# Open Ruche project

The project we are working on is a long-standing initiative. Its origins are rooted in the desire to make life easier for beekeepers, to help them preserve and take care of their little protected. Indeed, beekeeping and bees in general are among the most recently affected by globalization and global warming. These are for example the cause of the arrival of Asian hornets as well as verroas, small mites attacking our favorite honey makers. Changes in climate and temperature also play a major role in whether or not liquid gold is well made. The connected hives that we develop as a class project are intended to overcome most of these problems, via innovative solutions that we will detail in more detail in the following lines. The purpose of this document is to allow anyone with good intentions and a minimum of equipment to be able to reproduce without too much difficulty the solution that we have developed in our training. Let's start.

## The beginnings of the project

As you may know, bees and their guardians have been in trouble for some time, for several different reasons. However, instead of setting out new problems or trying in vain to establish an exhaustive list of them, let us instead separate them into categories. And because here we are here to move things forward in a positive way, let's see how a solution can solve several problems at once.

The connected hive is actually a housing connected to several sensors strategically placed around the hive as well as within it. This box, which collects the data of the various sensors, will then proceed to send them to a data site (Sigfox), which will itself return thein order to make the information relating to the hive of interest readable in the form of tables in particular. Some information, such as the state of the system battery if its level is critical, will be sent directly by email to the beekeeper.

A hive is regularly subject to various changes. Some are daily and quite natural while others, as you may have guessed, are not. The temperature is, for example, extremely important in the production of honey and the maintenance of the good health of bees. Indeed, a certain temperature must be maintained in the hive in order to allow their residents to produce the food so appreciated by the French. It is normally regulated by our striped girlfriends, but last summer, temperatures reached such a level that the poor found themselves totally destitute. Worse still, they were so high that the outside temperature became a crucial parameter to harvest. In some places, the thresholds reached have gone so far as to melt nearly 80 hives, thereby burying their unfortunate occupants during this period alone.

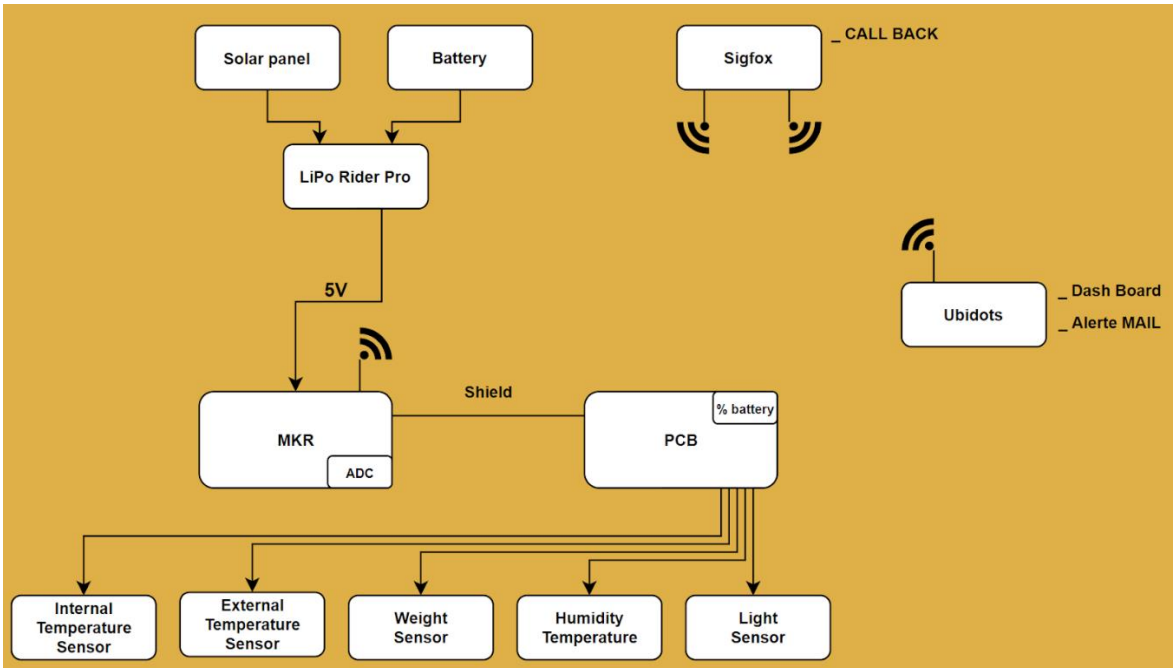
Another condition normally regulated by bees is humidity, which also plays an important role in, as you may have guessed, the production of honey. Too high a moisture content in a hive can lead to poor drying of honey, which is a major step in the production of honey. A survey of excessive values should therefore alert beekeepers to a change in unusual conditions.

