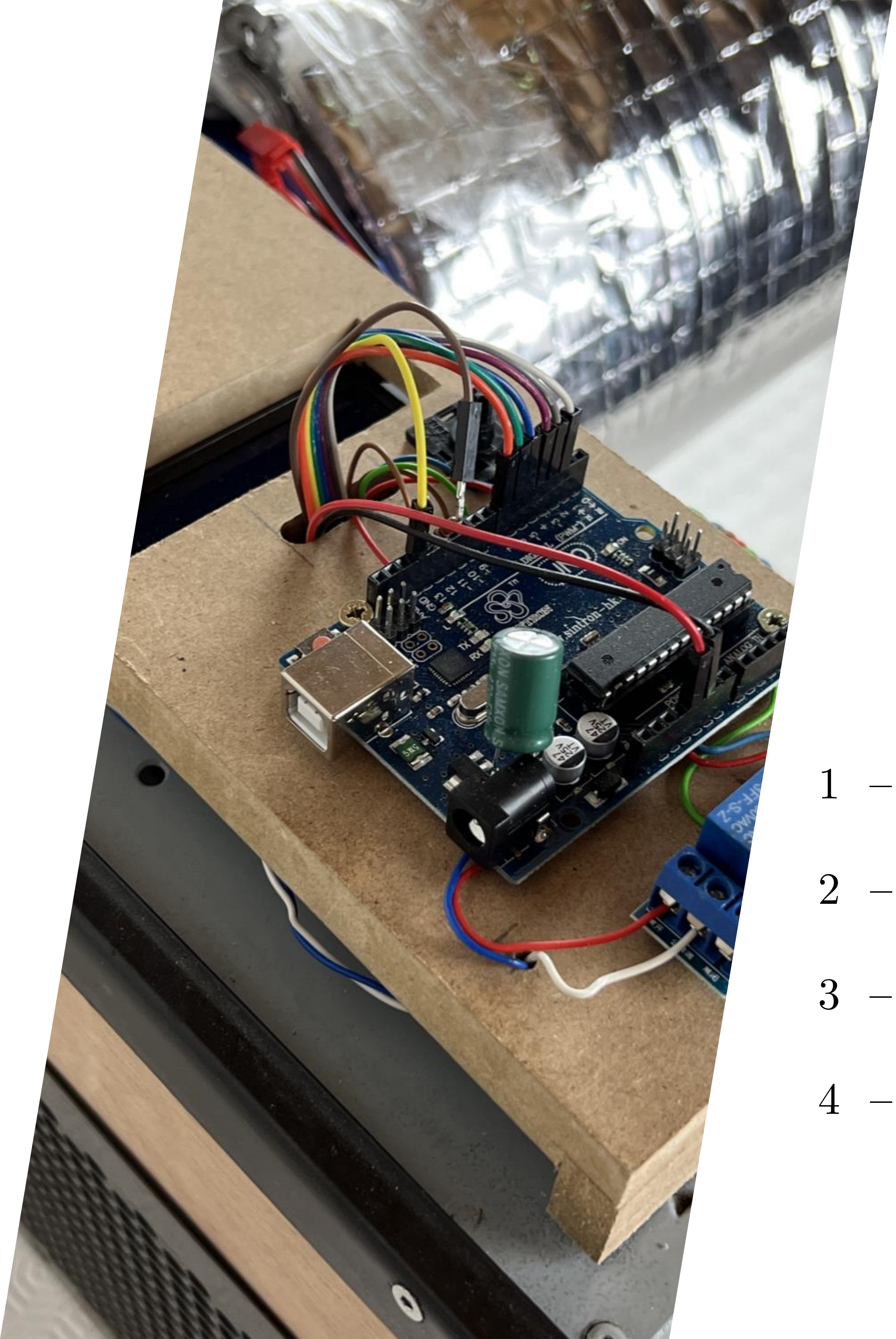
# CLIMATISATION PAR ÉVAPORATION

## IMPLÉMENTATION ET EFFICACITÉ

TIPE

Candidat n° 42427
Emilien WOLFF



# SOMMAIRE

***Problématique*** *: par quels moyens peut-on disposer d'un système de climatisation efficace d'une pièce, compact et peu couteux ?*

1 – Mise en contexte et CDCF

2 – Maquette conceptuelle

3 – Résultats et analyses

4 – Conclusion

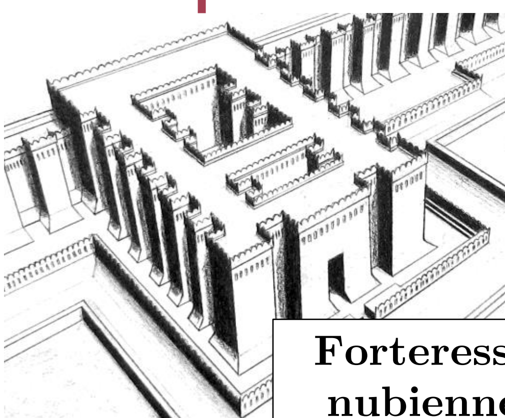
# 1 – MISE EN CONTEXTE ET CDCF

## HISTORIQUE

**Tour *Yakhchal***

***Zeer*, frigo du désert**

**- 2155** | **-2500** | **-2000** | **-400** | **2023**



**Forteresse nubienne**

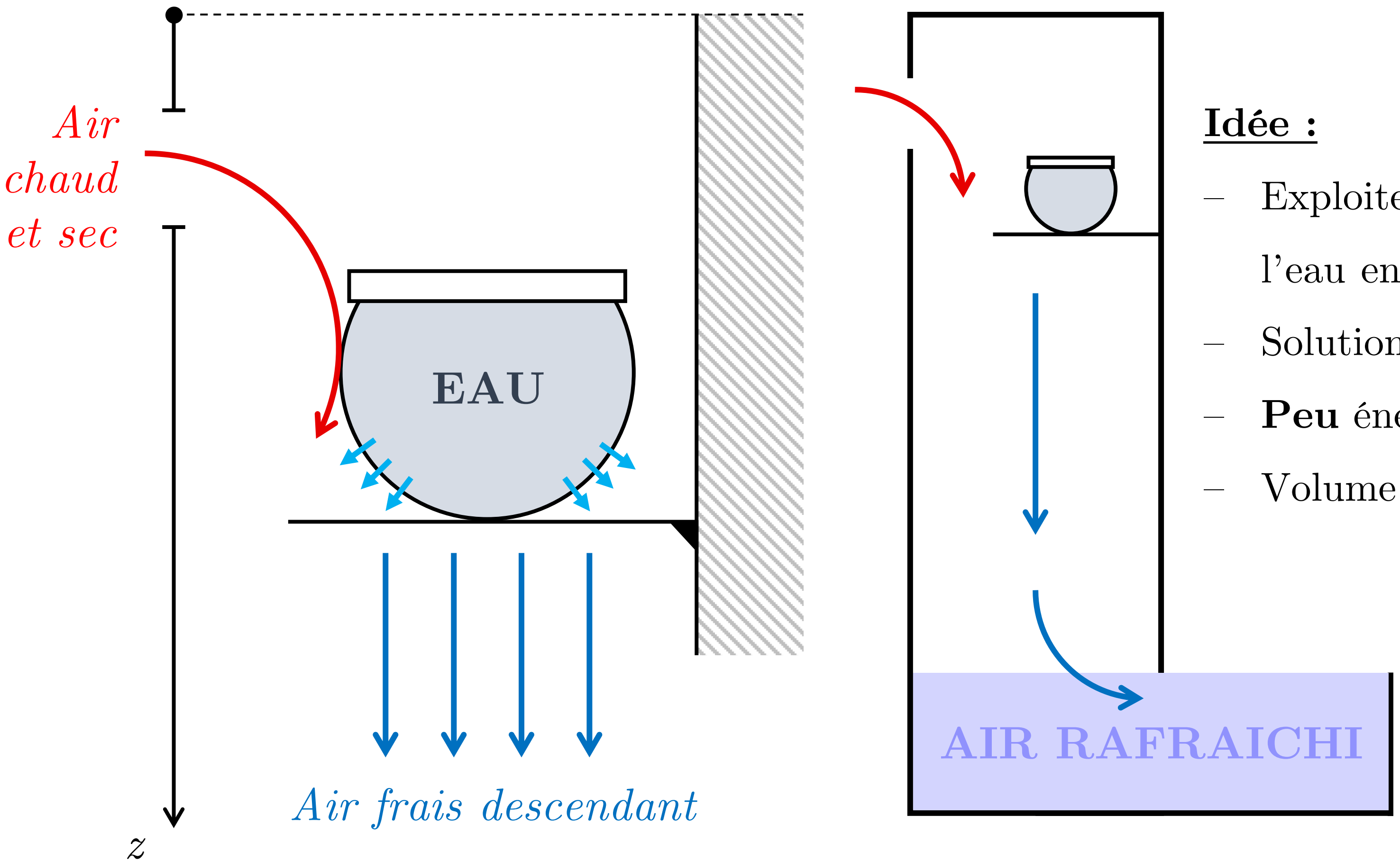
**Jarre, *Kashan***

**Climatiseur moderne**

# 1 – MISE EN CONTEXTE ET CDCF

**EXEMPLE**

**Exemple illustré** : la tour *Yakhchal*




<u>**Idée :**</u>

- Exploiter **l'évaporation** de l'eau en été
- Solution **compacte**
- **Peu** énergivore
- Volume **limité**

# 1 – MISE EN CONTEXTE ET CDCF

## PRINCIPES

- Cyclage **interne** de l'air
- Compresseur, échangeurs, fluide
- **Assèchement** de l'air interne

