Reporting Quatre Quadrants n°4

Projet Minuto - FISE A1

Groupe B11

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

12 novembre 2024

Consitution de l'équipe

Groupe/Equipe : B11
Chef de projet : Emilien WOLFF

Actualisation de la fiche à la date du : 12/11

▷ Ce que nous avions prévu de faire aujourd'hui

- Mesure de c_T à l'aide de la relation :

$$c_{\text{laiton}} = \frac{E}{m_{\text{laiton}} \times \Delta T} = \frac{(V \times I) \times \Delta t}{m_{\text{laiton}} \times (T_{\text{f}} - T_{\text{i}})}$$

  Or on remarque que :

$$\frac{\Delta T}{\Delta t} = \frac{P}{mc_t}$$

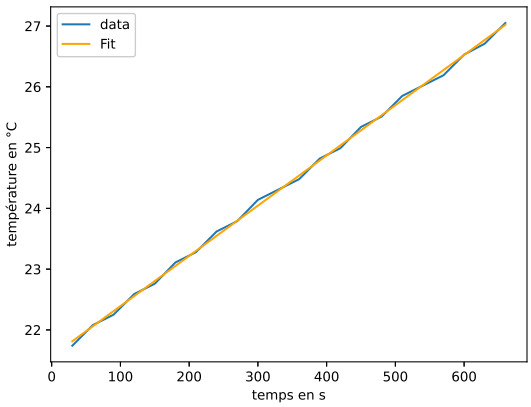
  Ce rapport est donc constant et nous allons donc réaliser plusieurs mesures de températures sur des intervalles $\Delta t = 30$ s puis réaliser une régression linéaire.
- Suivi du projet dans *GanttProject*
- Continuer le compte-rendu final sur *Overleaf*
- Étalonnage du **pont de Wheatstone**
- Quantifier les échanges conducto-convectifs non pris en compte dans la relation

$$F_{\text{irrad}} = \frac{m \cdot c_T \cdot \Delta T}{\Delta t \cdot S}$$

- Comparaison de la précision du **pont de Wheatstone** et du montage avec le pont diviseur de tension
- Découpage et détorsadage du fil de NiCr pour qu'il ait les bonnes dimensions et pour garder uniquement le fil blanc chauffant

▷ Ce que nous avons réalisé effectivement

- Modélisation du boîtier en 3D à l'aide du logiciel Fusion 360
- Test d'affichage des valeurs de la température et de l'irradiance sur l'écran OLED.
- Prise de mesures du boîtier
- Découpage et détorsadage du fil de NICr pour qu'il ait les bonnes dimensions de manière à garder uniquement le fil blanc chauffant
- Pour mesurer la capacité calorifique c_T nous avons :
  — Choisi un Δt = 30 s pendant lesquelles nous avons mesuré un ΔT. Nous avons réalisé cela pendant 11 minutes. Voici le tableau des premières valeurs :



Régression linéaire sur les valeurs mesurées

Temps (s)	Température (°C)
0	21.91
30	21.74
60	22.08
90	22.25
120	22.59
150	22.76
180	23.11
210	23.28
...	...
660	27.05

— Mesuré l'intensité i à l'aide de la loi d'Ohm : $I = 510mA$
— Mesuré la masse m de laiton : $m = 622g$

- On trouve finalement :

$$\boxed{c_T = 423 \text{ J}\cdot K^{-1} \cdot kg^{-1}}$$

- Suivi du projet dans *GanttProject*
- Continuer le compte-rendu final sur *Overleaf*
- Modélisation 3D du boîtier

▷ Ce que nous prévoyons de faire les prochains jours

- Modification des côtes du boîtier et impression 3D (attention à ne pas choisir des couleurs sombres)
- Deuxième campagne de mesures

▷ Problèmes rencontrés et solutions mises en œuvre

- Essai infructueux du pont de Wheatstone (mauvais branchements que nous allons reprendre dans la semaine)