The weight of a hive is also constantly changing. Despite this, it follows a logical and perfectly explainable cycle. This means that if the weight comes out of this pattern, we are facing an anomaly. Changes are made as the bees move inside and out of the hive. At night, they are all in the latter, while during the day, the departure of the foragers and their returns influence the fluctuations of its weight. A weight taking a zero value abruptly can also be the sign of a fall or a theft, a real problem in the field, the houses of bees selling at gold price on the black market.

Once the various issues are clearly explained, the next step is to find a way to answer them correctly.

## An accessible project

The foundations are now laid. The project remains to be implemented. First, a functional diagram of the project is required to provide an overview of the components, their functions, and their layout.
So it is on this diagram that we will be able to see the MKR FOX 1200 Arduino board, and how it interacts with other parts of the system and retrieves the data to send to the user.




We will therefore supply the MKR FOX 1200 board with electricity using the Lipo Rider Pro adapter, a cost-effective solution to power a device with the help of a solar panel and a battery (in case the sun should be absent). The supplied Arduino will collect data from the internal and external temperature and humidity sensors, the weight sensor, as well as the light sensor via the shield-connected PCB. The MKR will then send the received data to Sigfox, a French network of antennas dedicated to the Internet of Things (IoT), which will itself send them to Ubidots using a Call Back function. Ubidots will therefore be responsible for interpreting the data and putting it in the form of tables, or any other more intuitive form than raw figures. If there is an anomaly, an email will be sent to the beekeeper who owns the hive to alert him and allow him to intervene as quickly as possible. The realization remains to be seen.

## Coding when you hold us

As you can imagine, each sensor will need a specific code. They are the ones who, developed in Arduino language, will allow the latter to raise the values of the sensors and send them to Sigfox. Without further ado, let's see those codes together.

```
#include "HX711.h"

// HX711 circuit wiring
const int DOUT = 2;
const int PSCK = 3;

HX711 scale;

float weight; // Weight
float calibration_factor = 20900;
float zero_factor = 777787;

void setup() {
  Serial.begin(115200);

  Serial.println("Initialisation de la balance...");

  scale.begin(DOUT, PSCK);

  scale.set_scale(calibration_factor); // Adjust to this calibration factor
  scale.set_offset(zero_factor); // Adjust to this zero factor
  // scale.tare(); // Reset the scale to 0

  Serial.println("La balance est prete!");

}

void readSensors() {
  weight = scale.get_units();

  if(weight<0) {
    weight=0;
  }

  Serial.print(weight, 2);
  Serial.println(" kg");
  Serial.print(" calibration_factor: ");
  Serial.print(calibration_factor);
  Serial.print(" zero_factor: ");
  Serial.println(zero_factor);
}

void loop() {
  scale.power_up();
  readSensors();
  scale.power_down();
  delay(2000);
}
```

