About the code

There are three files:

- OTATelnet_Blynk_PCB_UA_4-31.ino, the main file.
- OTA_Telnet_2.h, for the communications components
- creds.h, the credentials file.

Main file

The settings for the triggers are gathered early on.

```
//// this section for the triggers >>>>>>>>>>>>>>>>>>>>>>>>>>.
const float ramp = 4.0; // a sudden rise in temp will start the pump. Degrees C.
float minTemp = 45.0; // for getting the minimum of an array of temp values. Set it high to
start. ramp will compare with this.
const float maxTemp = 50.0; // startup temp
const int gasTrigr = 4000; // this value will set off the system
const int gasTrigrLowr = 2500; // this value will set off the system in combination with
elevated temp.
```

These constants allow you to vary the number of hose-lines you will be pumping, and the duration of water delivery for each hose-line.

```
const byte numLines = 7; // the number of water delivery pumping lines connected to the
relays
const byte lineTime = 60; // how long each water line will pump for in seconds
```

>>>

The lineTime constant will work between 10 seconds and 255 seconds. For testing, I suggest 10 seconds. For practical use, 60 seems right for my house. If your local fire weather warning is for 20 plus C but high winds with very dry vegetation, plus existing fires in the area, you might lower minTemp and maxTemp. I set them high initially to avoid false starts and wasting scarce water.

If your situation requires only, say, four water lines, change the constant numLines appropriately.

```
const int c = 35; // y=mx+c equation for temp sensor calibration
const float m = 14.6; // y=mx+c equation for temp sensor calibration
```

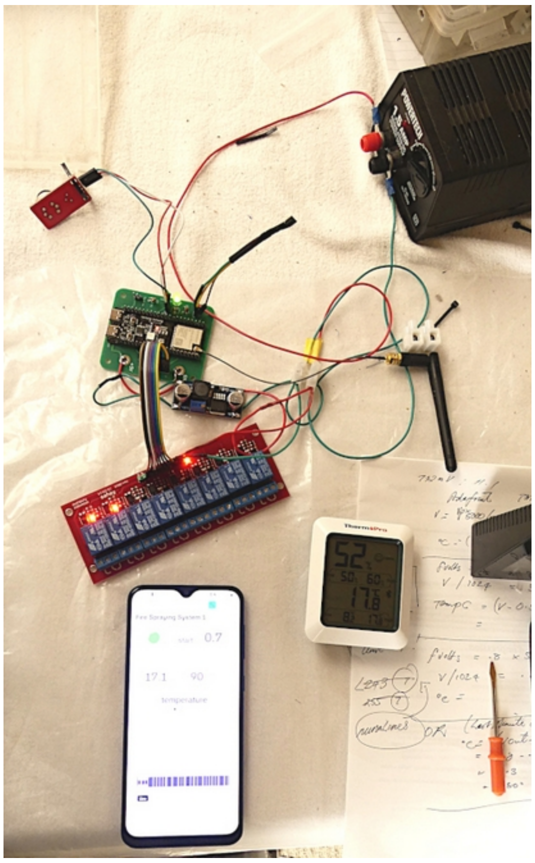
I have tried various temperature sensors and, in spite of the claims for some, they need calibration. The TMP36 sensor response is linear, although I have seen some log equations used. I set up a spreadsheet to plug in the voltage readings (float TMP36_mV in the code) and the temperature readings from a Thermopro TP350 temperature and humidity unit which should be accurate to 0.5 degrees C. A couple of coordinates are enough to calculate the equation. While this isn't laboratory accurate it's much closer than the specs settings. I have used 500 mV offset, from the specifications sheet. Next time I will connect the TMP36 to the 3.3v pin of the ESP-32, rather than the 5v I have now. My readings should then come in line with the specifications.

The function for temperature reading and averaging is

```
void checkSensorsAve() {
```

```
for (int j = 1; j <= numTempReads && analogRead(tempPin) > 0; j++) {
thisTempRead = (analogRead(tempPin));
tempReadBucket = tempReadBucket + thisTempRead;
}
```

numTempReads is set to 10. I have found that the sensor is capable of giving 500 or more readings each second, which one would think should give a better average. However the more reads, the more dropped reads, particularly if the unit is busy trying to connect to its communication sub-systems. Ten reads only each second gives a very stable result and leaves the remainder of the second for other processing activities. analogRead(tempPin) > 0 requires that a read can't be empty – which would upset the average.

	Testing during development: • checking the pumping duration and sequence • checking the Blynk information was working and the start button functioned • checking temperature calibration

I found the processor seems to be most stressed when the internet is patchy – quite likely in fire conditions. A number of programmers have suggested restarting the processor to give the best chance of reconnection. For this reason I built a counter to measure wifi downtime, and after 20 minutes (1200 seconds), the system restarts.

```
void restart() {
if ((downDecades > 120) && (started == 0)) ESP.restart();
}
```

The disadvantage of a restart is the loss of pumping cycle information. It's very useful to know how many times the unit has been triggered.

The pumping function, void pumping(), works through activating the relays for the pump and the water lines. The water line opens two seconds before the pump starts, so as not to stress the lines. The pump stops two seconds before the line closes. Then the next line sequence begins.

The float lineNowPumping is for checking where the pumping sequence is up to.
The float numPumpCycles shows the completed cycles.
Since I was limited with the number of variables I could use in Blynk, I combined these into what looks like a decimal number but isn't. On the phone app, Cycles/line of, say, 2.7, indicates there has been two completed cycles and the unit is pumping on line 7. When that last line is complete the number clicks over to 3.

creds.h

The credentials file isn't included because it's confidential. That's the idea.
Yours should look like this (keep the quotation marks):

```
const char* MYssid = "YourWiFiName";
const char* MYpassword = "YourWiFiPassword";

// Blynk
#define BLYNK_TEMPLATE_ID "YourBlynkTemplateXXXXXYYYYZZZ"
#define BLYNK_TEMPLATE_NAME "Fire spray system"
#define BLYNK_AUTH_TOKEN "YourBlynkToken"
```

>>>

GDM Tasmania 2024.