Promethean Temperature Sensor

Joe Collins

December 3, 2020

Contents

1 Objective

A temperature sensor that can be scraped/polled using Prometheus.

Prometheus is excellent for monitoring server metrics, so it makes sense to use it to monitor other metrics such as temperature and humidity. There doesn't appear to be anything available off the shelf. There are temperature monitors but some effort would be required to get them to integrate with Prometheus. So we might as well build a bespoke temperature sensor that can be scraped/polled directly by Prometheus.

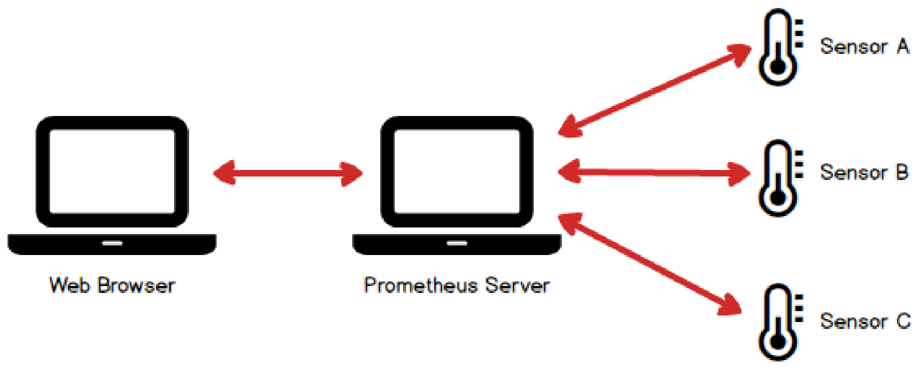

Figure 1: Prometheus scraping sensors

With Prometheus scraping the temperature sensors, a web browser can be used to few graphs of the scraped data.

2 Components

Each sensor is an Arduino microcontroller with a temperature sensore attached, total cost around £40.

Component	**Cost**
Arduino Uno Rev3, ATmega328P, CH340G Compatible Board	£5.79
UK 9V AC/DC Power Supply Adapter Plug for Arduino Uno	£7.95
Ethernet Shield LAN W5100 for Arduino Uno	£7.75
DHT22 AM2302 Digital Temperature and Humidity Sensor	£6.90
0.25 Watt Metal Film Resistor 10k Ohm	£0.99
Uno Ethernet Shield Case	£10.36
Total cost in November 2020	**£39.74**