Above, the scale code, in other words the weight sensor. It starts by calibrating the interested party using values found through another code. Actually, we are not going to see all the codes one by one. Once all developed, another will be written as "One Wire". As its name suggests, it allows all information shipments to be grouped together in a single thread and, here, in a single code.

```
#include <SigFox.h>
#include <ArduinoLowPower.h>
#include <OneWire.h> //DS18B20 Intern temp
#include "HX711.h" //scale HX711
#include "DHT.h" //DHT22 Extern humidity and temp

//Struct to send values to Sigfox
typedef struct __attribute__ ((packed)) sigfox_message {
 int16_t temp0;
 int16_t temp1;
 int16_t temp2;
 int16_t weight;
 int8_t tempDHT;
 int8_t humidity;
 int8_t etatBattery;
 int8_t humidityInside;
} SigfoxMessage;
// stub for message which will be sent
SigfoxMessage msg;
// ================== UTILITIES ==================
void reboot() {
 NVIC_SystemReset();
 while (1);
}
// ================================================================================ DS18B20 begin
 /*
  * Capteur DS18B20
  */
```

```
/* 1-Wire bus pin */
const byte BROCHE_ONEWIRE = 5;
/* Sensors address */
const byte SENSOR_ADDRESS_1[] = { 0x28,  0xB0,  0x26,  0x35,  0xC,  0x0,  0x0,  0xF};
const byte SENSOR_ADDRESS_2[] = { 0x28,  0xBA,  0x39,  0x53,  0xA,  0x0,  0x0,  0x9B};
const byte SENSOR_ADDRESS_3[] = { 0x28,  0x5,  0x59,  0x53,  0xA,  0x0,  0x0,  0xA9};
/* Create 1-wire object to use */
OneWire ds(BROCHE_ONEWIRE);
/**
 * Function to read temperature of DS18B20 sensor.
 */
float getTemperature(const byte addr[]) {
  byte data[9];
  // data[] : Données lues depuis le scratchpad
  // addr[] : Adresse du module 1-Wire détecté
   /* Reset 1-Wire bus and select sensor */
  ds.reset();
  ds.select(addr);
  /* Start data reaad and wait until it's done */
  ds.write(0x44, 1);
  delay(800);
  /* Reset 1-Wire bus, select sensor and started reading the scratchpad */
  ds.reset();
  ds.select(addr);
  ds.write(0xBE);
 /* Read scratchpad */
  for (byte i = 0; i < 9; i++) {
    data[i] = ds.read();
  }
  /* Read temperature in degrees */
  return (int16_t) ((data[1] << 8) | data[0]) * 0.0625;
}

```

```
61
62  void getDS18B20(){
63    float temperature[3];
64
65    /* Read all sensors*/
66    temperature[0] = getTemperature(SENSOR_ADDRESS_1);
67    temperature[1] = getTemperature(SENSOR_ADDRESS_2);
68    temperature[2] = getTemperature(SENSOR_ADDRESS_3);
69
70    /* Print temp */
71    /*Serial.print(F("Temperatures : "));
72    Serial.print(temperature[0], 2);
73    Serial.write(176); // Caractère degré
74    Serial.print(F("C, "));
75    Serial.print(temperature[1], 2);
76    Serial.write(176); // Caractère degré
77    Serial.print(F("C, "));
78    Serial.print(temperature[2], 2);
79    Serial.write(176); // Caractère degré
80    Serial.println('C');*/
81
82    msg.temp0 = (temperature[0]*10); // temperature in degrees
83    msg.temp1 = (temperature[1]*10); // temperature in degrees
84    msg.temp2 = (temperature[2]*10); // temperature in degrees
85  }
86  // ================================================================================ DS18B20 end
87
88  /*
89   * Weight sensor HX711
90   */
91  // ================================================================================ HX711 begin
92  // HX711 circuit wiring
93  const int DOUT = 2; // White wire
94  const int PSCK = 3; // Yellow wire
95
```

