Brewer’s Eye – Final Report

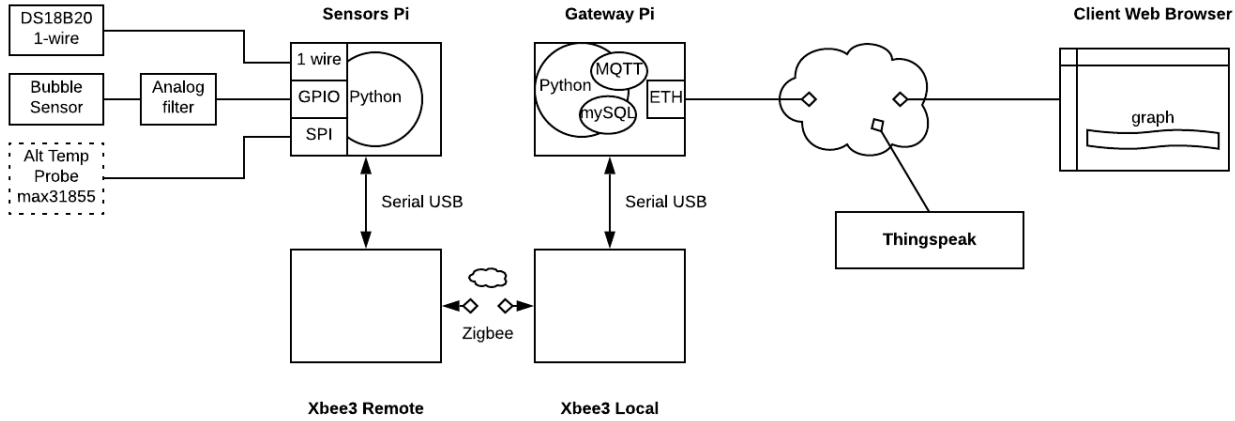
Micheal Burin
12/10/2019

https://github.com/burinm/BrewersEye

Table of Contents

Architecture

This is the block diagram for the Brewer's Eye connectivity and hardware. There is a remote sensor node that communicates to a gateway via the Zigbee protocol.



Zigbee

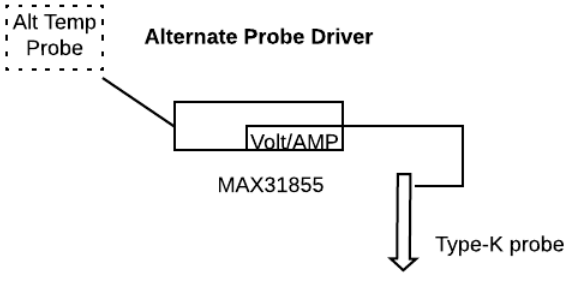
A custom message protocol is layered on top of this to make sure a whole message can fit in a one Zigbee packet. Message size is currently set to 60 bytes. This will avoid fragmentation if we are running in API mode (future enhancement). Currently the system is transmitting in Transparent mode. The custom message protocol includes a Node number so that future gateways can receive and parse messages from multiple sources.

Sensors

Currently the system is using 2 x DS18B20 (max38220) 1-wire temperature sensors. In the temperature range 10 – 45 degrees Celsius, they are accurate to +/- .5 degrees. The parts each come with a unique ID burned into the silicon which are currently coded hardcoded into the driver. They are currently using 3 wires (+, ground, data) for their connection. It is supposedly possible to use only the data/ground combination, but in testing this failed.

Type-K

An alternate option (stretch goal) was implemented that uses a type-K thermocouple probe as one of the temperature sensors. Type-K thermocouples work by utilizing the Seeback effect, which is a voltage difference between two dissimilar metals. In order to make a measurement, there needs to be a reference temperature and calibration. The max31855 driver takes care of this with an internal temperature sensor and outputs the calculation



digitally to SPI. I found in the end that this didn't work very well. 1) The max31855 only has a +/- 2 degree C accuracy, 2) The driver sometimes just outputted "0" 3) When connecting the probe at multiple junctions, I was creating more dissimilar metal junctions – which I suspect was affecting accuracy. T*his was disconnected for the final project in favor of using 2 max38220s instead*.

Thingspeak

Thingspeak (https://thingspeak.com/) is an online service that presents nice little graphs of IoT data you can upload to it. Brewer's Eye is using the Thingspeak library (paho also works here) to push MQTT messages to Thingspeak. Because the free version has a limited amount of data upload per year, the gateway is aggregating and averaging temperature/bubble messages before they are sent. Thingspeak then presents a neat public facing URL that presents graphs of the data.