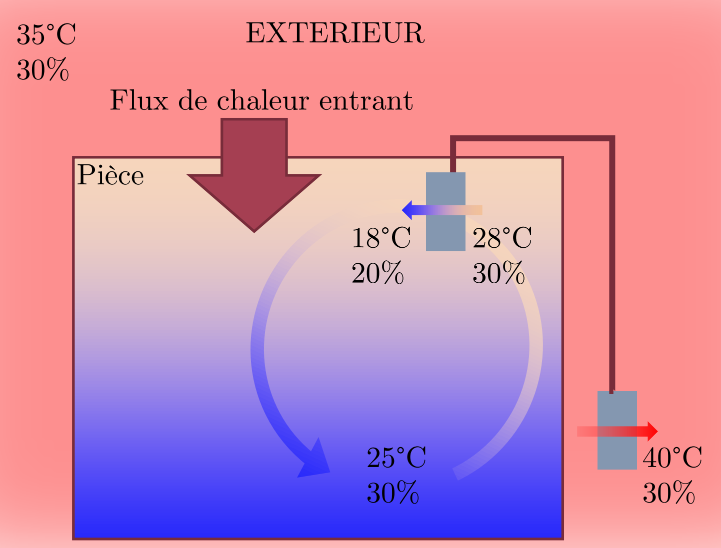



***Figure 1*** : Refroidissement par échangeur

# 1 – MISE EN CONTEXTE ET CDCF

## PRINCIPES

- Cyclage **interne** de l'air
- Compresseur, échangeurs, fluide
- **Assèchement** de l'air interne

- **Renouvellement** permanent de l'air
- Deux **ouvertures** sur l'extérieur
- Humidification de l'air : **confort**

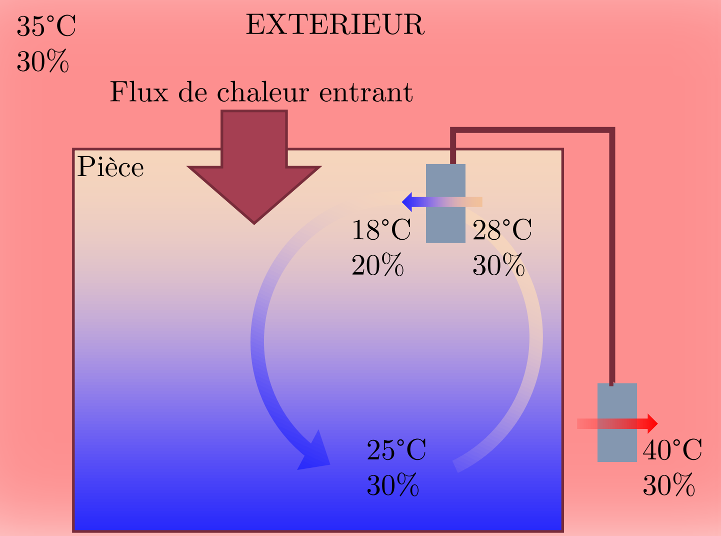



***Figure 1*** : Refroidissement par échangeur

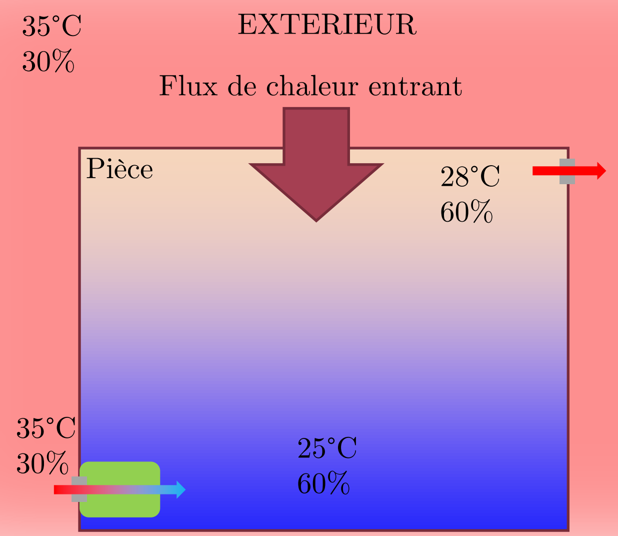



***Figure 2*** : Refroidissement par évaporation

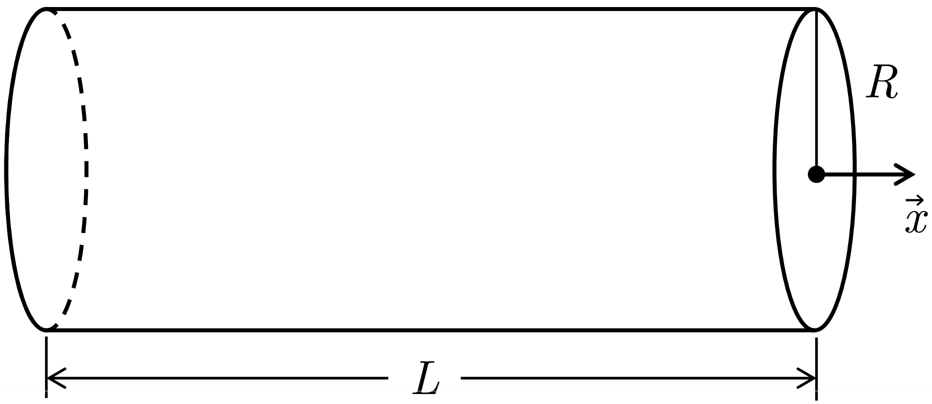
# 1 – MISE EN CONTEXTE ET CDCF

## CDCF

| Id | Exigence | Critère | Niveau |
| --- | --- | --- | --- |
| 1 | Confort d'habitation en été | Surface de pièce | [15 $m^2$ ; 20 $m^2$] |
| | | Régulation de la température intérieure | $< 25°C$ |
| | | Degré hygrométrique *NF35-102* | $HR_{\%} \in [50 ; 70]$ |
| | | Précision : erreur statique | $\pm$ 1°C |
| 2 | Disposer d'un système compact | Taille du boitier | 300(L) × 300(l) × 1000(h) |
| | | Poids du dispositif | $<$ 10 kg |
| 3 | Consommation faible | Eau | $<$ 20 L/jour |
| | | Electricité | $<$ 50W |
| 4 | Bruit de fonctionnement | Niveau sonore à 1m | $<$ 42dBSPL |

# 2 – MAQUETTE CONCEPTUELLE

## PRESENTATION

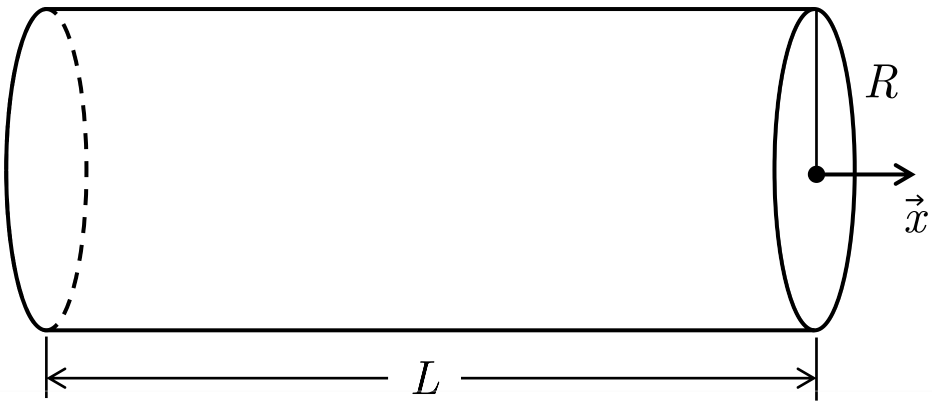



- Pertes de charge faibles
- Surface d'échange importante (périmètre)
- Disponibilité sur le marché

***Figure 3*** : modélisation *Fusion360* du tube utilisé

# 2 – MAQUETTE CONCEPTUELLE

## PRESENTATION




- Pertes de charge faibles
- Surface d'échange importante (périmètre)
- Disponibilité sur le marché

