PSE : Méthodes et Protocoles

1 Introduction

Lors de ce PSE nous avons étudié une phénomène qui a lieu sur la cristallisation du sel. Lorsqu'une solution saline s'évapore, le sel reste et cristallise. On observe alors que le sel cristallise en montant sur les parois, bien au-dessus du niveau de l'eau. Lorsqu'on ajoute une tige qui plonge dans l'eau, le sels cristallise même le long de la tige. Quels sont les paramètres d'influence de ce phénomène ? Comment influencent-t-ils la cristallisation ?

2 Montage expérimental

2.1 Schéma du montage

Le montage que nous avons réalisé est le suivant

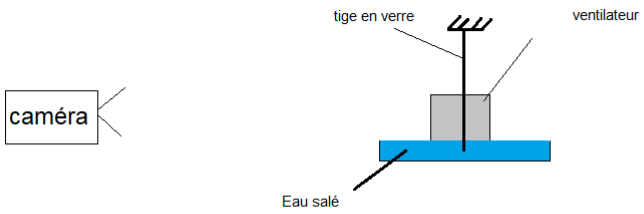

FIGURE 1 – Schéma du montage

La vitesse du ventilateur est pilotée par un générateur de tension, mais elle n'a pas été modifiée tout du long de l'étude. Nous avons aussi changé le récipient contenant l'eau, soit par une boite de pétri, soit par un cristallisoir. De plus, on a essayé de mettre des couvercles pour limiter l'évaporation. L'évolution de la cristallisation est photographiée par une caméra, au rythme d'une image par demi-heure. Un rétro-éclairage est assuré par un dépoli sur un fond noir (tissu) afin de mieux visualiser par transparence les cristaux de sels. Nous utilisons le logiciel Pylon Viewer associé à la caméra pour l'acquisition des images. Ces dernières sont ensuite traitées sur imageJ afin d'en déduire la hauteur de sel en fonction du temps.

2.2 Matériel utilisé

- Boîtes de pétri
- Cristallisoir
- Caméra
- Tige en verre (diamètre variable) d'au moins 20 cm de longueur
- Couvercle percé (un pour chaque diamètre)
- Ventilateur
- Potence et structure en Norcan
- Différents sels (KCl, $MgCl_2$, NaCl)
- Dépoli et tissu noir
- Logiciels : Python, Pylon Viewer et ImageJ

2.3 Remarques et conseils

— Les temps caractéristiques d'étude sont de l'ordre de la journée, il faut donc laisser le sel monter entre deux séances de PSE
— Des solutions à saturation permettent d'accélérer un peu le processus
—

3 Carte d'acquisition

Afin de vérifier l'influence des variations journalières de température et d'hydrométrie, nous avons ajouté au montage une station météo qui enregistre en temps réel la température et l'hydrométrie.

Le montage suivant à été réalisé, suivant le plan proposé sur le site suivant :
https://www.carnetdumaker.net/articles

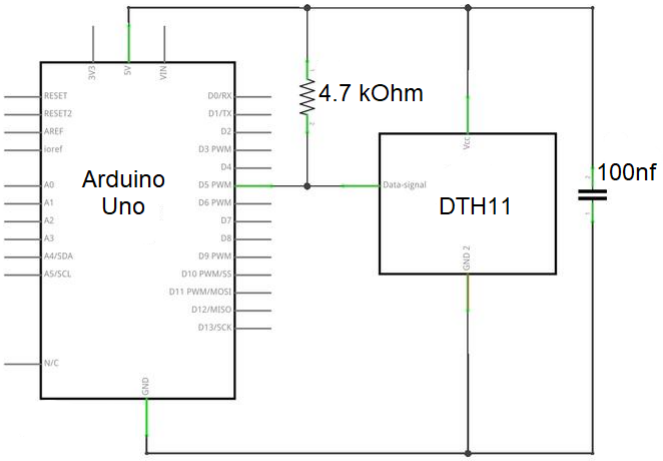

FIGURE 2 – Schéma de la station météo

Le code de la carte n'est pas très différent de celui du site. On laisse ensuite la carte sur la paillasse. La carte ne sert qu'à recueillir les informations sur l'hydrométrie et la température, elle est branchée en permanence sur un ordinateur qui enregistre dans un tableur les données. Les codes sont disponibles en annexe page 3 et 4.L'ordinateur est en python et la carte en C++.

