Séances 5 et 6 : Chauffage d'appoint pour les abeilles « BeeWarm »

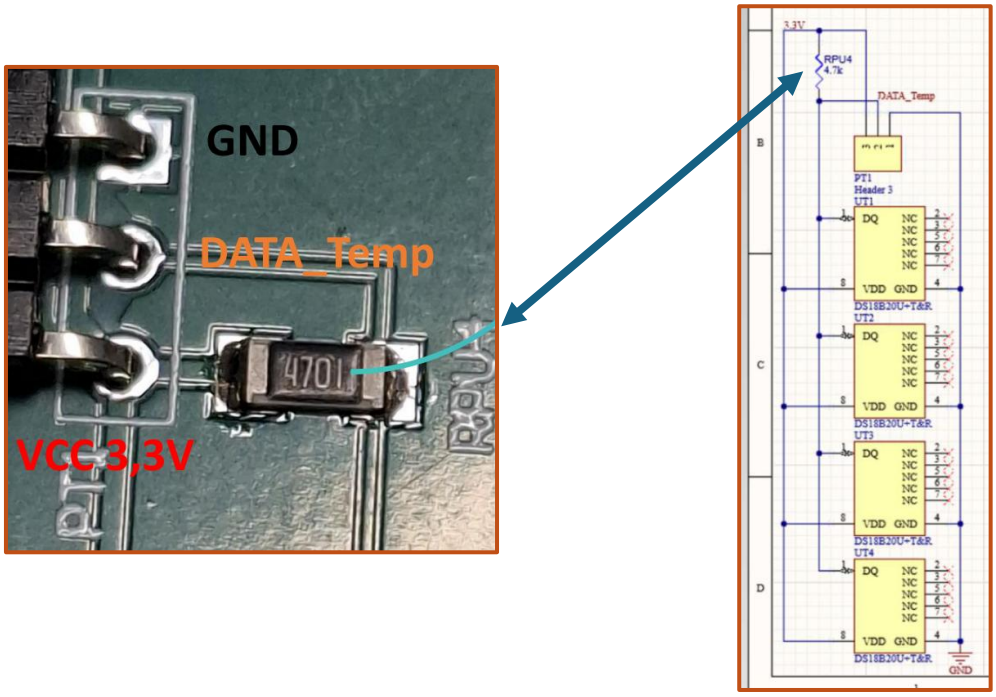
Introduction :

La 5ème et 6ème séances avaient pour objectif de remplacer les valeurs de température simulées par des mesures réelles issues des capteurs installés sur la carte électronique. Après avoir validé lors des séances précédentes la communication Bluetooth entre l'ESP32 et l'application mobile à l'aide de données fictives, il était nécessaire de mettre en œuvre la lecture effective des capteurs de température afin de rendre le système pleinement fonctionnel, ainsi que de pouvoir travailler véritablement avec la consigne modifiée.

1) Mise en place des capteurs de température :

La PCB intègre 4 capteurs de température DS18B20, dédiés à la mesure de la température de la végétaline, ainsi qu'un 5ème capteur spécifiquement destiné à la mesure de la température au cœur de la ruche.
Les capteurs de végétaline sont connectés entre eux sur un bus **OneWire** commun, ce qui leur permet de partager une ligne de données tout en conservant chacun une adresse propre. Cette ligne est reliée à l'ESP32 via la broche GPIO 14, avec une alimentation VCC en 3,3 V et une masse GND commune. Pour les tests, la carte est alimentée en 12 V, une alimentation solaire étant prévue par la suite.

Le capteur de température de la ruche est quant à lui déporté à l'extérieur de la PCB afin de pouvoir être placé directement dans l'environnement interne de la ruche. Il est connecté via un connecteur à trois broches comprenant l'alimentation VCC, la ligne de données OneWire et la masse GND. L'identification des broches a été facilitée par la présence de la résistance de tirage de 4,7 kΩ caractéristique du bus OneWire.




```
SENSOR found n° 0
2800B3D90F0000D2
SENSOR found n° 1
2812B3D90F0000E7
SENSOR found n° 2
2866FBD80F0000FB
SENSOR found n° 3
28616408EBE3980D
SENSOR found n° 4
280DB3D90F000098
```

Le protocole OneWire permet à chaque capteur DS18B20 de disposer d'une adresse unique codée sur 8 octets. Et donc, lors du démarrage de l'ESP32, l'ensemble des capteurs présents sur le bus est automatiquement détecté.