Annexes : codes de la session

Code final comprenant l'affichage sur le LCD

```

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>       // for the rectangle draw and text display
#include <Adafruit_SSD1306.h>   // to control the OLED chip


int measurement_counter = 0;  // count the number of measurements done; used for Serial print

//////////////////////////
//    SCREEN DESCRIPTION
//      ------------------
//     |------------------|   Area YELLOW 128 x 16 pixels (0 to 15)
//     |                  |
//     |                  |   Area BLUE 128 x 48 (16 to 63)
//     |                  |
//     |------------------|

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define SCREEN_BLUE_START 16 // OLED blue area starts here in Y position
// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin). No reset on the screen module.
#define SCREEN_ADDRESS 0x3C // The address written on the OLED board (0x78) must be divided by 2 => 0x3C

// Create the display object connected to I2C bus (Wire)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

```

```
29 ////////////////////////////
30
31 //       FUNCTIONS        //
32
33 ////////////////////////////
34
35 ////////////////////////////////////
36 //  Function: draws a horizontal bar filled for the value given, in proportion
       to the max_value given
37 //  Input: an integer for the value and an integer for the max value (bar
       totally filled). Negative value not supported
38 //  Output: none
39 ////////////////////////////////////
40 void draw_value_in_bar(unsigned long value, unsigned long max_value)
41 {
42     if (value>max_value) value=max_value;   // avoid to try to draw an
           impossible bar.
43     display.fillRect(0, 0, display.width(), SCREEN_BLUE_START-1, SSD1306_BLACK);
           // Draw black rectangle = erase the bar display area
44     display.drawRect(0, 0, display.width(), SCREEN_BLUE_START-1, SSD1306_WHITE);
           // Draw the frame for the full bar (empty fill)
45     display.fillRect(0, 0, (int)display.width()*value/max_value,
           SCREEN_BLUE_START-1, SSD1306_WHITE); // Draw the necessary fill
           corresponding to the value
46     display.display();  // push to the display
47 }
48
49 ////////////////////////////////////
50 //  Function: displays a string centered vertically and horizontally in the blue
       part of the screen
51 //  Input: a text in the STRING type
52 //  Output: none
53 ////////////////////////////////////
54 void draw_string_centered(String text)
55 {
56   Serial.print("Measurement ");
57   Serial.print(measurement_counter);
58   Serial.print(" : ");
59   Serial.println(text);
60
61     // Text dimensions calculation
62     int16_t x1, y1;
63     uint16_t textWidth, textHeight;
64     display.setTextSize(3);                     // Normal: 1   Use 3X normal
           size
65     display.setTextColor(SSD1306_WHITE);        // Draw white text
66
67     display.getTextBounds(text, 0, 16, &x1, &y1, &textWidth, &textHeight);
68     // Erase the display area with black rectangle
69     display.fillRect(0, 16, display.width(), display.height(), SSD1306_BLACK);
70     // compute the text cursor position to get it centered
71     int16_t xPos = (display.width() - textWidth) / 2;
72     int16_t yPos = ( ((display.height()- SCREEN_BLUE_START - textHeight) / 2) +
           SCREEN_BLUE_START );
73     display.setCursor(xPos, yPos);
74     display.print(text);
75     display.display();
76 }
77
78 // ADC characteristics
79 #define MAX_VOLTAGE_ADC 4.6  // the real voltage used by the ADC on the arduino
       board
80 #define ADC_STEPS 1024       // Number of steps of the conversion : 10 bits
81 // Declaration of the coefficient for the function approaching the thermistance
       voltage with resistive voltage divider
82 #define THERM_DIV_COEFF_A0 1 // for constant part
```

```
#define THERM_DIV_COEFF_A1 1 // for linear part
#define THERM_DIV_COEFF_A2 1 // for square part
#define THERM_DIV_COEFF_A3 1 // for cubic part

////////////////////////////////////