Currently the public facing page https://thingspeak.com/channels/899531 is displaying Fermenter temperature, Ambient temperature and average bubble count. Additionally there is a custom "visualization" where all 3 are plotted together

Webclient

The webclient for Brewer's Eye is python's tornado implemented on the gateway. It serves up a web page that has a scrollable graph of collected sensor data. The gateway stores the received data in a mySQL database. Each sensor node has it's own database with in mySQL. (currently, there is only on, node88). The web client presents a graph/timeline for a sensor node. Graph data is loaded on demand from mySQL. This allows browsing the entire history of sensor data.

There is an alerts page where a user can subscribe to events. These are thresholds set on temperature and bubbles per minute. The alerts are issued in the form of an email sent with a SMTP client. The mail client is using Python's default smtplib and email libraries. There is an API hook in the tornado webserver that the web page uses to send alerts.

Analog Filter

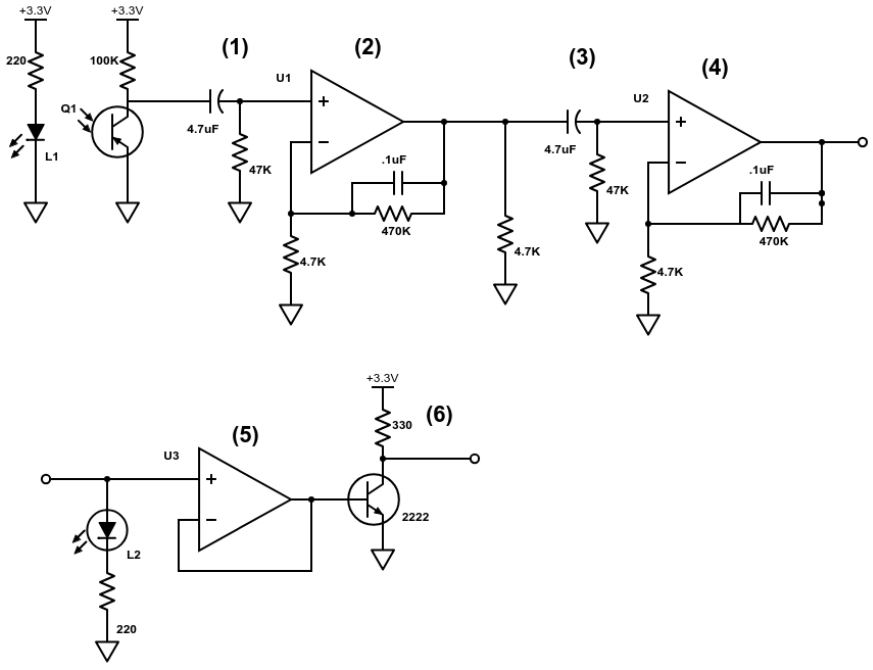
The analog filter takes input from an infrared sensor located inside the bubble chamber. One side of the chamber emits infrared light and the other side receives it. Bubbles change the intensity shining on the receiver. The filter essentially takes the derivative (changes in received light) of the input signal and runs it through a bandpass filter that only allows pulses between 295ms ↔ 1.4 seconds.

I borrowed a design from a pulse sensor here:
http://embedded-lab.com/blog/easy-pulse-version-1-1-sensor-overview-part-1/

Which I had worked with before for my IoT Low power project:

https://github.com/burinm/fatbit

This version is modified to use the off-the-shelf resistors and capacitors I had on hand. It also inverts the output, so there is an active low output for the Raspberry Pi. The inversion was necessary because the rail-to-rail output was not high enough to reliably drive the GPIOs. I substituted in a LM324N op amp.



I used this tool to draw the schematic: https://www.digikey.com/schemeit/

(1) Passive High-pass filter f = .72Hz, (1.4s) (2) Active Low-pass filter f = 3.4Hz, (295ms) (3) Passive High-pass filter f= .72Hz, (1.4s)	(4) Active Low-pass filter f = 3.4Hz, (295ms) (5) Unity gain (buffer for GPIO pins) (6) Invert signal

* f = 1/(2Pi * (R) * (C)) , high-pass C=4.7uF, R=47K; low-pass C=.1uF, R=470K

Project Deviation

Architecture components

Temperature sensor

The DHT22 temperature/humidity sensor was swapped out for the max32820 (DS18B20). Initially, the plan was to use 2 temperature sensors, a 1-wire device, and a type-K device. Because the two sensors reading didn't match closely enough (see architecture section above), it was decided to double down on the max32820 sensors. This also eliminated extra wiring and should draw less power for a future version of Brewer's Eye.

Thingspeak

The original design was to include a timeline graph and alert notifications driven from Thingspeak. After some initial research and implementation, it became apparent that Thingspeak was not the correct tool.

Unfortunately, customizing Thingspeak graphs proved to be difficult. This service's main focus is on smart data processing using a MATLAB backend. Any deviations from their graph templates means implementing this in MATLAB code. The public facing page can display what they call "visualizations", but any custom plugins are stuck on a private, non-public page. At this point, I decided that a custom html/javascript/python implementation would be more appropriate.