Le matériel nécessaire est :

— Carte arduino UNO
— Capteur DTH11 (mais le DTH22 serait un meilleur choix)
— condensateur de 100 nF
— résistance de 4.7 k Ω

Les données sont enregistrées au format csv, idéal pour le traitement numérique.

4 Bibliographie

[1] M. J. Qazi, H. Salim, C. A. W. Doorman, E. Jambon-Puillet, N. Shahidzadeh, Salt creepinp as a self-amplifying crystallization process. Sci. Adv. 5, eaax1853 (2019)

A Annexe 1

```
# C:\Users\Elian\Documents\PC\2A\PSE\station météo\station.py
01| ###Acquisition des données arduino PSE
02|
03| import serial
04| import time
05|
06| nom_fichier = 'test.csv' #bien changer le nom du fichier à chaque
acquisition
07| duree_mesure = 24 #durée d'acquisition en heure
08|
09| ##Initialisation du fichier
10|
11| # make sure the 'COM#' is set according the Windows Device Manager
12| ser = serial.Serial('COM8', 9800, timeout=1)
13| fichier = open(nom_fichier,'w')
14| fichier.write('Temps(heure);Hydrometrie;Temperature\n')
15| fichier.close()
16|
17| ##Mesure
18|
19| start = time.time()
20| with open(nom_fichier,'a',encoding = 'utf-8') as f:
21|     while (time.time()-start)/3600 < duree_mesure:
22|         line = ser.readline() #recéption des données
23|         if line:
24|             string = line.decode()  # conversion en unicode
25|             f.write(str((time.time()-start)/3600)+';')
26|             f.write(string)
27|             print(string)
28| ser.close()
```

B Annexe 2

```
#include <Arduino.h>

/** Broche "DATA" du capteur */
const byte BROCHE_CAPTEUR = 5;

/* Code d'erreur de la fonction readDHT11() et readDHT22() */
const byte DHT_SUCCESS = 0;        // Pas d'erreur
const byte DHT_TIMEOUT_ERROR = 1;  // Temps d'attente dépassé
const byte DHT_CHECKSUM_ERROR = 2; // Données reçues erronées

int i = 0;


byte readDHTxx(byte pin, byte* data, unsigned long start_time, unsigned long timeout)
{
  data[0] = data[1] = data[2] = data[3] = data[4] = 0;
  // start_time est en millisecondes
  // timeout est en microsecondes

  /* Conversion du numéro de broche Arduino en ports / masque binaire "bas niveau" */
  uint8_t bit = digitalPinToBitMask(pin);
  uint8_t port = digitalPinToPort(pin);
  volatile uint8_t *ddr = portModeRegister(port);   // Registre MODE (INPUT / OUTPUT)
  volatile uint8_t *out = portOutputRegister(port); // Registre OUT (écriture)
  volatile uint8_t *in = portInputRegister(port);   // Registre IN (lecture)

  /* Conversion du temps de timeout en nombre de cycles processeur */
  unsigned long max_cycles = microsecondsToClockCycles(timeout);

  /* Evite les problèmes de pull-up */
  *out |= bit;  // PULLUP
  *ddr &= ~bit; // INPUT
  delay(100);   // Laisse le temps à la résistance de pullup de mettre la ligne de
données à HIGH

  /* Réveil du capteur */
  *ddr |= bit;  // OUTPUT
  *out &= ~bit; // LOW
  delay(start_time); // Temps d'attente à LOW causant le réveil du capteur
  // N.B. Il est impossible d'utilise delayMicroseconds() ici car un délai
  // de plus de 16 millisecondes ne donne pas un timing assez précis.

  /* Portion de code critique - pas d'interruptions possibles */
  noInterrupts();

  /* Passage en écoute */
  *out |= bit;  // PULLUP
  delayMicroseconds(40);
  *ddr &= ~bit; // INPUT

  /* Attente de la réponse du capteur */
  timeout = 0;
  while(!(*in & bit)) { /* Attente d'un état LOW */
    if (++timeout == max_cycles) {
        interrupts();
        return DHT_TIMEOUT_ERROR;
      }
  }