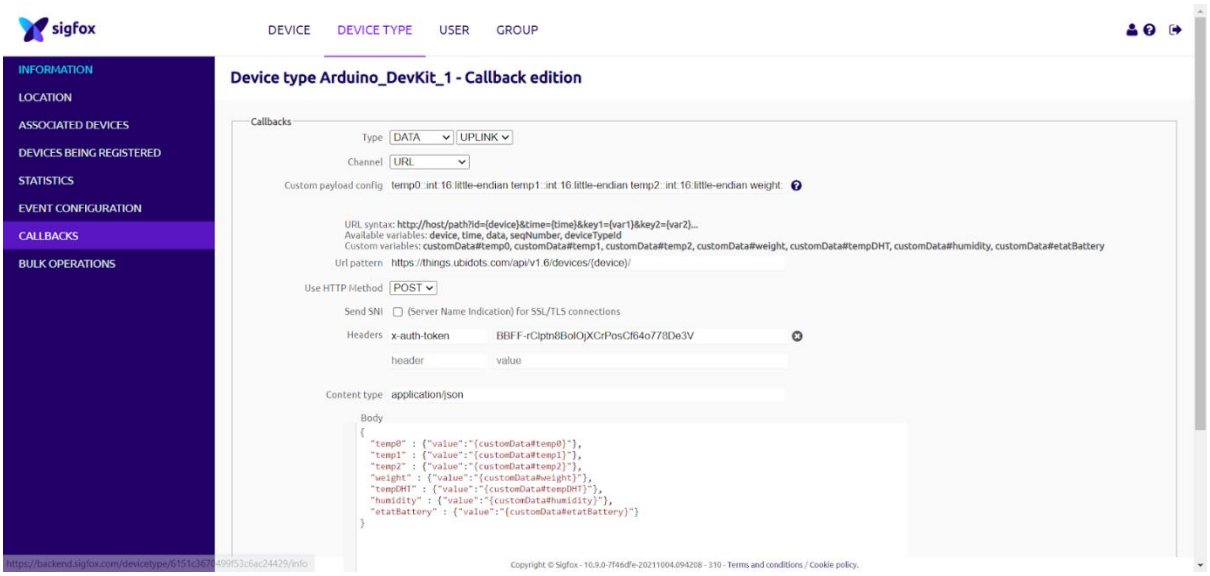
```
95
96  HX711 scale;
97
98  float calibration_factor = 20950; // Personnal factor for calibration of the scale
99  float zero_factor = 777787; //Numeric value of 0 Kg
100
101 void getHX711(){
102   scale.power_up(); //Turn on the scale
103   msg.weight = (scale.get_units())*10; // Weight in kg
104   scale.power_down(); //Turn off the scale
105   if (msg.weight < 0) {
106     msg.weight = 0;
107   }
108   /*Serial.print(msg.weight/10);
109   Serial.println(" kg");*/
110 }
111 // ================================================================================ HX711 end
112
113 // ================================================================================ DHT22 begin
114 #define DHTPIN 1 // Digital pin connected to the DHT sensor
115 #define DHTTYPE DHT22 // Specify which DHTxx we use
116 DHT dht(DHTPIN, DHTTYPE);
117
118 void getDHT22(){
119   msg.tempDHT = ((dht.readTemperature())*2); // temperature degrees
120   msg.humidity = ((dht.readHumidity())); // humidity percent
121   /*Serial.print(msg.tempDHT);
122   Serial.println("°C -> DHT");
123   Serial.print(msg.humidity);
124   Serial.println("% of humidity");*/
125 }
126 // ================================================================================ DHT22 end
127
```

```
// ================================================================================ DHT22 inside begin
#define DHTPINinside 4 // Digital pin connected to the DHT sensor
#define DHTTYPEinside DHT22 // Specify which DHTxx we use
DHT dhtInside(DHTPINinside, DHTTYPEinside);

void getDHT22inside(){
  msg.humidityInside = ((dhtInside.readHumidity())); // humidity percent inside the hive
  /*
  Serial.print(msg.humidityInside);
  Serial.println("% of humidity");
  */
}
// ================================================================================ DHT22 inside end

// ================================================================================ Battery begin
#define ADC A1

void getBattery(){
  analogReadResolution(10);  // 10 bits accuracy
  float voltage=0,conversion=0;
  int sensorValue = analogRead(ADC); //ADC read
  //float voltage = ((float)sensorValue * (4.2/1023)); // scaling to display battery value
  // conversion = (voltage-3.2)*100;
  // 880,900,920,940,960,980,1000
  // multiplication of 12.5 of the battery
  // for adaptating the percent
  if(sensorValue < 856) msg.etatBattery = 0;
  else if((sensorValue >= 856)&&(sensorValue < 880)) msg.etatBattery = 1;   // 0    / 12.5 %
  else if((sensorValue >= 880)&&(sensorValue < 900)) msg.etatBattery = 2;   // 12.5 / 25 %
  else if((sensorValue >= 900)&&(sensorValue < 920)) msg.etatBattery = 3;   // 25   / 37.5 %
  else if((sensorValue >= 920)&&(sensorValue < 940)) msg.etatBattery = 4;   // 37.5 / 50 %
  else if((sensorValue >= 940)&&(sensorValue < 960)) msg.etatBattery = 5;   // 50   / 62.5 %
  else if((sensorValue >= 960)&&(sensorValue < 980)) msg.etatBattery = 6;   // 62.5 / 75 %