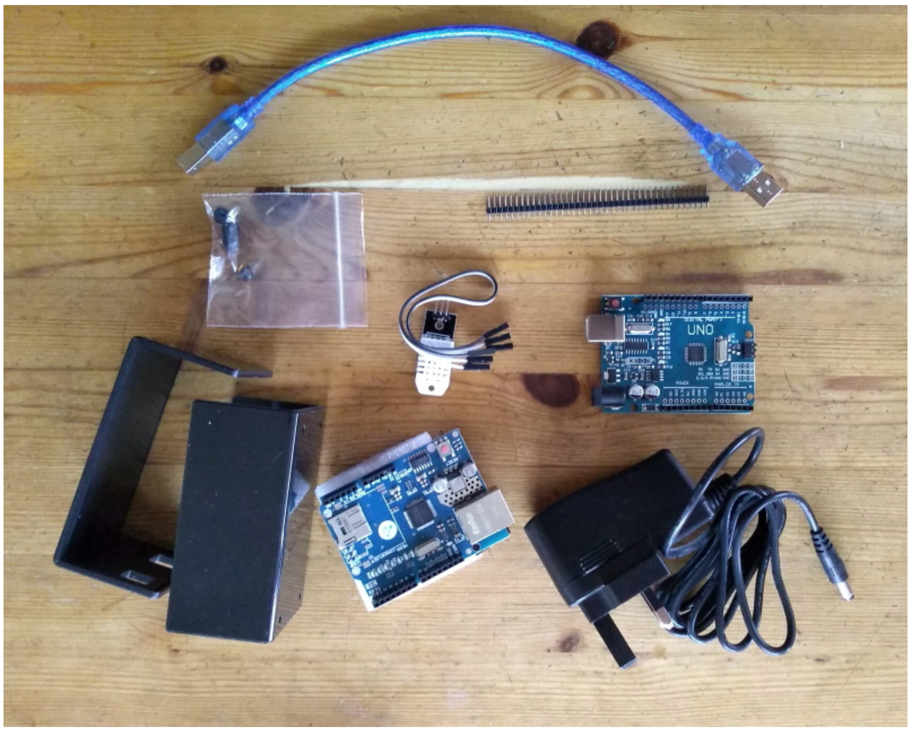

Figure 2: Components

3 Wiring the Sensor

The sensor is a DHT22 AM2302 capacitive humidity sensing digital temperature and humidity module. It is has a calibrated digital signal output of the temperature and humidity sensors. Other sensors are available such as the cheaper DHT11, but supposedly it is less sensitive and less durable. Whilst sensitivity is not important durability probably is.

The sensor is wired with a 10k Ohm pull up resistor. This ensures that the signal wire has a small but consistent current, so it is less susceptible to electrical interference. In theory you can also set the pin mode with `pinMode("pin", INPUT_PULLUP);` to use a built in pull up resistor.

The sensor works without the pull up resistor but is probably less accurate (though I haven't tested it).

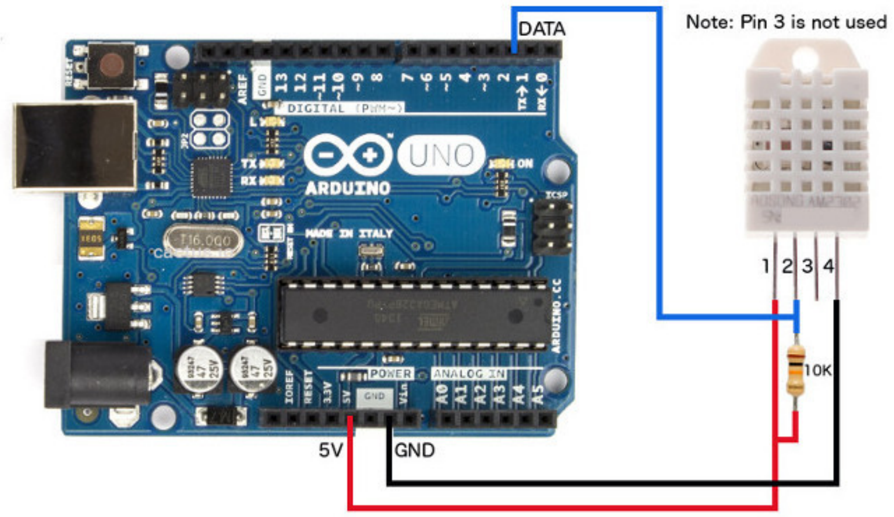

Figure 3: Wiring diagram with pull up resistor

Since the Arduino has an ethernet shield, practically the wiring looks like this with insulation around the connectors in case they get moved.