**Distribution d'eau**

<u>Cône de brumisation</u> : $\alpha = 60°$

<u>Débit</u> : 43 L/h

<u>Tissu perméable</u> non tissé pour retenir l'eau sur le cylindre

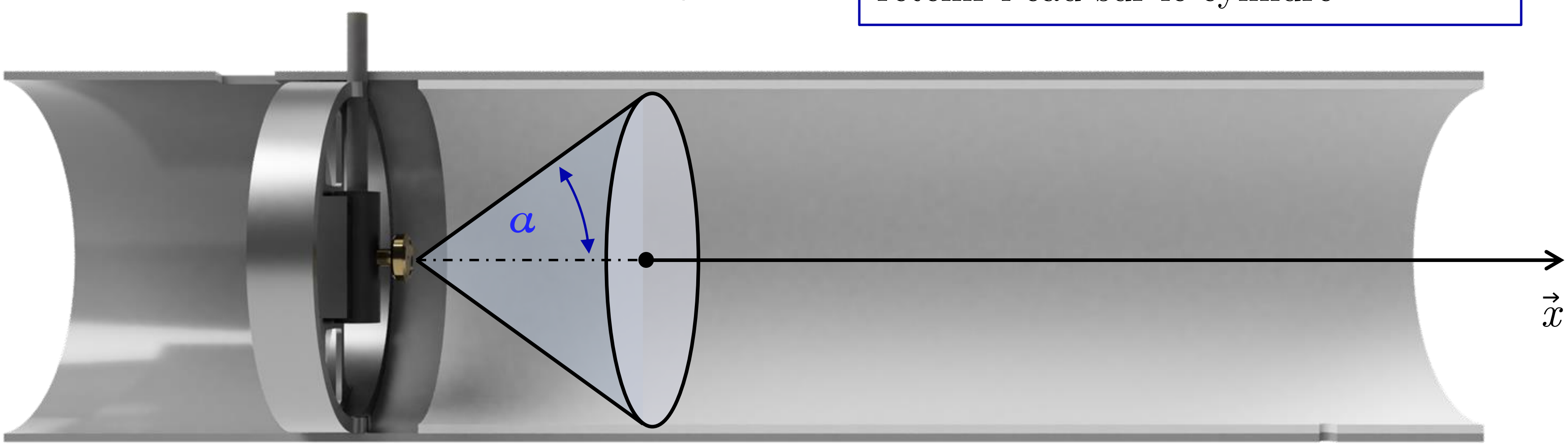



***Figure 3*** : modélisation *Fusion360* du tube utilisé

# 2 – MAQUETTE CONCEPTUELLE

## PRESENTATION

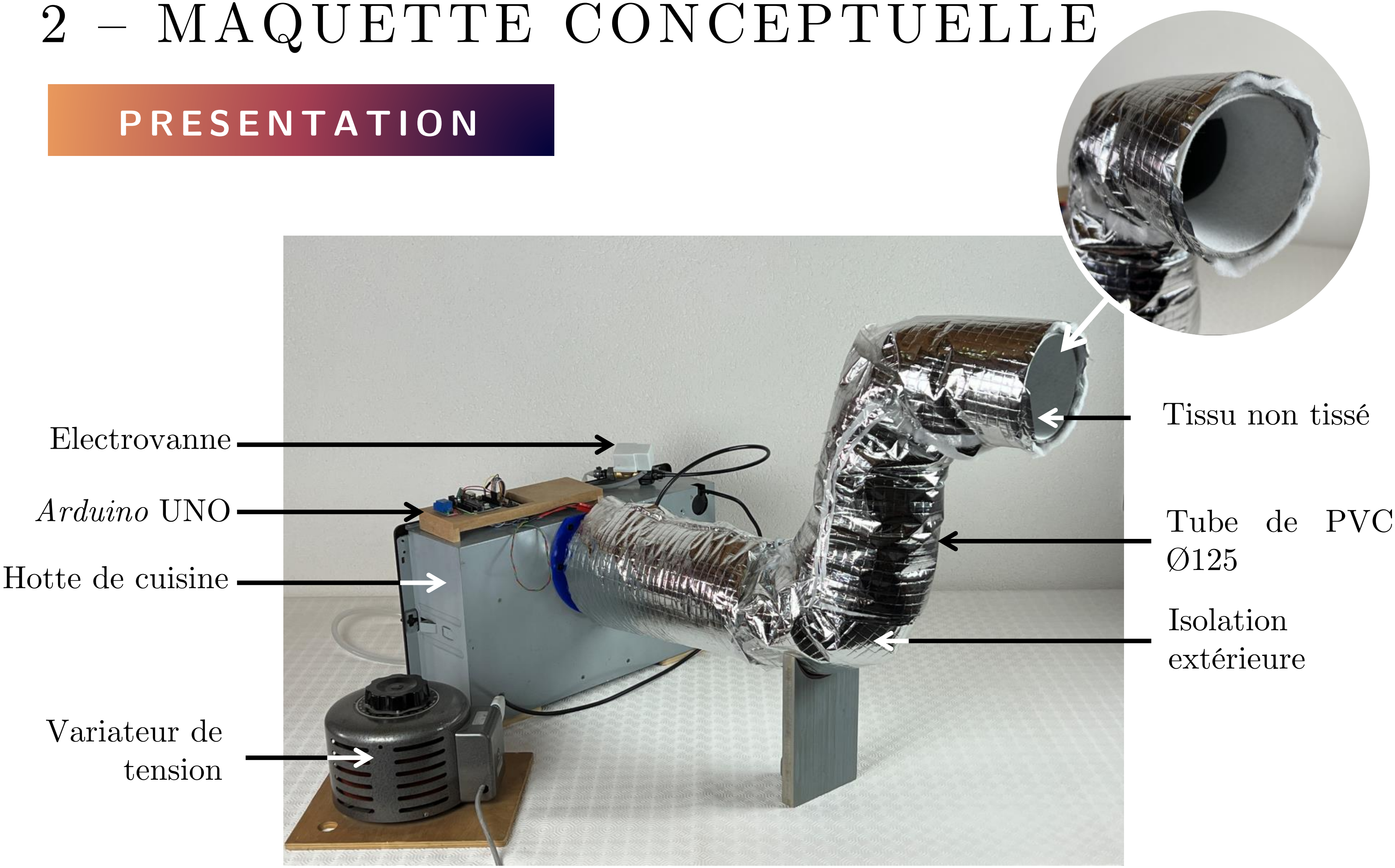



***Figure 4*** : maquette fabriquée

# 2 – MAQUETTE CONCEPTUELLE

## PRESENTATION

**Avantages**

- Efficacité de flux (consommation)
- Silence

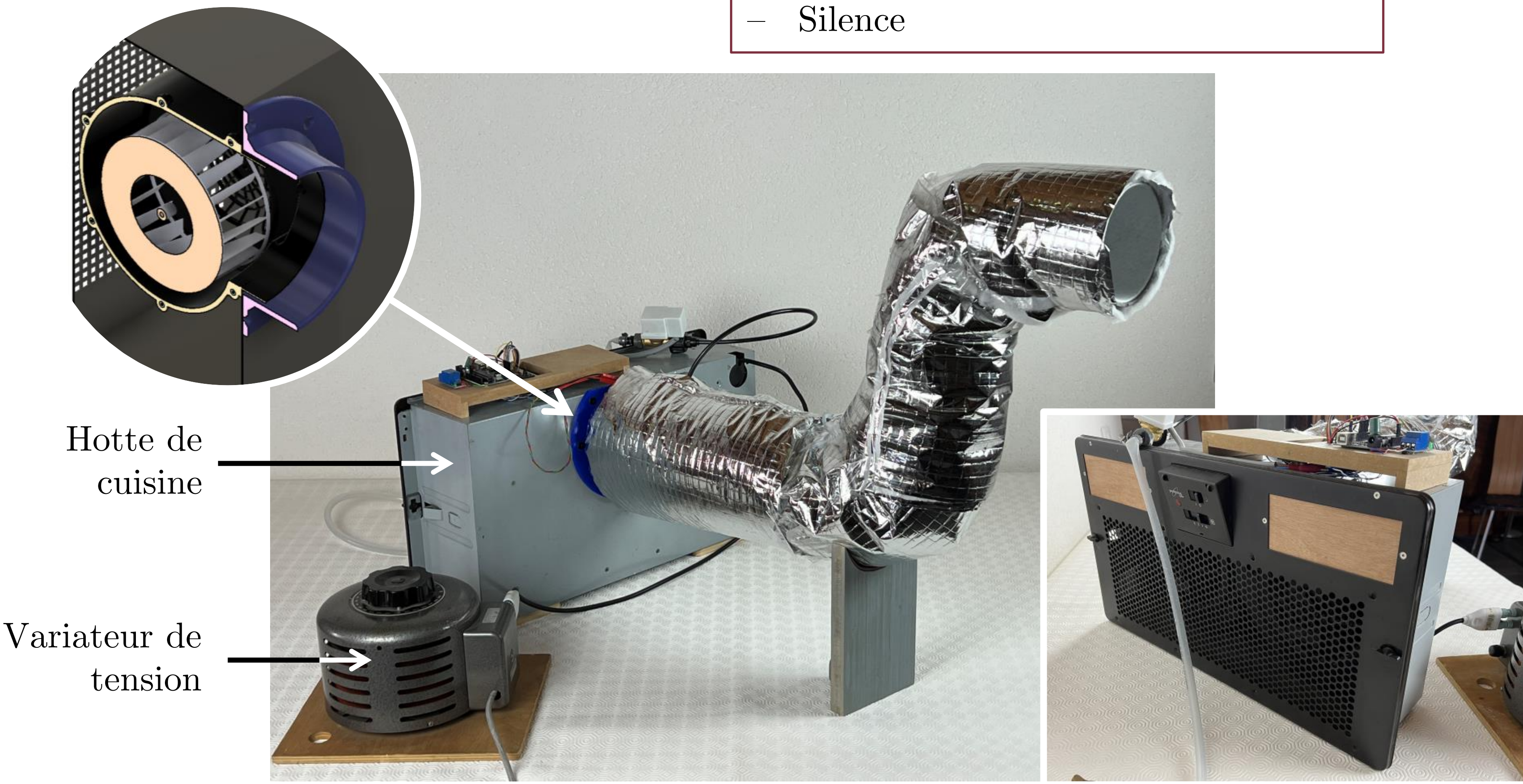



***Figure 4*** : maquette fabriquée

# 2 – MAQUETTE CONCEPTUELLE

## ETALONNAGE

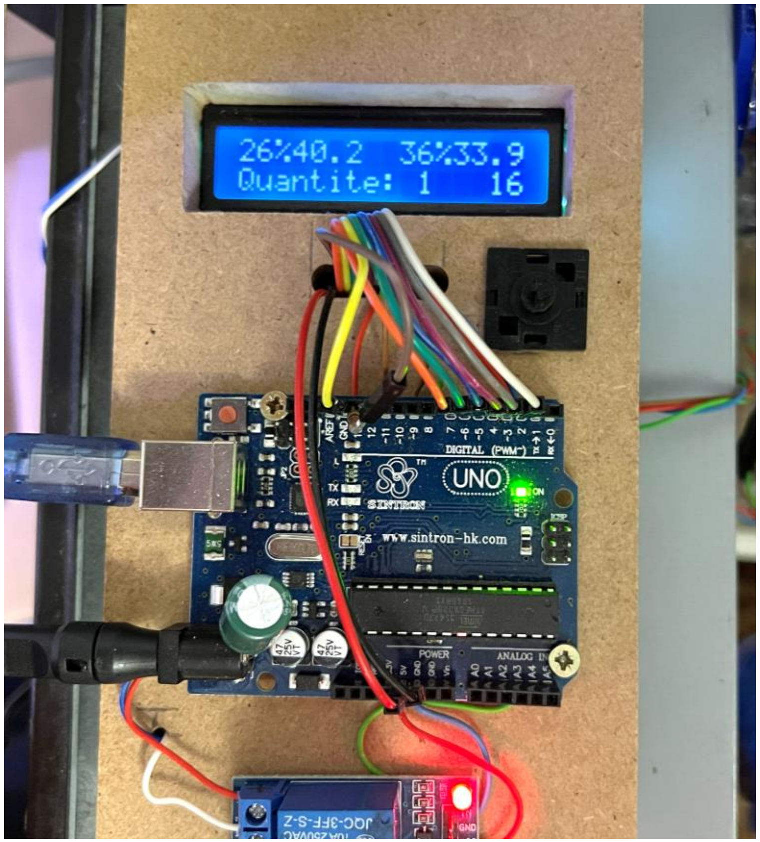
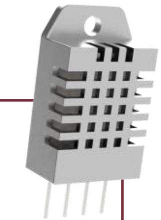


