measures.json	Tabulated Sensor Data from CCS811	GET
/tabmeasures.json	Sensor Data for Tab Display	GET
/gpio	LED mode (Parameters include *id* and *state*)	POST
/application/json	Application History JSON files	

4.2 Back-end and Hardware: ESP8266 MCU & MSP430

Our hardware system schematic is shown in **Figure 3** below.

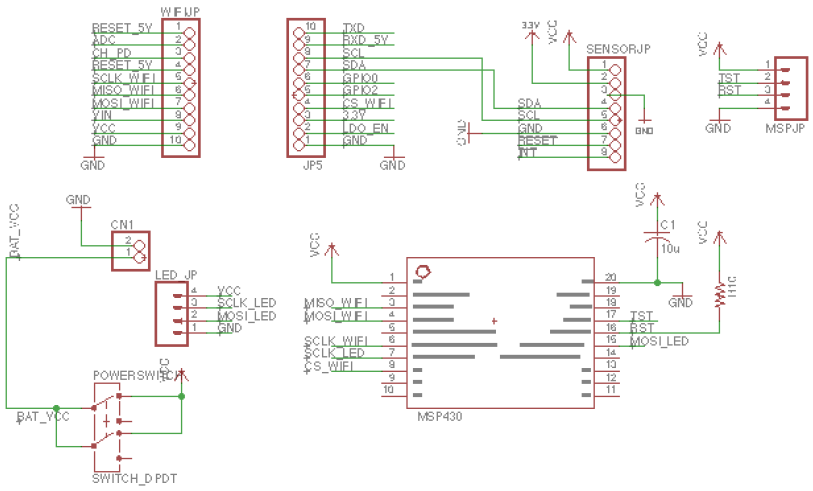

Figure 3. System Schematic

The CCS811 sensor communicates with ESP8266 via I2C. Then, the ESP8266 transmits the sensor data and the LED mode as the SPI master to the MSP slave. The USCIA0 module is used to receive data from the ESP and triggers an RX interrupt when the transmission begins. The MSP uses USCIB0 to transmit the byte sequence to control the LED slave and triggers an TX interrupt. During the interrupts, the MSP goes into low power mode(LPM0).

The ESP8266 MCU is programmed with Arduino. The ESP8266:

1. Connects to the WPA network
2. Sets up the Server to host the static page
3. Writes data to endpoints and answers & handles HTTP requests
4. Reads in sensor data through I2C and creates the JSON files to hold the data
5. Transmits LED mode to MSP430 through SPI

Most of the main code except for the SPI communication with MSP430 has been adapted from (3) as well. There are many Arduino libraries that help achieve the functionalities listed above, so most of the challenges were related to communication with the MSP430. On the ESP side, we used the Arduino SPI module to transmit the LED mode to the MSP slave. During the development process, we used an echo program for the MSP and printed out the data echoed back from the MSP to the Serial Monitor to troubleshoot the ESP-MSP communication.

Once the MSP430 receives the indicated LED mode from the ESP8266 MCU, it sends the corresponding LED color patterns to the APA102 indicator LEDs. Code for the transmitting to the APA102 was adapted from previously written code for ELEC327, rainbow.spi. Currently, there are four patterns: "Rainbow", "Berry", "Tropical" and "Mermaid". Tropical consists of a rotation of Red, Yellow, and Green. Berry consists of a rotation of Blue, Purple, and Pink, and Mermaid consists of a rotation of Blue and Green. We also included an off mode which turns off the LEDs.

5 Conclusion

5.1 Challenges

The most challenging part of the development of AirPatrol is the identification of a problem's origination. Since there are many components involved in a connected system, it is difficult to identify which component needs attention in a faulty connection. Most of the time, the system merely requires a soft reset in order to function properly again.

Building this project encouraged us to really understand wireless communication, HTTP, and the basics of web development. We shifted our mindset to thinking of the system as a whole instead of merely identifying problems within a unit. From debugging the web interface, the ESP8266 MCU, the MSP430, and a few random faulty connections, we learned how to use serial monitors to identify and fix problems. Our exposure to the capabilities of microcontrollers like the ESP8266 has inspired us to continue developing projects that include wireless, interactive systems like this one.

5.2 Improvements

Our original goal for the indicator LEDs was for them to change according to the the value obtained from the sensors. However, under the constraint of time, we were unable to synchronize the ESP8266 MCU and MSP430 to transmit multiple bytes and communicate with the LEDs all at the same time. We hope to implement this feature in the next upgrade.

6 References

1. "APA102 Aka 'Superled.'" Tim's Blog, 27 Nov. 2016, cpldcpu.wordpress.com/2014/08/27/apa102/.

2. "arduino-esp8266fs-plugin." Github Repository, esp8266, 29 April. 2017, https://github.com/esp8266/arduino-esp8266fs-plugin/releases/tag/0.3.0

3. "ESP8266-Webserver-Tutorials." Github Repository, projetsdiy, 30 April. 2017, https://github.com/projetsdiy/ESP8266-Webserver-Tutorials/tree/master/Part5_$DHT22WebserverESP8266_SPIFFS$.

4. "SPIFFS Filesystem." SPIFFS Filesystem - ESP-IDF Programming Guide v3.0-Dev-2280-g051d8d6 Documentation, esp-idf.readthedocs.io/en/latest/api-reference/storage/spiffs.html.

Arduino Libraries: ESP8266WiFi, ESP8266WebServer, WiFiClient, ESP8266mDNS, TimeLib, SPI, Adafruit_CCS811