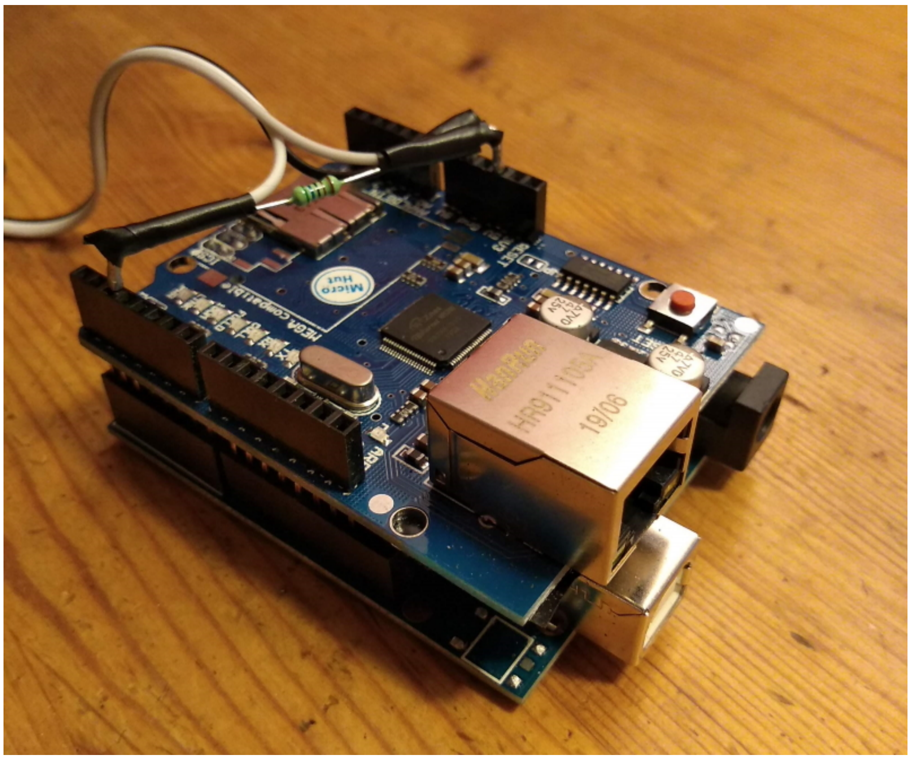

Figure 4: Wiring with insulation

4 Programming

Tooling:

- VSCode
  https://code.visualstudio.com/
- PlatformIO
  https://marketplace.visualstudio.com/items?itemName=platformio.platformio-ide

All Arduino programs have the same format with `setup()` and `loop()` functions. The `loop()` runs continuously and the `setup()` is run when the Arduino is turned on or when the reset button (red button near the USB socket) is pressed.

"../src/main.cpp" 5a≡

```
#include <Arduino.h>
⟨ libraries 5b, ... ⟩
/*** Configuration ***/
⟨ configuration 6a, ... ⟩
/*** Functions ***/
⟨ functions 6c, ... ⟩
/*** Setup ***/
void setup()
{
  Serial.begin(9600);
  while (!Serial){;}
⟨ setup 6b, ... ⟩
}
/*** Loop ***/
void loop()
{
⟨ loop ?? ⟩
}
◇
```

Since we're using PlatformIO we need the Arduino header (`<Arduino.h>`). For normal Arduino programs it is not required.

It's useful to be able to connect to the Arduino over the USB cable, so the `setup()` opens serial for output to monitor and waits for the port to open. Waiting for the serial port to connect is only needed for native USB ports.

4.1 Sensor

To interact with the DHT22 sensor we need specific drivers (`<DHT.h>`), which rely on the Adafruit Unified Sensor Driver (`<Adafruit_Sensor.h>`), The Adafruit Unified Sensor Driver is an abstraction layer which makes creating reliable drivers is easier.

The Serial Peripheral Interface (`<SPI.h>`) is a synchronous serial data protocol used by microcontrollers for communicating with peripheral devices quickly over short distances.