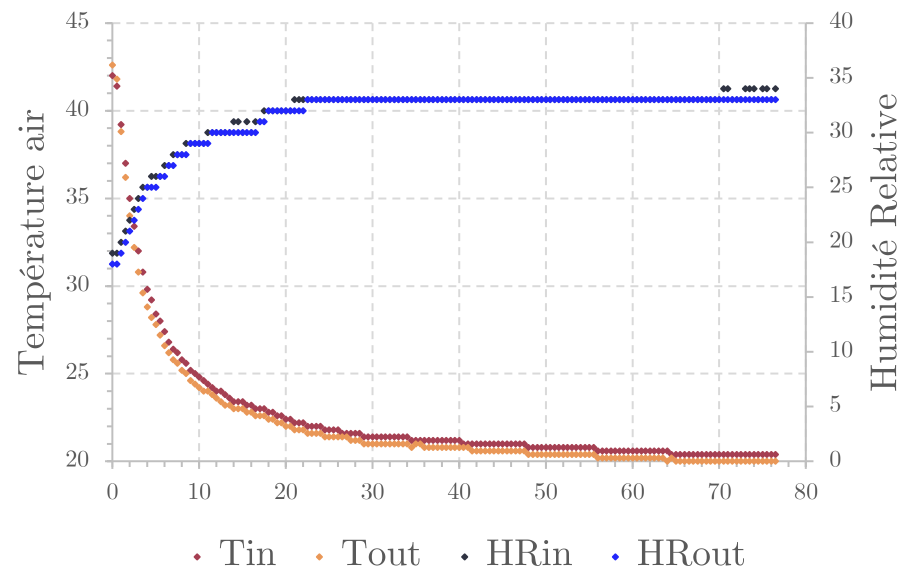

***Figure 5*** : montage *Arduino*

**Capteur *DHT22***



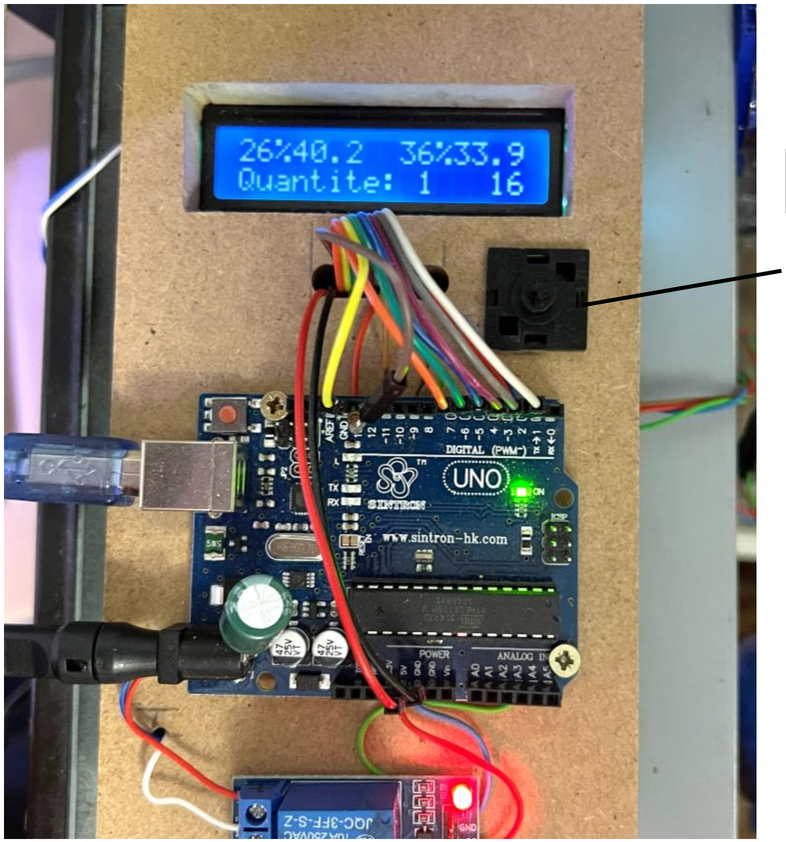
- <u>Alimentation</u> : 3,3V - 6V
- <u>*HR*</u> : [0% - 100%] ±2%
- <u>Température</u> : [-40°C ; +80°C] ±0,5°C
- <u>Période de mesure</u> : 2s

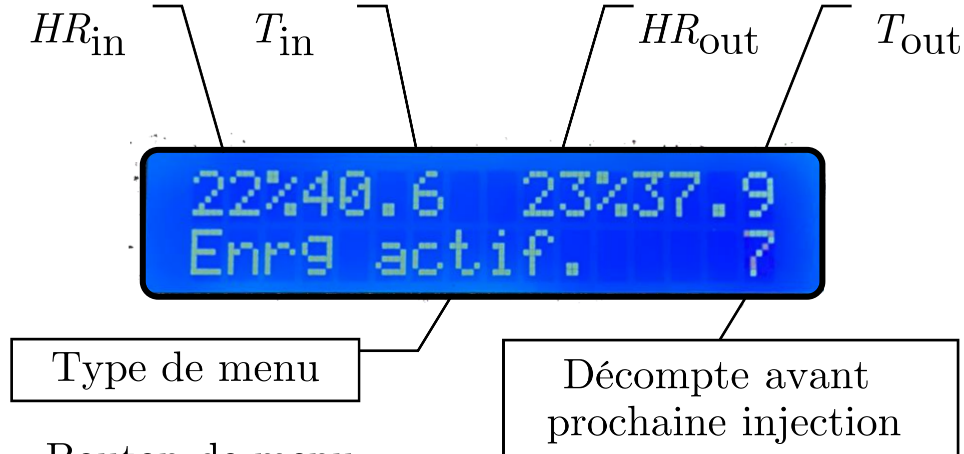
***Refroidissement du tunnel sous 1,5ms$^{-1}$***
***Départ 42°C***