2) Adaptation du code Arduino :

Les bibliothèques **OneWire** et **DallasTemperature** ont été utilisées pour assurer la communication avec les capteurs DS18B20.

Tout d'abord, le programme commence par inclure toutes les bibliothèques nécessaires :

- OneWire et DallasTemperature : communication avec les sondes de température DS18B20 (bus OneWire).
- Wire : gestion du bus I2C, utilisé notamment pour le capteur de courant/tension de la ruche.
- BluetoothSerial : permet de créer un port série Bluetooth pour échanger des données avec l'application dans le smartphone.

- Adafruit_INA219 : bibliothèque dédiée au capteur de puissance INA219.

L'objet SerialBT représente la liaison série Bluetooth de l'ESP32.

Ensuite, on définit des constantes matérielles et variables globales :

- ONE_WIRE_BUS : broche utilisée par les capteurs DS18B20 (ligne DATA du bus OneWire).
- TEMP_SENSOR_PIN : alias pour clarifier le rôle de la broche.
- tempDeviceAddress : tableau qui sert à stocker l'adresse d'un capteur détecté sur le bus.
- tVeg et tRuche : températures mesurées pour la végétaline et la ruche.
- temperatureConsigne : température cible pour le chauffage de la ruche (modifiable par Bluetooth).
- oneWire et sensors : objets qui encapsulent la communication avec les DS18B20.
- numberOfDevices : nombre total de capteurs de température présents sur le bus.
- numberOfCapteurRuche : nombre de capteurs dédiés à la ruche (ici 1).
- ina219 : objet représentant le capteur de courant/tension INA219.
- SIGNAL1 et SIGNAL2 : broches qui pilotent les résistances de chauffage via PWM.
- SDA_PIN et SCL_PIN : broches I^2C pour le capteur INA219.
- capteurRuche : adresse unique (8 octets) du capteur placé dans la ruche. Elle permet de le distinguer des capteurs de végétaline.

```
#include <OneWire.h>
#include <Wire.h>
#include <DallasTemperature.h>
#include "BluetoothSerial.h"
#include <Adafruit_INA219.h>
BluetoothSerial SerialBT;

#define ONE_WIRE_BUS 14   // Broche du DS18B20 (DATA_temp)
#define TEMP_SENSOR_PIN ONE_WIRE_BUS

DeviceAddress tempDeviceAddress; // We'll use this variable to store a found device address

float tVeg = 0.0;
float tRuche = 0.0;
int temperatureConsigne=33;

// Initialisation du bus OneWire et du capteur
OneWire oneWire(TEMP_SENSOR_PIN);
DallasTemperature sensors(&oneWire);
int numberOfDevices; // Number of temperature devices found
int numberOfCapteurRuche = 1;

Adafruit_INA219 ina219;
#define SIGNAL1 2
#define SIGNAL2 5 // resistances
bool heating = false;

#define SDA_PIN 21 // pin ina219 serial data
#define SCL_PIN 22 // pin ina219 serial clock

DeviceAddress capteurRuche = { 0x28, 0x61, 0x64, 0x08, 0xEB, 0xE3, 0x98, 0x0D };
```