⟨ *libraries* 5b ⟩ ≡

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <SPI.h>
◇
```

Fragment defined by 5b, 7b.
Fragment referenced in 5a.

The sensor communicates using pin 2 (see wiring) and the sensor type needs to be set. The temperature and humidity are stored in global variables since they'll be used by pretty much every part of the program.

⟨ *configuration* 6a ⟩ ≡
```
#define DHTPIN 2
#define DHTTYPE DHT22
DHT dht = DHT(DHTPIN, DHTTYPE);
float temperature = 0;
float humidity = 0;
◇
```
Fragment defined by 6a, 7c.
Fragment referenced in 5a.

The sensor needs to `begin()`. Formerly this method was used to pass in parameters relating the the speed of the Arduino. Now the sensor automatically adapts but `begin()` is still needed.

⟨ *setup* 6b ⟩ ≡
```
Serial.println("Set␣up␣sensor");
dht.begin();
◇
```
Fragment defined by 6b, 8ab.
Fragment referenced in 5a.

Set the values of humidity (as a percentage) and the temperature (as Celsius) as globals. This is largely so I don't have to bother passing the values around. In the event that nothing comes back from the sensor, the values of `humidity` and `temperature` won't get reset. So if the DHT22 sensor fails in use the device will carry on reporting old (but plausible) values until the Arduino is reset.

⟨ *functions* 6c ⟩ ≡
```
void readSensor()
{
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();
  if (isnan(humidity) || isnan(temperature))
  {
    Serial.println("Failed␣to␣read␣from␣sensor");
    return;
  }
}
◇
```
Fragment defined by 6c, 7a, 8c, 9a.
Fragment referenced in 5a.

In use there is no need to write out to the serial monitor, since the USB cable will not be attached. However, during development it is handy to be able to see what is going on.

⟨ *functions* 7a ⟩ ≡

```
void serialPrintReadings()
{
  Serial.print("Humidity:␣");
  Serial.print(humidity);
  Serial.print("␣%␣|␣");
  Serial.print("Temperature:␣");
  Serial.print(temperature);
  Serial.println("␣C");
}
◇
```

Fragment defined by 6c, 7a, 8c, 9a.
Fragment referenced in 5a.

4.2 Ethernet Client

The goal is to provide an HTTP end point for Prometheus to scrape data from. This is possible using just `<Ethernet.h>` (the library for the ethernet) but this approach only allows for one web page and no routing. Typically Prometheus expects the metrics to be on a route at `/metrics`. Also having routing would give the possibility of having a more friendly webpage showing the readings.

The Arduino web server libraryArduino web server library. The package downloaded by PlatformIO is marked `lasselukkari/aWOT@0.0.0-alpha+sha.bf07e6371c` which is curious since the rest of the meta data indicates that it is version 3.0.2 (the current latest version).

⟨ *libraries* 7b ⟩ ≡

```
#include <Ethernet.h>
#include <aWOT.h>
◇
```

Fragment defined by 5b, 7b.
Fragment referenced in 5a.

The MAC address and IP address for the device are provided as global values. This isn't strictly necessary since they are only used once in the `setup()` but this is the pattern followed by most examples I found so I did the same. However it is useful for the `server` and the `app` to be available as global variables since the `loop()` makes repeated use of them.

⟨ *configuration* 7c ⟩ ≡

```
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(10, 0, 21, 212);
EthernetServer server(80);
Application app;
◇
```

Fragment defined by 6a, 7c.
Fragment referenced in 5a.

The MAC address and IP address are then used in the `setup()` to initialize the ethernet shield and confirm that it is present and working.

⟨ *setup* 8a ⟩ ≡

```
Ethernet.begin(mac, ip);
if (Ethernet.hardwareStatus() == EthernetNoHardware)
{
  Serial.println("Ethernet␣shield␣was␣not␣found.");
  while (true)
  {
    delay(1); //Do nothing since there is no point in proceeding.
  }
}
if (Ethernet.linkStatus() == LinkOFF) {
  Serial.println("Ethernet␣cable␣is␣not␣connected.");
}
◇
```

Fragment defined by 6b, 8ab.
Fragment referenced in 5a.

Two routes are provided, one to '/' for a web page and one to '/metrics' for the Prometheus metrics.

⟨ *setup* 8b ⟩ ≡

```
app.get("/", &indexCmd);
app.get("/metrics", &metricsCmd);
◇
```

Fragment defined by 6b, 8ab.
Fragment referenced in 5a.

The web page is as simple as can be but refreshes at regular intervals to show the temperature.

⟨ *functions* 8c ⟩ ≡