# 2 – MAQUETTE CONCEPTUELLE

## CONTRÔLE



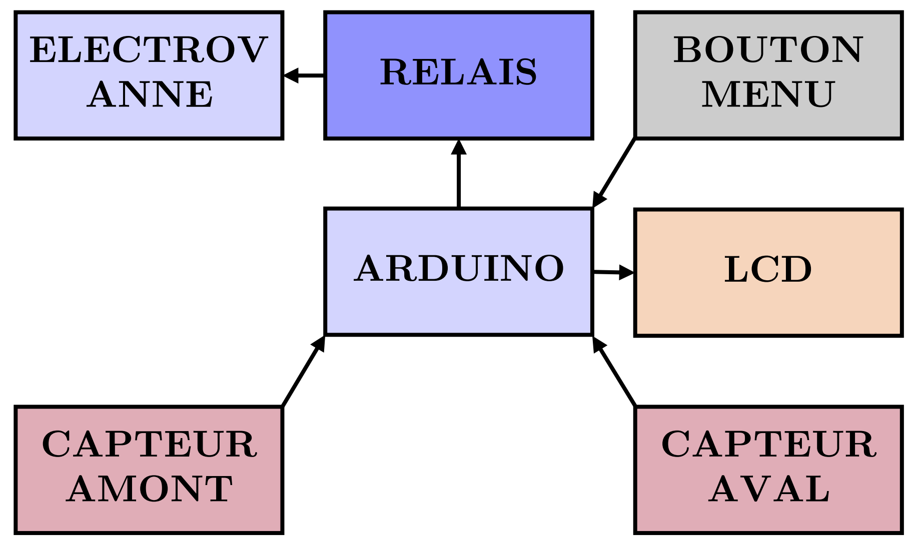




***Figure 5*** : montage *Arduino*

***Figure 6*** : schéma bloc global

# 2 – MAQUETTE CONCEPTUELLE

## STRUCTURE PROGRAMME

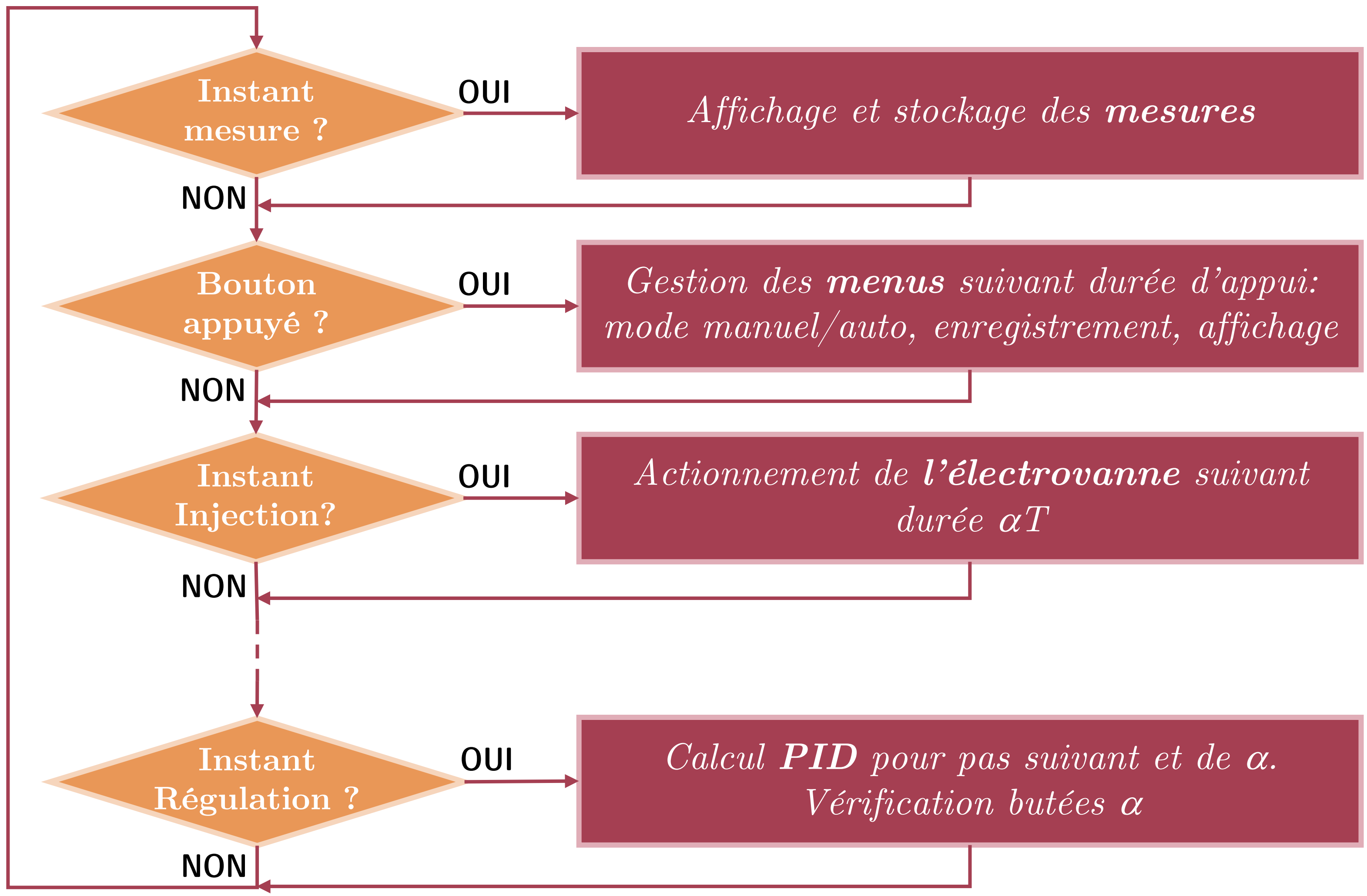
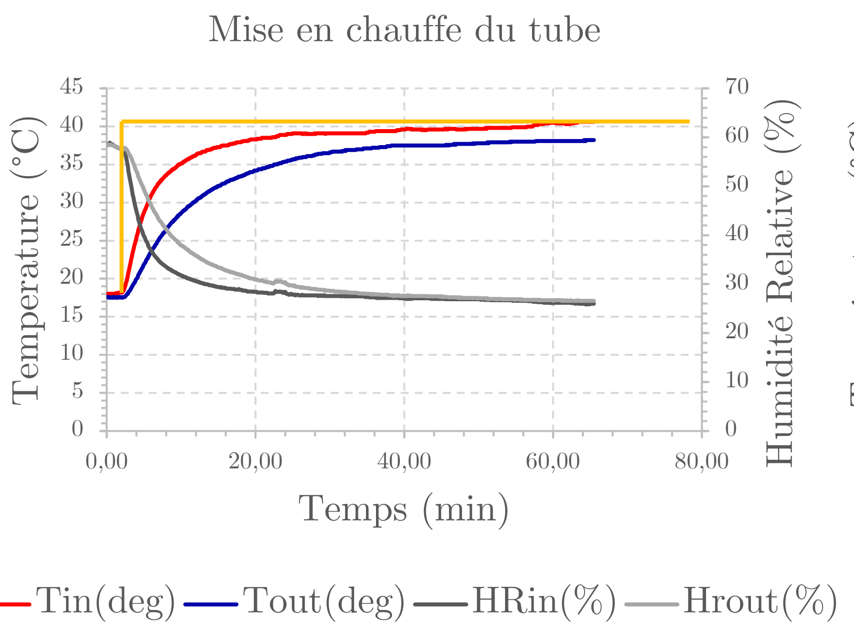



***Figure 7*** :
organigramme de programmation

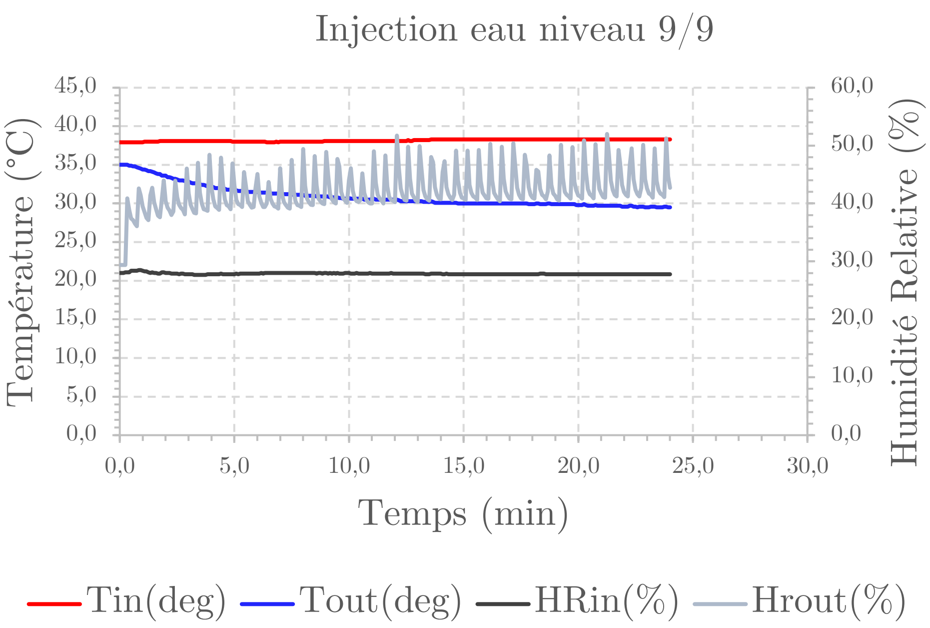
# 2 – MAQUETTE CONCEPTUELLE

## MESURE DU TAU

***Inertie thermique du système***
***Temps à 63%: env. 8 minutes***




***Injection eau***
***Temps à 63%: env. 8 minutes***




**Les mesures doivent durer au moins 25 minutes pour bien visualiser l'asymptote**

# 2 – MAQUETTE CONCEPTUELLE

THEORIE

➢ **Objectif initial:** Etablir une relation entre la densité de flux de molécules d'eau et la chute de température de l'air à l'interface.

Phénomène d'évaporation étudié depuis 150 ans et pas encore totalement compris

- Hertz, 1882
- Stefan, 1889
- Knudsen, 1915
- Kennard, 1938
- Knacke, 1956

# 2 – MAQUETTE CONCEPTUELLE

## THEORIE

➢ **Objectif initial:** Etablir une relation entre la densité de flux de molécules d'eau et la chute de température de l'air à l'interface.

Phénomène d'évaporation étudié depuis 150 ans et pas encore totalement compris

- Hertz, 1882
- Stefan, 1889
- Knudsen, 1915
- Kennard, 1938
- Knacke, 1956
- Wolff 2023 …

➢ **Stratégie:** Intuitivement, les grandeurs suivantes sont reliées.

- $\Delta T(Dm_{\text{eau}}\ ,\ Dm_{\text{air}})$ (1)
- $Dm_{\text{eau}}(P_{\text{part eau}})$ (2)

De plus, **certaines grandeurs dépendent de la température**, qui varie le long du tube. **Complexité.**

# 2 – MAQUETTE CONCEPTUELLE

## THEORIE

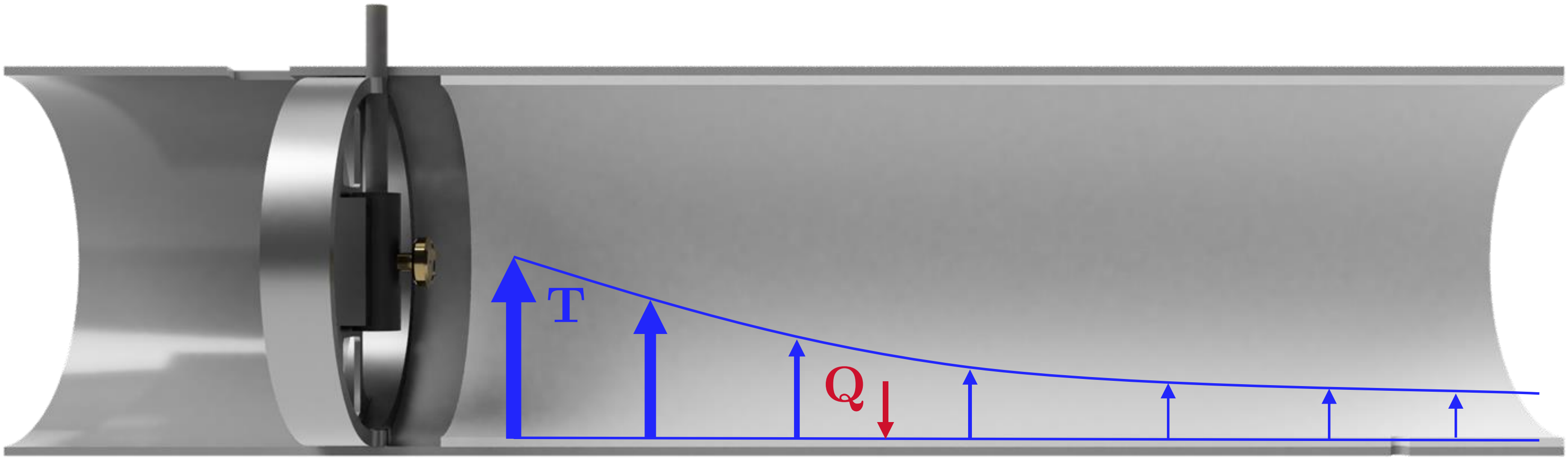



***Figure 8*** : Champ de température T le long du tube

$$\left.\begin{aligned} Q_{\text{air}} &= m_{\text{air}}.Cp_{air}.\Delta T \\ Q_{\text{eau}} &= m_{\text{eau}}\,\Delta h_{\text{ce}} \\ \underline{\text{Hyp}}:\ Q_{\text{air}} &= Q_{\text{eau}}\ (\text{transfert à } 100\%) \end{aligned}\right\}$$