  else if((sensorValue >= 980)&&(sensorValue < 1000)) msg.etatBattery = 7;  // 75   / 87.5 %
  else if((sensorValue >= 1000)&&(sensorValue < 1024)) msg.etatBattery = 8; // 87.5 / 100 %
```

```
163     // else Serial.println("ERROR battery");
164     // Serial.println(msg.etatBattery*12.5);
165   }
166
167   // ================================================================================ Battery end
168
169
170   void sendValues(){
171     // Clear all pending interrupts
172     SigFox.begin();
173     SigFox.status();
174     // Send the data
175     SigFox.beginPacket();
176     SigFox.write((uint8_t*)&msg, sizeof(SigfoxMessage));
177     /*Serial.print("Status: ");
178     Serial.println(SigFox.endPacket());*/
179     SigFox.endPacket();
180     SigFox.end();
181     /*Serial.println("\n\n\n\n\n");*/
182   }
183   /** Fonction setup() **/
184   void setup() {
185
186     // while (!Serial);
187     if (!SigFox.begin()) {
188     /*Serial.println("SigFox error, rebooting");*/
189     reboot();
190     }
191     // Enable debug prints and LED indication
192     SigFox.debug();
193     /* serial port initialisation */
194     //Serial.begin(115200);
195     /* scale Initialisation*/
196     scale.begin(DOUT, PSCK); // PIN initialisation
197     scale.set_scale(calibration_factor); // Adjust to this calibration factor
```

```
198     scale.set_offset(zero_factor); // Adjust to this zero factor
199     dht.begin(); //DHT22 initialisation
200     dhtInside.begin(); //DHT22Inside initialisation
201     pinMode(ADC, INPUT); //For the battery, configuration of ADC
202     pinMode(0, OUTPUT); //Show a light when the system start
203     digitalWrite(0, HIGH);   // turn the LED on (HIGH is the voltage level)
204     delay(3000);                       // wait for a second
205     digitalWrite(0, LOW);    // turn the LED off by making the voltage LOW
206   }
207
208
209   /** Fonction loop() **/
210   void loop() {
211     getDS18B20();
212     getHX711();
213     getDHT22();
214     getDHT22inside();
215     getBattery();
216     sendValues();
217
218     LowPower.deepSleep(1200000); //wait 20min (1200000)
219     //delay(10000); //60 et 140 mA
220   }
```

As mentioned earlier, this is the Arduino code, which allows the raw data to be sent to Sigfox. This network, which also has a graphical interface, will have to be added a function called “Callbacks”. It is thanks to this function that the information will be interpreted and sent to Ubidots.




For example, the variable “temp0”, which corresponds to the temperature returned by sensor number 1, will take the value of the first DHT11, the temperature sensor, and send it to Ubidots for it to be displayed more intuitively. In the end, this information will be shown to the apiarists :


In fact, the image above represents all the data Sigfox send to Ubidots. We can see the battery state, the consumption over time, the different temperatures, the humidity and the weight.

## Members of the group

Jeremy COLZY
Keïta EFFOUDOU MEZI
Lucas GUEDON
Valentine YAO