```
void indexCmd(Request &req, Response &res)
{
  Serial.println("Request␣for␣index");
  res.set("Content-Type", "text/html");
  res.println("<html>");
  res.println("<head>");
  res.println("␣␣<meta␣http-equiv=\"refresh\"␣content=\"5\">");
  res.println("</head>");
  res.println("<body>");
  res.println("␣␣<H1>Temp:␣"+ String(temperature) + "</p>");
  res.println("</body>");
  res.println("</html>");
}
◇
```

Fragment defined by 6c, 7a, 8c, 9a.
Fragment referenced in 5a.

Prometheus has a particular style for showing metrics ready for scraping. In this case both humidity and temperature are gauges, values that vary. For interest the alternative Prometheus metric type is a counter, for values that increase.

⟨ *functions* 9a ⟩ ≡
```
void metricsCmd(Request &req, Response &res)
{
  Serial.println("Request␣for␣metrics");
  res.set("Content-Type", "text/plain");
  res.print("#␣HELP␣temperature␣is␣the␣last␣temperature␣reading␣in␣degrees␣celsius\n");
  res.print("#␣TYPE␣temp␣gauge\n");
  res.print("temperature␣"+ String(temperature) + "\n");
  res.print("#␣HELP␣humidity␣is␣the␣last␣relative␣humidity␣reading␣as␣a␣percentage\n");
  res.print("#␣TYPE␣humidity␣gauge\n");
  res.print("humidity␣"+ String(humidity) + "\n");
}
```
◇
Fragment defined by 6c, 7a, 8c, 9a.
Fragment referenced in 5a.

Most of the time the Arduino is running through this loop. It takes about 250 milliseconds to read from the sensor. The 2 second delay is to prevent reading from the sensor faster than it can respond. This does mean that a request from Prometheus for data might be delayed by up to 2 seconds before the web server can respond.

⟨ *loop* ?? ⟩ ≡
```
  delay(2000);
  readSensor();
  serialPrintReadings();
  EthernetClient client = server.available();
  if (client.connected()) {
    app.process(&client);
    client.stop();
  }
```
◇
Fragment referenced in 5a.

4.3 Build and Upload

The PlatformIO extension with VSCode includes commands to `PlatformIO: Build` and `PlatformIO: Upload`. Initially the upload did not work so I installed CH340 drivers on the advice of this reference https://nagesh-uluvar.blogspot.com/2016/12/arduino-not-getting-detected-usb20.html. The short answer is to install these drivers https://sites.google.com/site/nageshuluvar/home/DRIVER1_CH340.zip?attredirects=0&d=1. There are versions at https://sparks.gogo.co.nz/ch340.html for the Mac as well.

Additionally the upload was sometimes unreliable. It appears that PlatformIO attempts to read back the uploaded file. The read back doesn't always work, but more often that not the file had been successfully uploaded to the Arduino.

4.4 Testing

Once the files have been uploaded to the Arduino, it should restart and the serial monitor should show something like this.

```
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Webserver set up
Sensor set up
```

```
Humidity: 55.90 % | Temperature: 22.10 C
Humidity: 56.30 % | Temperature: 22.20 C
```

To test the Prometheus end point you can use an ethernet patch cable to connect to it directly (rather than via a router).

- Assign a manual IP address to the laptop's ethernet connection say 10.0.21.1.
- Subnet mask 255.255.255.0.
- Assign a manual IP address to the Arduino's ethernet, say 10.0.21.212.
- Subnet mask 255.255.255.0.
- Leave the default Gateway empty.
- Use an ethernet patch cable to link the two (since 100BaseT onwards it doesn't have to be a special cross over cable).
- You should then be able to get your Arduino site up on `http://192.168.0.2` from the laptop.

This is the endpoint at `http://10.0.21.212/metrics` which should show something like this.