$$\boxed{\Delta T(Dm_{\text{eau}}) = \frac{\Delta h_{\text{ce}} \cdot \boldsymbol{Dm}_{\textbf{eau}}}{Cp_{\text{air}} \cdot Dm_{\text{air}}} \qquad (1)}$$

$$\boxed{\Delta T(Dm_{\text{eau}}) = \mathrm{K}_1 \cdot \boldsymbol{Dm}_{\textbf{eau}} \qquad (2)}$$

# 2 – MAQUETTE CONCEPTUELLE

## THEORIE

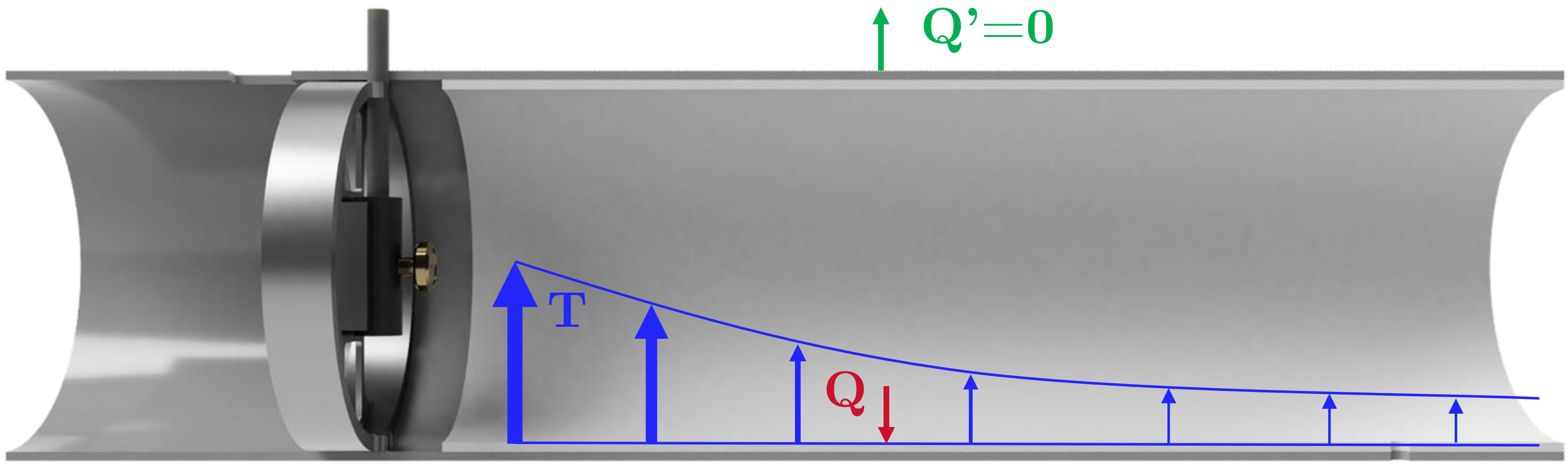



***Figure 8*** : Champ de température T le long du tube

$$Q_{\text{air}} = m_{\text{air}}.Cp_{air}.\Delta T$$

$$Q_{\text{eau}} = m_{\text{eau}}\ \Delta h_{\text{ce}}$$

<u>Hyp</u>: $Q_{\text{air}} = Q_{\text{eau}}$ (transfert à 100%)

$$\boxed{\Delta T(Dm_{\text{eau}}) = \frac{\Delta h_{\text{ce}} \cdot \boldsymbol{Dm_{\text{eau}}}}{Cp_{\text{air}} \cdot Dm_{\text{air}}} \qquad (1)}$$

$$\boxed{\Delta T(Dm_{\text{eau}}) = \mathrm{K}_1 \cdot \boldsymbol{Dm_{\text{eau}}} \qquad (2)}$$

Valable tant que $Dm_{\text{eau}}$ reste en dessous du maximum possible d'évaporation $\left(\ \boldsymbol{Dm_{\text{eau}}(T)}\ \right)_{\text{max}}$

# 2 – MAQUETTE CONCEPTUELLE

## THEORIE

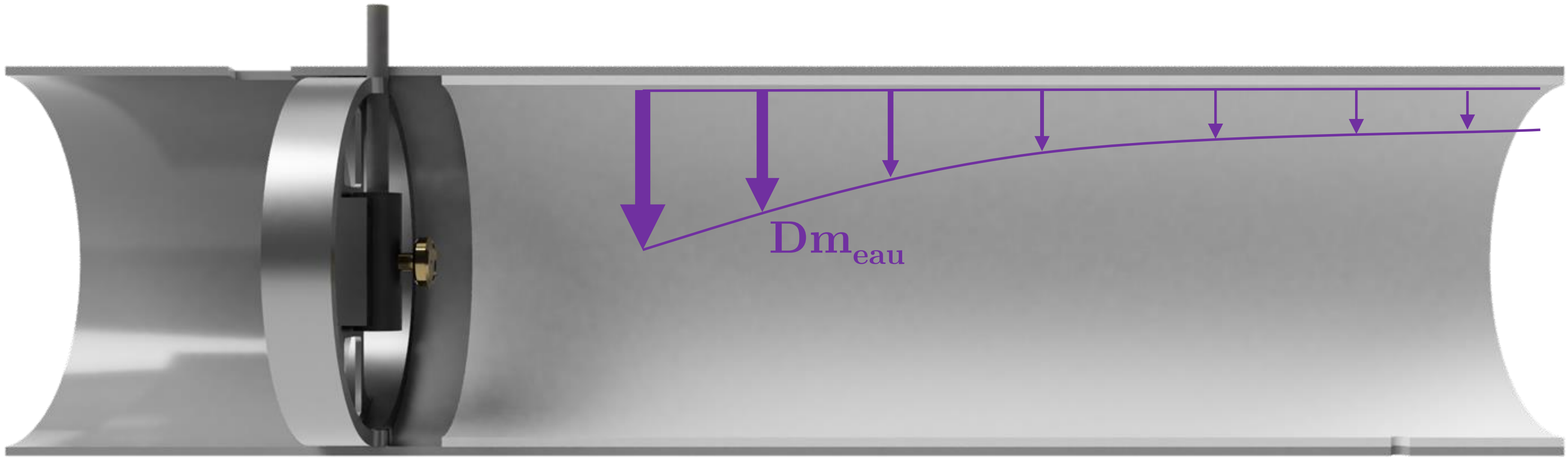



***Figure 10*** : Débit d'évaporation d'eau

Recherche documentaire : la meilleure relation adaptée est **Hertz-Knudsen :**

$$\boxed{\frac{(Dm_{eau}(T))_{max}}{S_{échange}} = a\sqrt{\frac{M}{2\cdot\pi R T}}(P_{sat} - P_{part\ eau}) \qquad (3)}$$

$$P_{part\ eau} = \mathrm{HR\%} \cdot P_{sat}$$

Rankine: $$\boxed{Ln(P_{sat}) = 13{,}7 - \frac{5120}{T} \qquad (4)}$$

## THEORIE

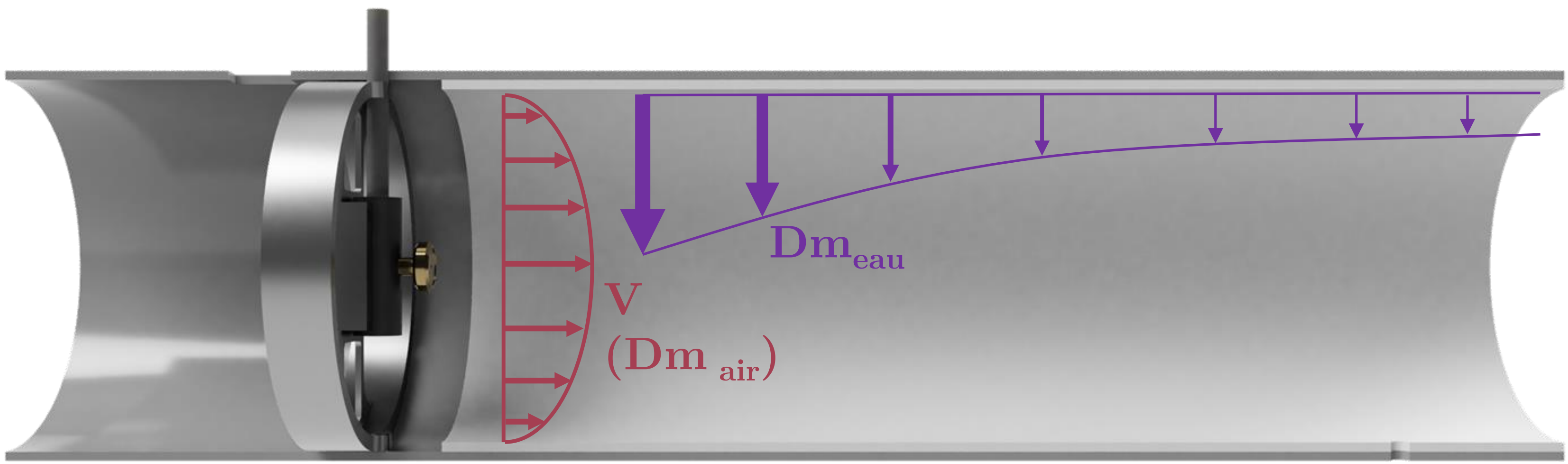



***Figure 11*** : Champ de vitesse (hypothèse laminaire) et débit massique d'évaporation

Recherche documentaire : la meilleure relation adaptée est **Hertz-Knudsen :**

$$\frac{(Dm_{eau}(T))_{max}}{S_{échange}} = a\sqrt{\frac{M}{2\cdot\pi R T}}(P_{sat} - P_{part\ eau}) \qquad (3)$$