Lors de l'initialisation (fonction setup()), on démarre les communications série (Serial.begin(115200)) et Bluetooth (SerialBT.begin("BeeWarmESP32")) afin de permettre le débogage et les échanges avec l'application mobile. Les capteurs de température DS18B20 sont ensuite initialisés à l'aide des bibliothèques OneWire et DallasTemperature (sensors.begin()), puis le nombre de capteurs est détecté avec getDeviceCount().

```
void setup() {
  delay(200);
  SerialBT.begin("BeeWarmESP32");  // nom de l'esp32 visible sur le smartphone
  Serial.begin(115200);
  delay(200);
  Serial.println("TESSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSST1");
  // Start up the library for temp sensor
  sensors.begin();
  // Grab a count of devices on the wire
  numberOfDevices = sensors.getDeviceCount();
  // Loop through each device, print out address
  for(int i=0;i<numberOfDevices; i++) {
    // Search the wire for address
    Serial.println("TESSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSST2");
    if(sensors.getAddress(tempDeviceAddress, i)) {
      Serial.print("SENSOR found n° ");
      Serial.print(i, DEC);
      Serial.println();
      printAddress(tempDeviceAddress);
      Serial.println();
    }
    else {
      Serial.print("Ghost device at ");
      Serial.print(i, DEC);
      Serial.println(" but could not detect address. Check power and cabling...");
    }
  }
  //end temp config

  // Configuration de l'I2C
  if(!Wire.begin(SDA_PIN, SCL_PIN)){
      Serial.print("E: ");
      Serial.println("I2C non initialise");
  }
```

```
  else{
      Serial.print("I: ");
      Serial.println("I2C initialise");
  }

  //INA219
  if (! ina219.begin()) {
      Serial.print("E: ");
      Serial.println("Failed to find INA219 chip");
      while (1) { delay(10); }
  }
  else{
    ina219.setCalibration_32V_2A();
    Serial.print("I: ");
    Serial.println("INA219 initialise et calibre pour utilisation: 32V,2A");
  }

  delay(200);
}

void printAddress(DeviceAddress deviceAddress) {
  for (uint8_t i = 0; i < 8; i++) {
    if (deviceAddress[i] < 16) Serial.print("0");
    Serial.print(deviceAddress[i], HEX);
  }
}
```

Chaque capteur est parcouru afin d'afficher son DeviceAddress, ce qui permet d'identifier le capteur spécifique de la ruche (capteurRuche). Enfin, le bus I^2C est configuré via Wire.begin(SDA_PIN, SCL_PIN) et le capteur de puissance INA219 est initialisé et calibré (ina219.begin(), setCalibration_32V_2A()), garantissant des mesures fiables de courant et de puissance.

La fonction loop() regroupe, quant à elle, les tâches principales exécutées en continu. Elle commence par la réception d'une consigne de température via SerialBT.available(), permettant de modifier la variable temperatureConsigne à distance.

Elle est extraite du message sous la forme SET:SP=XX, la valeur numérique étant isolée après le caractère = puis convertie en entier. Cette nouvelle consigne est immédiatement prise en compte par l'algorithme de régulation thermique, permettant un ajustement rapide et flexible du comportement du système

Les températures sont ensuite acquises avec sensors.requestTemperatures(). Grâce à la comparaison des adresses (memcmp()), la température du capteur de la ruche est stockée dans tRuche, tandis que les mesures des autres capteurs sont moyennées pour obtenir tVeg. Ces données sont ensuite envoyées au smartphone sous forme de message Bluetooth (SerialBT.println()).