```
> curl 10.0.21.212
# HELP temperature is the last temperature reading in degrees celsius
# TYPE temp gauge
temperature 23.30
# HELP humidity is the last relative humidity reading as a percentage
# TYPE humidity gauge
humidity 47.60
```

The Prometheus server is available as a docker image, so you can run it up with `docker-compose up`. The Prometheus interface should be available at `http://localhost:9090` and should be able to provide a graph like this.

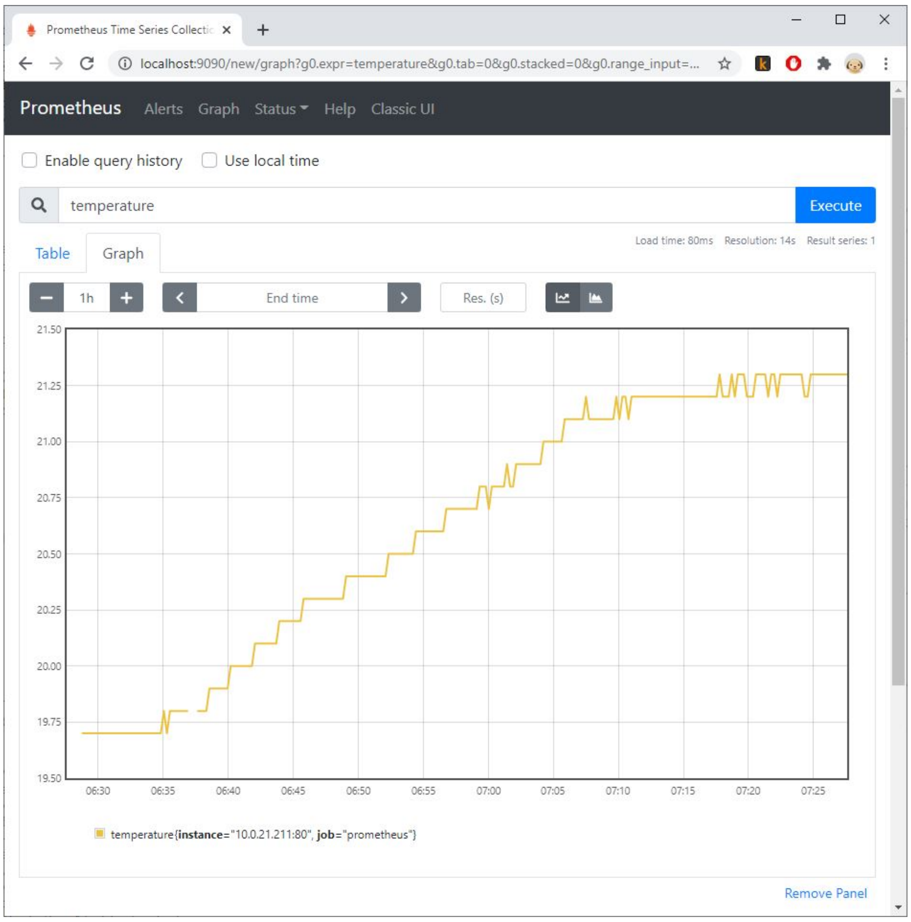