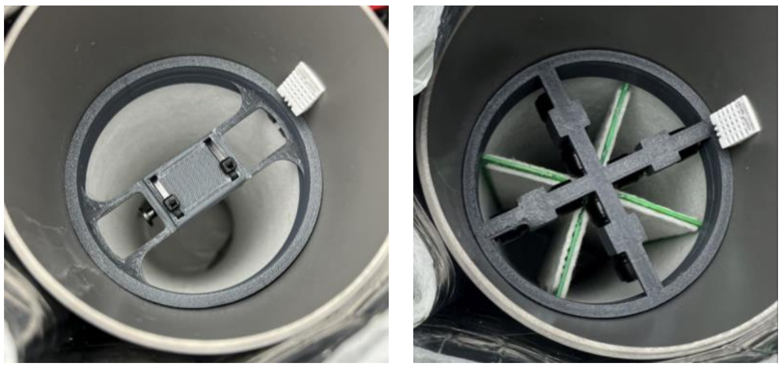
$$P_{part\ eau} = \mathrm{HR\%} \cdot P_{sat}$$

Rankine: $$Ln(P_{sat}) = 13{,}7 - \frac{5120}{T} \qquad (4)$$

**Complexité des phénomènes interdépendants**

# 2 – MAQUETTE CONCEPTUELLE

## FCT DE TRANSFERT



*Implantation dans la maquette*

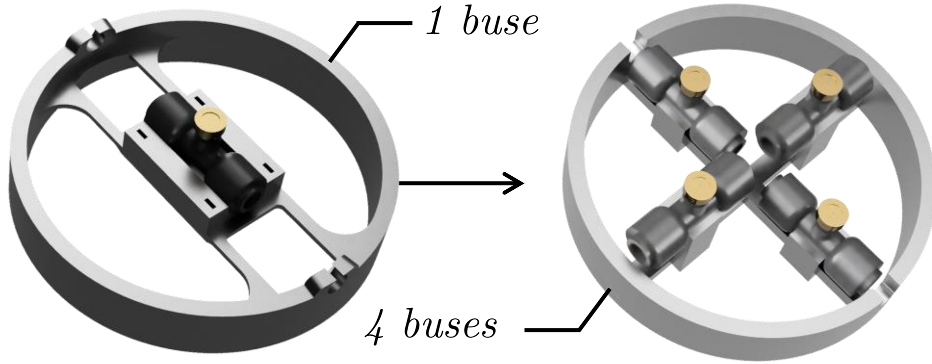
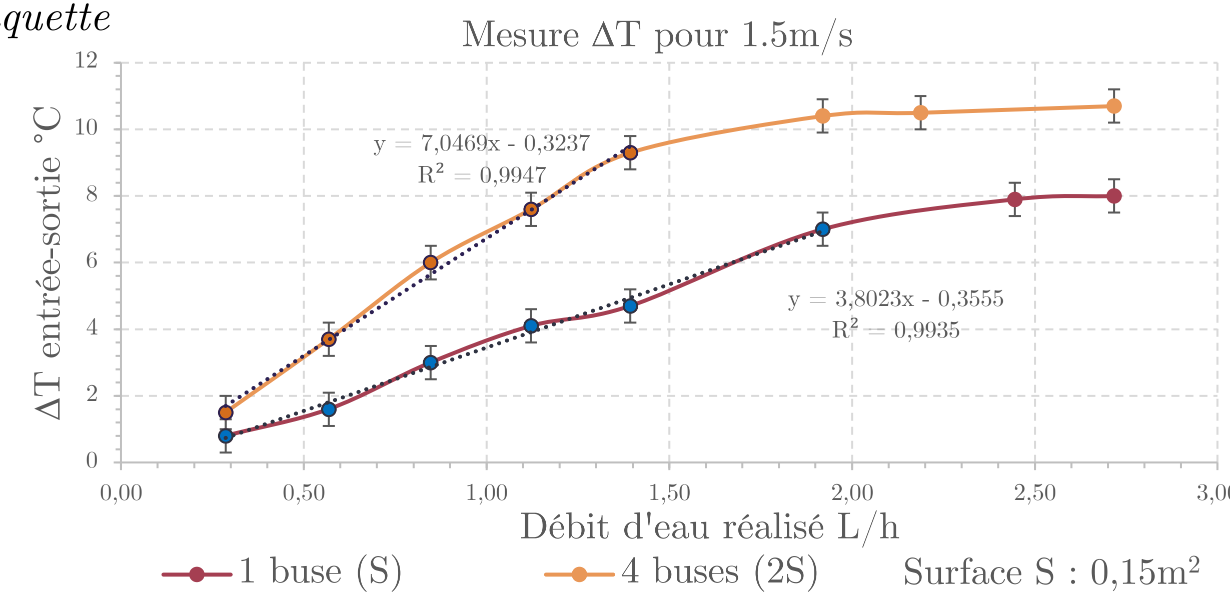



***Figure 12*** : Modélisation des supports de buses

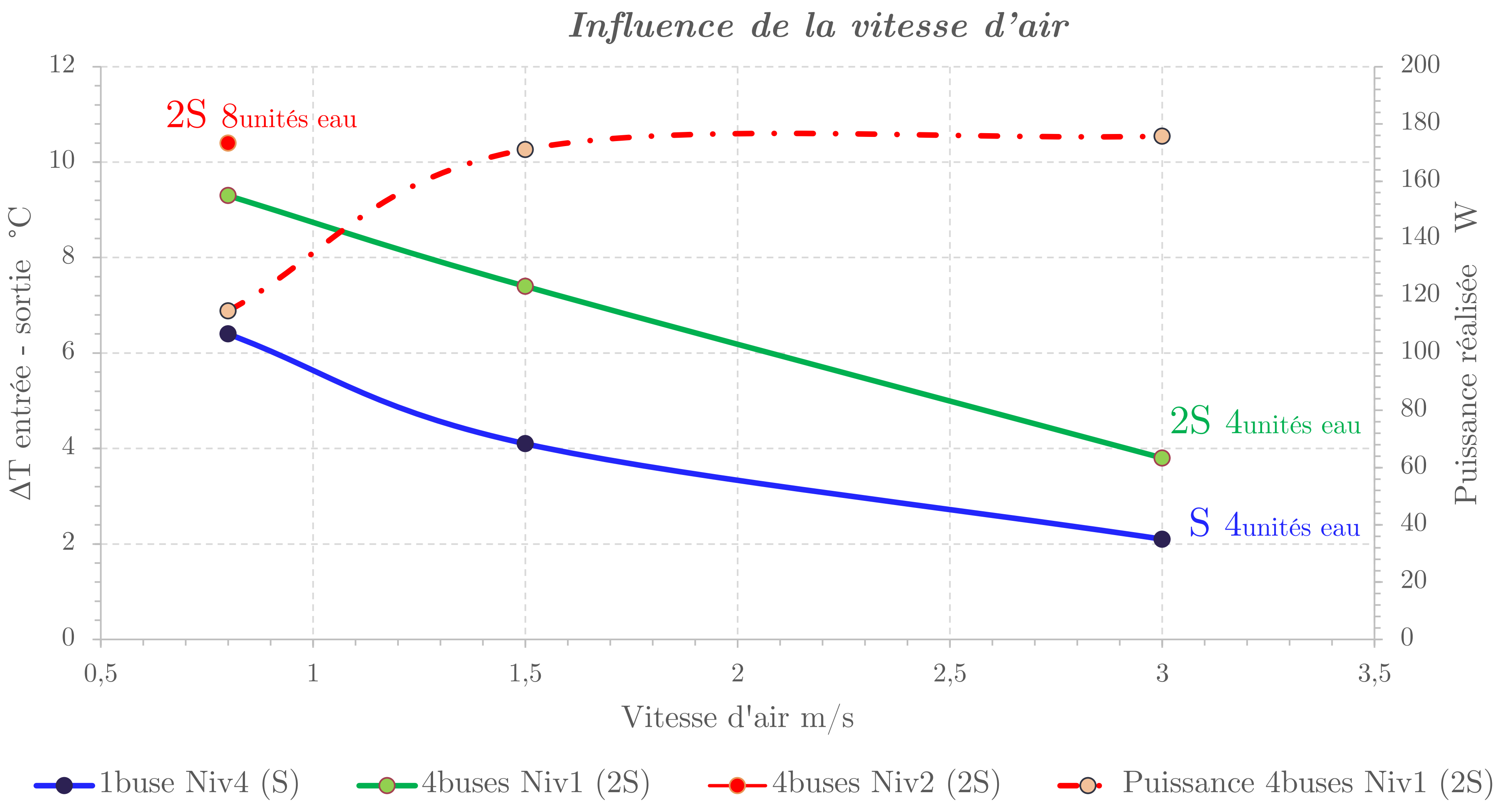
$$T(p) = \frac{3{,}8}{1+480p} \quad \text{(Surf. S)}$$

