YAAWS

(Yet Another Arduino Weather Station)

Alas, the need has arisen for yet another Arduino weather station. My contribution is driven by a couple of factors; the basic philosophies of weather reporting, a small contribution in the area of hardware and the ability for a software dummy like myself to accomplish some rather sophisticated programming.

This is a multipart series:

1. System overview, outdoor unit hardware and communication to MQTT
2. Mechanical and electronics construction of wind monitoring hardware
3. Raspberry Pi indoor unit, local display and logging to website

**Weather Station Philosophy**

There’s only one thing that engineers love more than LEDs – that would be graphs. But LEDs (unnecessary power consumption that no one sees when its inside a box) and graphs (history, not what I want to know when I am on my way out the door) may not the best design decisions for many users.

Gathering weather data is driven in a large part by what the user is hoping to understand. The data gathered, the sampling rate and the presentation are all affected by manner in which it will be viewed In my own case, I am usually interested in what the weather conditions are *right now* as it will affect my outdoor activities. Consequently, I want to know the cloud cover, temperature, wind direction and wind strength together with a short-term forecast. At other times, I like to muse over the long-term data such as atmospheric pressure trends or battery voltages.

Some data such as temperature, atmospheric pressure, humidity, air quality change relatively slowly whereas wind conditions can change from one second to the next. This affects the sampling rate and storage of data.

A final consideration is the presentation of the data. Immediate outdoor data should be always on, right there and obvious at a glance. Long term data can be accessed on a more indirect and infrequent basis such as a web browser.

**Hardware**

My previous weather station was based on an outdoor unit consisting of an Arduino Pro Micro (32U4) communicating to a suite of I2C peripherals and connected to an indoor Raspberry Pi via Nordic nRF24 link. The outdoor unit was low cost and consumed relatively low power but the nRF24 link was quite troublesome to program, particularly in Python. After seeing the great job that [Harald Kreuzer](https://www.haraldkreuzer.net/en/news/build-your-own-raspberry-pi-weather-station-weather-forecast-and-esp32-wireless-sensors) did, I was inspired to re-evaluate the entire system.

The outdoor unit in YAAWS is based on an ESP-01. This ESP8266 based board is just about the lowest cost single board computer available (~$1 from China). The beauty of the ESP01 is that it communicates directly over the local WiFi network and it can handle I2C sensors. The biggest downside is that it is relatively power hungry, consuming about 70 mA at 3.3 Volts and it’s a bit of a pain to program, but the code runs just fine on a D1 mini for more interactive debugging.

Part of the power consumption issue can be addressed by turning off the power when not sampling. This has the added benefit of performing a hard reset of the entire processor / sensor system. After all, what is the first instruction that the tech support person says - “turn off the power and wait 10 seconds”. The power supply is based on a photovoltaic cell charging a single LiPo cell through a low-dropout linear regulator which supplies 3.3 Volts to the ESP01. An I2C real time clock on the power supply controls the enable pin of the linear voltage regulator allowing the ESP01 to set the power off interval time. The PV and LiPo cells are sized to support expected needs.

The indoor unit remains a Raspberry Pi (3B+ in my case) with aways on display of immediate data and updating a website with long term data. An aside here: I have a closet full of old PCs just crying for the chance to do something useful – this would be a good project for one of those.

**Software**

A couple of notes about the project. One of the problems that I constantly encounter is dated material on the web. Stuff changes. What worked 10 years ago may not work today. Consequently, I will provide links to the most recent, relevant material that is closest to the source rather than incorporate instructions here.

I’m so old that I get free ski passes. Although I have been programming for many years, I am not up-to-speed when it comes to Python. My code tends to reflect the style when[*The Gambler*](https://www.youtube.com/watch?v=7hx4gdlfamo) was playing on the radio. I’m sure that the code could be improved but, like my intro said – the project represents what can be accomplished by someone “not versed in the art”.

The remote unit is programmed in the Arduino IDE – easy. Data is sent over WiFi to an MQTT broker running on the Pi. The Pi logs long term data in an SQLite database and presents fast changing data locally on screen. Long term data is plotted using matplotlib. The plots are sent via FTP to my personal website for viewing at a PC. The Pi is programmed in Python.

**Troubleshooting**

MQTT – in the source code of the ESP01 and Pi modules, you will see hive as an alternative broker. Using a cloud-based broker bypasses the need to setup MQTT on the Pi.

ESP – D1 mini

Pi system log – tail -f /var/log/syslog

sdfg