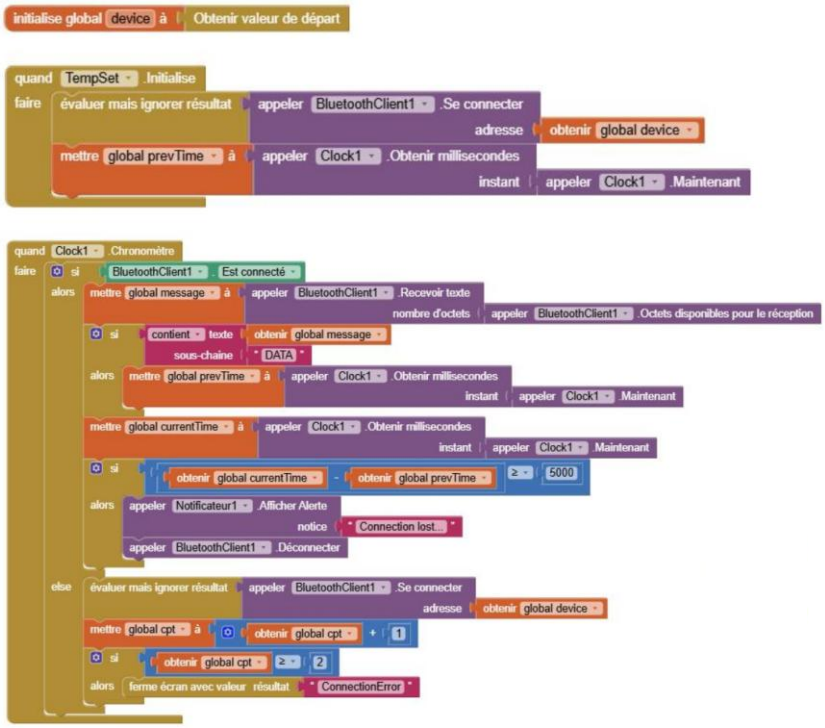
Enfin, la puissance disponible est mesurée par le capteur INA219 (getPower_mW()), puis les résistances de chauffage sont pilotées en PWM via analogWrite(SIGNAL1, SIGNAL2) en fonction de l'écart entre tRuche et temperatureConsigne. Une temporisation (delay(3000)) stabilise le cycle de fonctionnement.

```
void loop() {
  if(SerialBT.available()){
    String messageRecu = SerialBT.readString(); // On recupere le message sous forme de caractere
    messageRecu.trim();   // Enlève espaces, \r, \n
    delay(6);
    Serial.println(messageRecu);
    temperatureConsigne = messageRecu.substring(messageRecu.lastIndexOf('=') + 1).toInt();
    Serial.print("Température de consigne modifiée : ");
    Serial.print(temperatureConsigne);
    Serial.print(" °C");
    Serial.println();
  }
  // Lire la valeur du capteur de temperature
  sensors.requestTemperatures(); // Send the command to get temperatures
  // Loop through each device, print out temperature data
  tVeg = 0.0;
  tRuche = 0.0;
  for(int i=0;i<numberOfDevices; i++) {
    // Search the wire for address
    if(sensors.getAddress(tempDeviceAddress, i)){
      Serial.print("Capteur: ");
      Serial.println(i,DEC);
      float tempC = sensors.getTempC(tempDeviceAddress);
      if (tempC == DEVICE_DISCONNECTED_C) {
        Serial.println("Capteur de temp deconnecte !");
        return;
      }
      if (memcmp(tempDeviceAddress, capteurRuche, 8) == 0) {
        tRuche = tRuche + tempC;
      }
      else {
        tVeg = tVeg + tempC;
      }
      Serial.print(tempC);
```

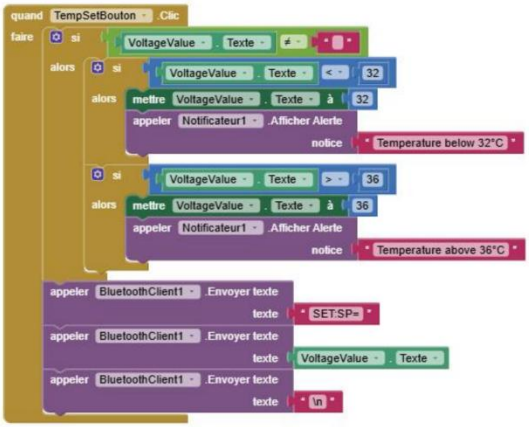
```
      Serial.println("°C");
    }
  }
  tVeg = tVeg/(numberOfDevices-numberOfCapteurRuche); // moyenne des mesures de température pour la végétaline

  String message = "DATA;" + String(tRuche, 1) + "°C;" + String(tVeg, 1) + "°C"; // conversion float à string avec ##,#
  SerialBT.println(message); // envoie du message
  Serial.println();
  Serial.println(message);

  // Controle de température avec les resistances

  float shuntVoltage = ina219.getShuntVoltage_mV();
  float busVoltage   = ina219.getBusVoltage_V();
  float current_mA   = ina219.getCurrent_mA();
  float power_mW     = ina219.getPower_mW();

  Serial.print("Bus Voltage:   "); Serial.print(busVoltage);   Serial.println(" V");
  Serial.print("Shunt Voltage: "); Serial.print(shuntVoltage); Serial.println(" mV");
  Serial.print("Current:       "); Serial.print(current_mA);   Serial.println(" mA");
  Serial.print("Power:         "); Serial.print(power_mW);     Serial.println(" mW");
  Serial.println("-----------------------------------");


  int MPPT = 4000;
  int puissanceDeChauffe = (power_mW*255)/MPPT; // produit en croix
  if (tRuche > temperatureConsigne + 2) {
    analogWrite(SIGNAL1, puissanceDeChauffe);
    analogWrite(SIGNAL2, puissanceDeChauffe);
  }
  else if (tRuche > temperatureConsigne - 2) {
    analogWrite(SIGNAL1, 0);
    analogWrite(SIGNAL2, 0);
  }
```