//  Function: gives the temperature corresponding the the ADC value, using the thermistance curve
//  Input: the ADC value, and int
//  Output: the temperature as a float
////////////////////////////////////
float calculate_temperature(int adc_value)
{
    float voltage, temperature;
    voltage = MAX_VOLTAGE_ADC * adc_value / ADC_STEPS;
    temperature = THERM_DIV_COEFF_A0 + THERM_DIV_COEFF_A1*voltage + THERM_DIV_COEFF_A2*pow(voltage,2) + THERM_DIV_COEFF_A3*pow(voltage,3);
    return(temperature);
}

#define BLOC_MASSE  0.5 // mass in SI
#define BLOC_CAPA 300   // thermal capacity in SI
#define BLOC_SURFACE 0.05*0.05  // Surface of the black face in SI
////////////////////////////////////
//  Function: calculates the irradiance
//  Input: the current temperature of the bloc and the current time
//  Output: irradiance as an INT
////////////////////////////////////
float previous_temp;            // the previous temperature measured
unsigned long previous_time;    // the value is in millisecond
int calculate_irradiance(float temp_now, unsigned long time_now)
{
    float irrad;
    irrad = BLOC_MASSE * BLOC_CAPA * abs(temp_now - previous_temp) / BLOC_SURFACE / (time_now - previous_time) * 1000;
    return((int)irrad);   // round to integer value, precision of the system is more than 5%
}


////////////////////////////

//          SETUP         //

////////////////////////////

float current_temp;
unsigned long current_time;
int analogPin = A0;         // the pin where the voltage value must be read
unsigned long measurement_start_time = 0;   // Time at which the measurement has started
int irradiance=0;
int measurement_duration = 10000;   // initial value set to 10 sec to have enough Delta T. This could be changed by the program to adjust if deltaT is not enough


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

  // SSD1306 OLED init
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }
  // Clear the display buffer and updates the screen (full clear)
  display.clearDisplay();
```

```
  display.display();

  previous_temp = calculate_temperature(analogRead(analogPin));   // initialize the value to the current temperature
  previous_time = millis();   // initialize time value
  Serial.println("");
  Serial.print("Measurement interval set to : ");
  Serial.print(measurement_duration/1000.0);
  Serial.println(" sec ");
}

#define MAX_SUN_IRRADIANCE 1200 // value max expected, can be used to avoid abnormal displays
#define MIN_TEMP_VARIATION 0.0001 // Minimum of variation to have a trustable irradiance calculation. 0.01 is normal variation in 1s with 1000W/m2
#define DELAY_BETWEEN_SCREEN_UPDATE 100  // in milliseconds



////////////////////////////

//          LOOP          //

////////////////////////////
void loop()
{
  delay(DELAY_BETWEEN_SCREEN_UPDATE);  // make a pause to have enough time between two measurements for temperature to change

  // update the time bar
  current_time = millis();
  draw_value_in_bar( current_time - previous_time , measurement_duration );  // draw the bar, max is set to the total measurement time

  if ( current_time - previous_time > measurement_duration )     // now the interval is reached, do the measurement and calculation and display
  {
      // get values
      current_temp = calculate_temperature((int)random(1024));     // fake value for test without hardware
      // current_temp = calculate_temperature(analogRead(analogPin));   // Real value measured
      current_time = millis();
      measurement_counter += 1;
      // calculate the irrandiance
      irradiance = calculate_irradiance(current_temp, current_time);

      // Display on screen
      if (irradiance < MAX_SUN_IRRADIANCE)
      {
          draw_string_centered(String(irradiance));          // Display the value below the time bar
      }
      else
      {
          draw_string_centered("ERROR");  // Value abnormal
      }


      // store for next delta T and delta t calculation
      previous_temp = current_temp;
      previous_time = current_time;
  }
}
```

Remarques : Le code fonctionne correctement et mesure de températures sont bonnes