```

```
  timeout = 0;
  while(*in & bit) { /* Attente d'un état HIGH */
    if (++timeout == max_cycles) {
        interrupts();
        return DHT_TIMEOUT_ERROR;
      }
  }

  /* Lecture des données du capteur (40 bits) */
  for (byte i = 0; i < 40; ++i) {

    /* Attente d'un état LOW */
    unsigned long cycles_low = 0;
    while(!(*in & bit)) {
      if (++cycles_low == max_cycles) {
        interrupts();
        return DHT_TIMEOUT_ERROR;
      }
    }

    /* Attente d'un état HIGH */
    unsigned long cycles_high = 0;
    while(*in & bit) {
      if (++cycles_high == max_cycles) {
        interrupts();
        return DHT_TIMEOUT_ERROR;
      }
    }

    /* Si le temps haut est supérieur au temps bas c'est un "1", sinon c'est un "0"
*/
    data[i / 8] <<= 1;
    if (cycles_high > cycles_low) {
      data[i / 8] |= 1;
    }
  }

  /* Fin de la portion de code critique */
  interrupts();

  /*
   * Format des données :
   * [1, 0] = humidité en %
   * [3, 2] = température en degrés Celsius
   * [4] = checksum (humidité + température)
   */

  /* Vérifie la checksum */
  byte checksum = (data[0] + data[1] + data[2] + data[3]) & 0xff;
  if (data[4] != checksum)
    return DHT_CHECKSUM_ERROR; /* Erreur de checksum */
  else
    return DHT_SUCCESS; /* Pas d'erreur */
}

/**
 * Lit la température et le taux d'humidité mesuré par un capteur DHT11.
 *
 * @param pin Broche sur laquelle est câblée le capteur.
 * @param temperature Pointeur vers la variable stockant la température.
```

```
 * @param humidity Pointeur vers la variable stockant le taux d'humidité.
 * @return DHT_SUCCESS si aucune erreur, DHT_TIMEOUT_ERROR en cas de timeout, ou DHT_CHECKSUM_ERROR en cas d'erreur de checksum.
 */
byte readDHT11(byte pin, float* temperature, float* humidity) {

  /* Lit le capteur */
  byte data[5];
  byte ret = readDHTxx(pin, data, 18, 1000);

  /* Détecte et retourne les erreurs de communication */
  if (ret != DHT_SUCCESS)
    return ret;

  /* Calcul la vraie valeur de la température et de l'humidité */
  *humidity = data[0];
  *temperature = data[2];

  /* Ok */
  return DHT_SUCCESS;
}

void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  float temperature, humidity;
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(600000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(500);
  /* Lecture de la température et de l'humidité, avec gestion des erreurs */
  // N.B. Remplacer readDHT11 par readDHT22 en fonction du capteur utilisé !
  switch (readDHT11(BROCHE_CAPTEUR, &temperature, &humidity)) {
  case DHT_SUCCESS:

    /* Affichage de la température et du taux d'humidité */
    Serial.print(humidity, 2);
    Serial.print(';');
    Serial.print(temperature, 2);
    Serial.print(';');

    break;

  case DHT_TIMEOUT_ERROR:
    Serial.println(F("Pas de reponse !"));
    break;

  case DHT_CHECKSUM_ERROR:
    Serial.println(F("Pb de communication !"));
    break;
}}


```