3) Interaction améliorée avec l'application :

Côté application, deux pages principales ont été mises en œuvre : la page de *Monitoring* permettant d'afficher les valeurs envoyées par l'ESP32, et la page *SET*, dédiée à la modification de la consigne de température. Sur cette page, la réception des messages Bluetooth sert principalement à vérifier que la connexion reste active. En l'absence de données reçues pendant un certain temps, l'application considère que la connexion est perdue et tente automatiquement de se reconnecter.

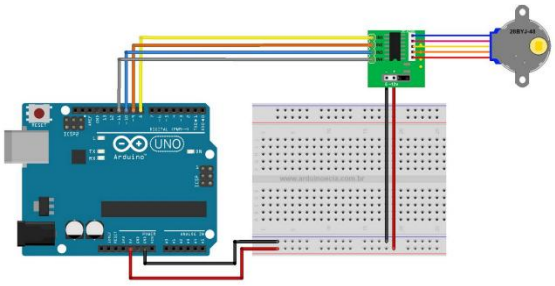


Lors de la saisie d'une nouvelle consigne, une vérification logicielle est effectuée : si la température entrée est inférieure à 32 °C ou supérieure à 36 °C, elle est automatiquement corrigée et une alerte est affichée à l'utilisateur. Une fois la valeur validée, la commande Bluetooth correspondante est envoyée à l'ESP32, garantissant une transmission fiable et sécurisée de la consigne.



4) Rotation dans les deux sens du moteur :

Pour faire touner le panneau, un second ESP32 est dédié au pilotage d'un moteur pas à pas. Le contrôle est assuré par une fonction développée en bas niveau, qui active successivement les bobines du moteur à l'aide de digitalWrite() selon une séquence définie, intégrée dans une boucle for allant de 0 à 511. Ce nombre de 512 pas correspond au fonctionnement interne du moteur et de son réducteur, et permet



d'obtenir un tour complet de l'axe de sortie, garantissant une rotation précise du panneau.

Lors du développement, in a utilisé la bibliothèque Stepper pour simplifier le pilotage du moteur. Cependant, lors des tests, la fonction Stepper.step() provoquait une rotation en sens antihoraire sulement. J'ai participé au débogage de cette partie du système. Il a alors été décidé d'abandonner la fonction de la bibliothèque au profit d'une fonction personnalisée, permettant de forcer une rotation dans le sens horaire simplement en modifiant l'ordre d'activation des bobines.

```
#define IN1  2
#define IN2  3
#define IN3  4
#define IN4  5
int temps =10;
int i=0;
void setup(){
  Serial.begin(115200);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  for (i=0; i <= 511; i++){
    Serial.println(i);
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, HIGH);
    delay(temps);

    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
    digitalWrite(IN3, HIGH);
    digitalWrite(IN4, LOW);
    delay(temps);
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, HIGH);
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    delay(temps);

    digitalWrite(IN1, HIGH);
    digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    delay(temps); }}
void loop() {}
```