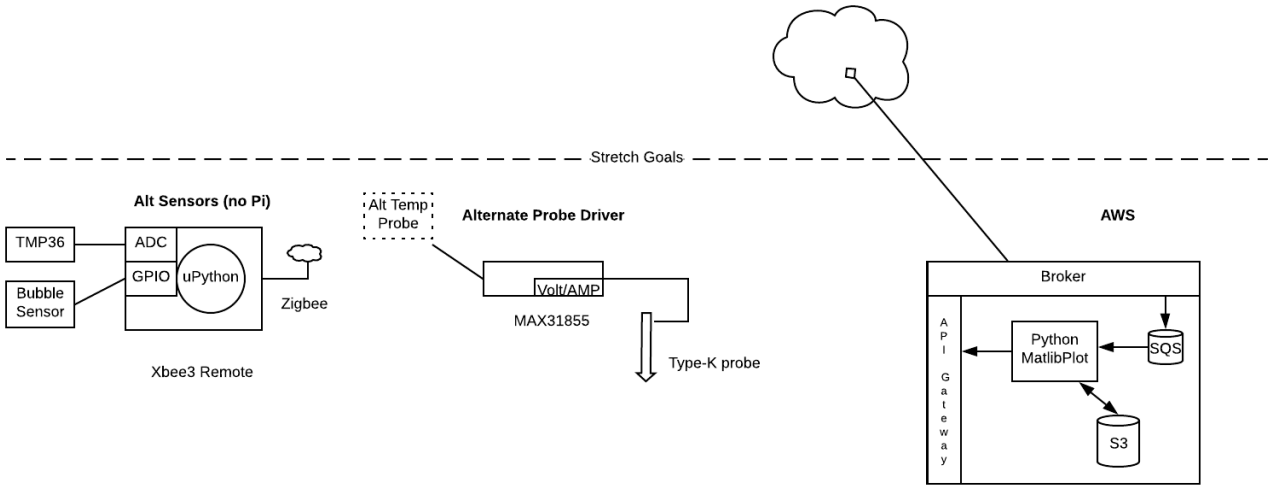
Thingspeak was useful for:
1) A highly-available, zero-administration, reachable-from-anywhere portal to an IoT project
2) To get the most up to date data from a gateway's sensors,
3) As a sanity check to make sure the system is still up and running.

After abandoning Thingspeak, I didn't look into driving alert notifications. There is apparently a IFTTT (If this, then that) tool that can hook into web services. So, I believe this would have been possible.

Stretch Goals

Type-K

I'm glad that these were intended to be stretch goals only. There was only enough time to look into the type-K temperature sensor as described above.



Alt Sensors / AWS

The other goals are intended to continue the evolution of Brewer's Eye. I'd like to develop this idea more outside of class:

1) TMP36 is an analog sensor. The idea here is that the Zigbee, which has an ADC, could be the entire sensor board. There would be no need for a Raspberry PI on the sensor array, and the whole sensor unit becomes more power efficient, physically smaller and cheaper to build.

2) The AWS research was to be an extension to what we learned in class. Specifically, I never tried out AWS to serve up external web APIs (only used SQS). It would be nice to have a high-availability web service interface to Brewer's Eye. This way, the gateway is relieved from doing "heavy lifting". The user is relieved from doing system administration. The gateway could then use a smaller micro-controller and be more power/space friendly.

3) There is an API mode on the Zigbee where the programmer manipulates the packets directly. This again extends the idea that the sensor array would run just on a Zigbee board. (Currently the UART/USB interface is used on the Pi to take advantage of Zigbee's transparent mode). This would also enable using multiple sensor arrays at the same time to the same gateway. I have to confirm it, but I'm pretty sure that transparent mode breaks up packets across transmissions. My current message protocol relies on full messages being intact. Multiple nodes would potentially intermix packets when trying to read from multiple nodes at once, so they have to be atomic. Implementing API mode would fix this. (My current protocol implementation already keeps sensor messages small enough to be stuffed in one packet)

Third Party Code

All snipits, templates and how-to code used from sources on the web are documented with comments in the code.