Figure 5: Temperature graph

5 Packaging

The Arduino does put out a bit of heat, so it doesn't make sense to mount the temperature sensor directly on the case. Hence it is mounted on a milk bottle top. The cable tie is 2.5 mm wide so the holes are 2.5 mm in diameter. The hold for the wires has a 4mm diameter.

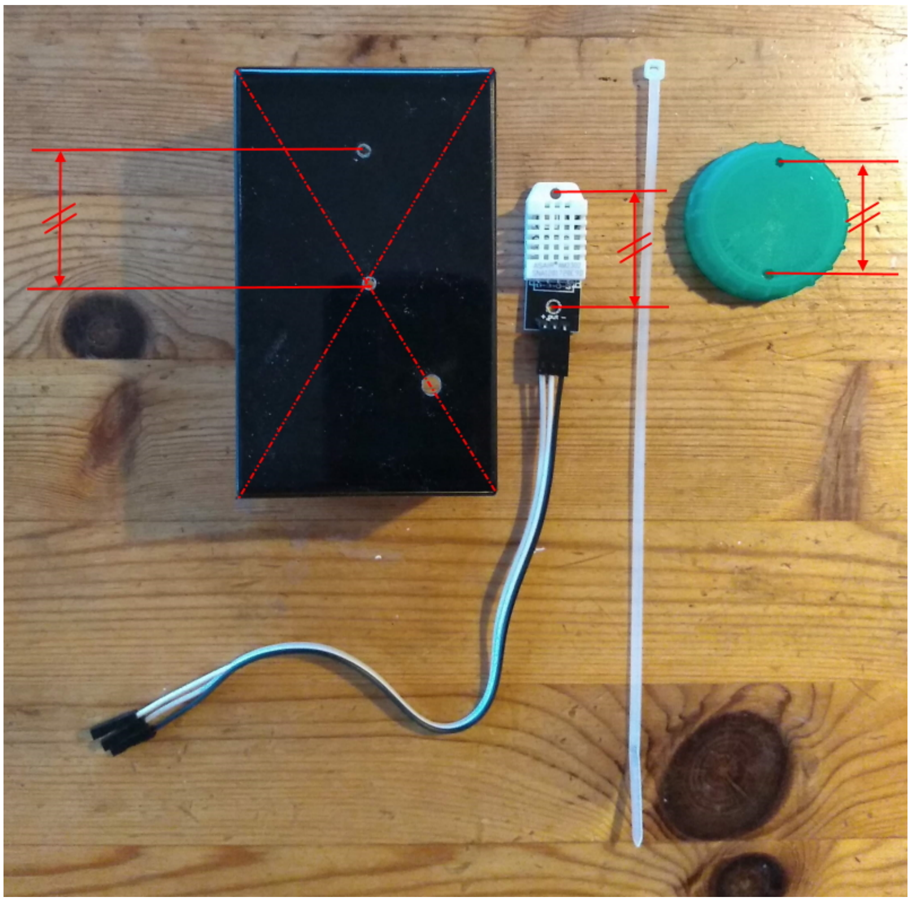

Figure 6: Components for mounting the sensor

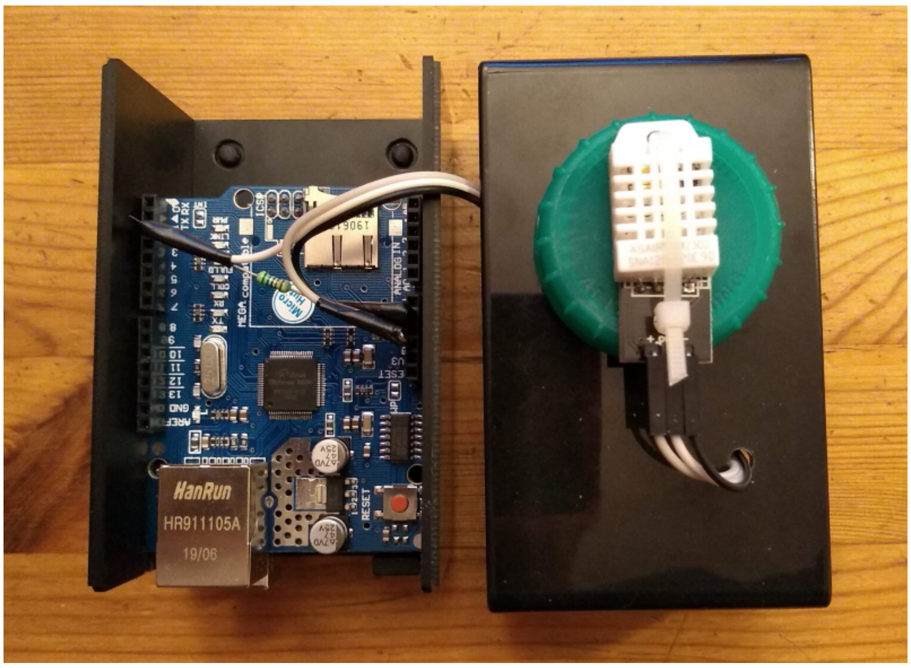

Figure 7: Mounted sensor and wiring