The following are third party packages used in Brewer's Eye:

CircuitPython components

Components from Adafruit's uPython like implementation

https://circuitpython.org/
board - Small part of circuit python that abstracts pin/port names
busio – Used to talk to SPI devices on the Pi
digitalio – Setup GPIO pins

Rpi.GPIO

Read, Write, and get events from Raspberry Pi GPIO pins
https://pypi.org/project/RPi.GPIO/

CircularBuffer.py

A python implementation of the circular buffer described in Making Embedded Systems, Elecia White, (C) 2012

Ported from my C version (2016):
https://github.com/burinm/drivers/blob/master/mylib/circbuf_tiny.h
https://github.com/burinm/drivers/blob/master/mylib/circbuf_tiny.c

MySQLdb

Access mysql-like datebases with python
https://pypi.org/project/MySQL-python/

adafruit_max31855

Read from type-K temperature interfaces
https://github.com/adafruit/Adafruit-MAX31855-library

max31820.py

Code that uses the 1-wire linux file system interface to read the max31820
https://learn.adafruit.com/adafruits-raspberry-pi-lesson-11-ds18b20-temperature-sensing
https://learn.adafruit.com/adafruits-raspberry-pi-lesson-11-ds18b20-temperature-sensing/software

thingspeak

Update and send data to Thingspeak service
https://pypi.org/project/thingspeak/

pyserial

Use serial port devices on the Raspberry Pi
https://pypi.org/project/pyserial/

paho

Test MQTT client to upload to Thingspeak
https://pypi.org/project/paho-mqtt/

typing, functools

(integrated into python) Markup functions and interfaces with types

json

(integrated into python) Encode and decode data structures/JSON

tornado

Web server for python
https://www.tornadoweb.org/en/stable/

smtplib, email

(integrated in python) Send emails with SMTP server
https://docs.python.org/3/library/email.examples.html#email-examples

jquery

Javascript library to fetch web APIs
https://jquery.com/
https://github.com/burinm/jquery-3.4.1 – My local copy

vis

Javascript library to create 2d graphs and timelines
https://visjs.org/
https://github.com/burinm/vis-6.2.9 – My local build

Project Observations

Timeline Graph

The development of the timeline graph went better than expected. Once I abandoned Thingspeak for this purpose, I had a plethora of Javascript libraries to choose from. For future use, react-vis, by Uber, looks fun and interesting. https://uber.github.io/react-vis/

Vanilla vis has some bugs, but I was still able to accomplish my goal of being able to browse through my entire sensor dataset. It made implementing the scrolling and zooming portion I needed for the graph a snap.

I had to implement the on-demand caching for the web server database queries, but I think this would be necessary for any library. There is still more work to be done here (i.e. - purge entries in cache to keep page running smoothly)

The two bugs I had here were:

1) Graph left side y-axis scale display broken.

2) (enhancement) Vis can't display a timeline and a graph2d on top of each other.

To get around this, I used both a timeline and a graph2d object and sent events between the two to keep them updated

Xbee

The Xbees implementing Zibgee were especially pleasant to work with. The initial setup had to be done via a special program by Digi (xctu) https://www.digi.com/products/embedded-systems/digi-xbee/digi-xbee-tools/xctu. This worked great on linux, and the setup was a snap.

Running Xbees in Transparent mode across serial ports on the Raspberry Pi turned out to be quite easy. Setup a unique address for each, set addresses as destination/source, and setup PAN IDs. When the PAN IDs (personal area network ID) are set the same, the Xbees were able to send packets in Transparent Mode. After setup, I tested by simply opening two serial consoles, one per Xbee, and typed text in each.

Bubble Sensor

The bubble sensor hardware was a fun but challenging portion of this project. I faced the following issues:

1) Tuning the analog filter to the correct sensitivity

The original circuit is derived from an op-amp filter chain that is meant to measure a heartbeat by reflected/absorbed infrared light in human skin. I needed to adjust the intensity of the transmitter LED and the pull down resistor in the middle of the filter chain to not saturate the signal.

One bonus of fixing up the filter was that *I was able to remove the debounce code in the driver.* The filter made nice enough peaks and troughs to count without a digital filter.

Also, I had originally wired one of the stages incorrectly on my breadboard. (Cause of initial demo failure in the lab)

2) Prevent infra-red light leaks into the system

My first designs used transmissive material (clear acrylic plastic rods) to transfer the transmitter and receiver signals into the bubble chamber. It turns out (since they are not perfect like fibre-optic cables), they leaked light from outside sources. In particular, infra-red from ambient sunlight would set off the detector. Since the detector is measuring the rate of change of infrared light, just walking by the sensor would set it off.

3) Controlling the rate of gas escaping from the air-lock

The chamber I used as an airlock was meant to be an opaque version of a pint glass filled with water. Normally a tube from the fermentor is stuck into the glass so that the opening of the tube is touching the bottom of the glass, underneath the liquid.

My design, which puts the air inlet into the bottom of the chamber proved problematic. It created a loop in the tubing which held the escaping gas. This caused larger than usual bubbles to build up. Instead of having nice tiny bubbles that would be easy to count , the whole system would burp all at once. This destroyed the fine grain data I was hoping to collect for analysis.

There is glue code that averages out the bubble count for the time being.