$$T(p) = \frac{7}{1+480p} \quad \text{(Surf. 2S)}$$



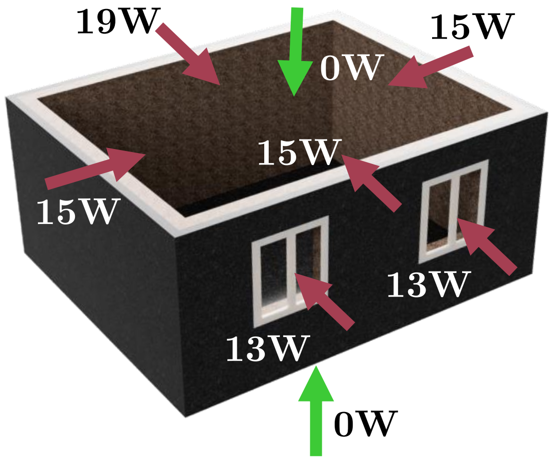
# 3 – RÉSULTATS ET ANALYSES

## ADEQUATION AU BESOIN ?



# 3 – RÉSULTATS ET ANALYSES

## ADEQUATION AU BESOIN ?



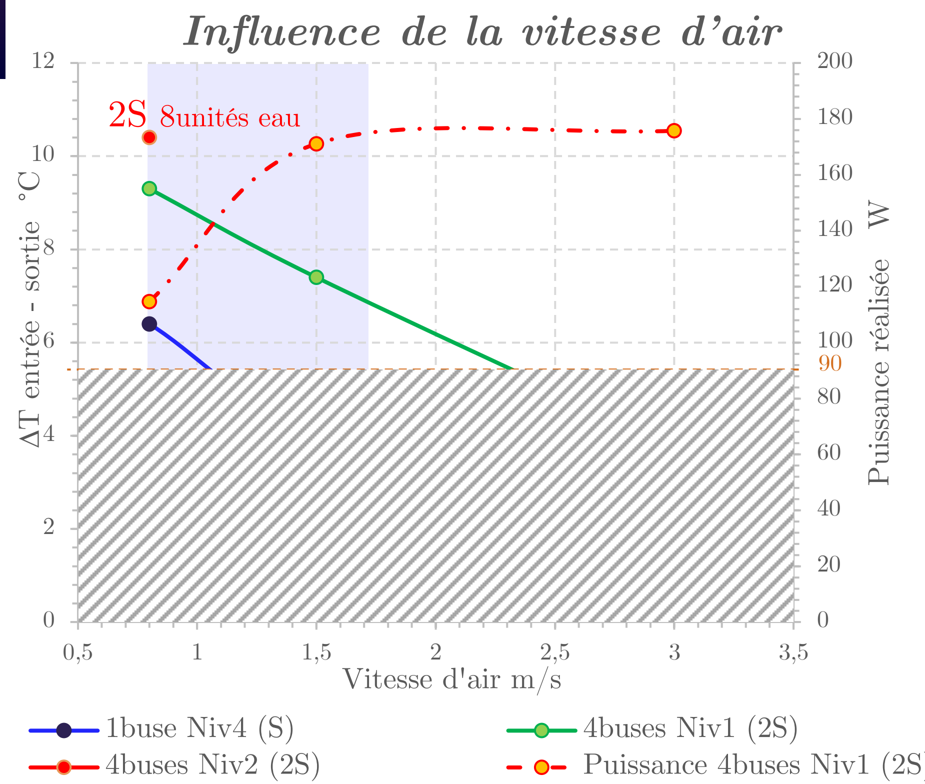

**Besoin**

Compenser les entrées d'une pièce de 20m²(CDC)

*Mur brique 20cm, λ=0,17 W/m.K*
*Isolant polystyrène 12cm, λ=0,022* W/m.K
*Fenêtre PVC,* $U_w$=*1,3*W/m²K
$T_{ext}=35°C$ , $T_{int}=25°C$

***Puissance entrante :*** $P_{ext}$***=90*****W**

**Surface 2S au minimum**




***MAIS***

***Puissance 1Personne : 80*****W**
**Equipements dissipateurs ...**

**Surface 2S insuffisante**

# 3 – RÉSULTATS ET ANALYSES

## FCT de TRANSFERT

***Blocs :***

- $K_{corr}$ : $K_p$=2,5 , $K_i$=0,03
- $K_{mli} = \frac{0{,}2\,.\,8}{30\,.\,255}$
- $K_{eau} = 43$ (L/h)
- $T(p) = \frac{7}{1+480p}$ (°C.h/L)
- $K_{capt} = 1$

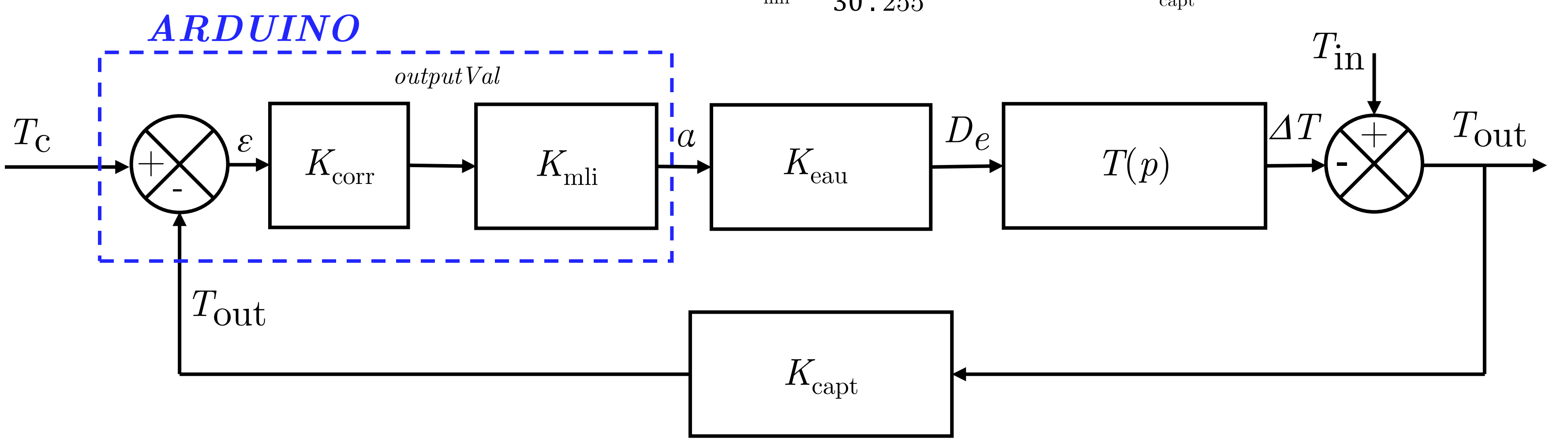



***Figure 13*** : schéma bloc du système de régulation

# 3 – RÉSULTATS ET ANALYSES

## FCT de TRANSFERT

***Blocs :***

- $K_{corr}$ : $K_p$=2,5 , $K_i$=0,03
- $K_{mli} = \dfrac{0{,}2\,.\,8}{30\,.\,255}$
- $K_{eau} = 43$ (L/h)
- $T(p) = \dfrac{7}{1+480p}$ (°C.h/L)
- $K_{capt} = 1$

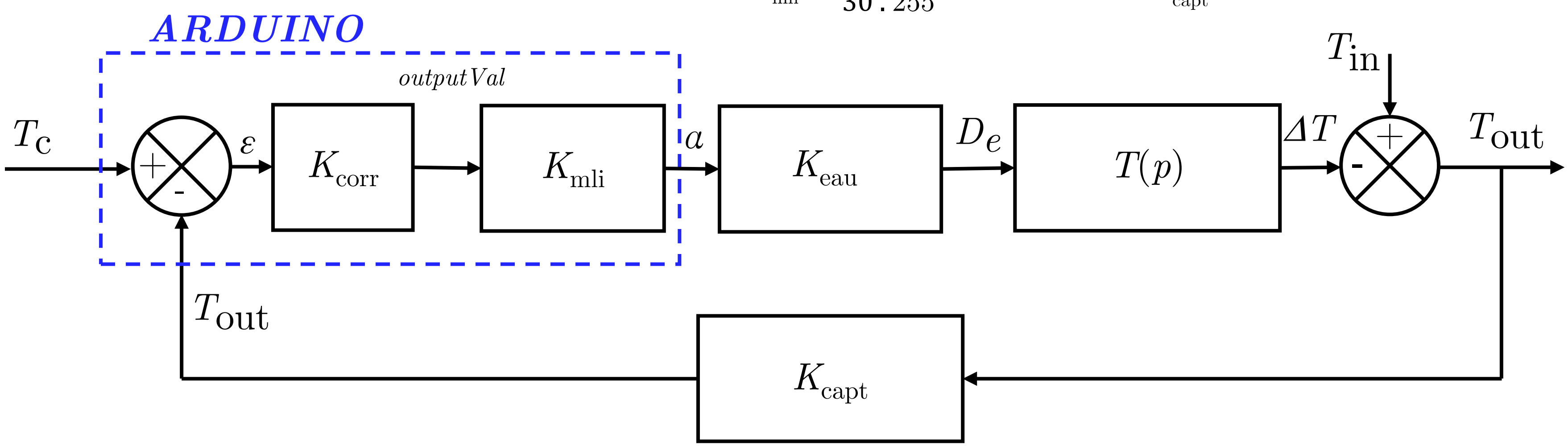
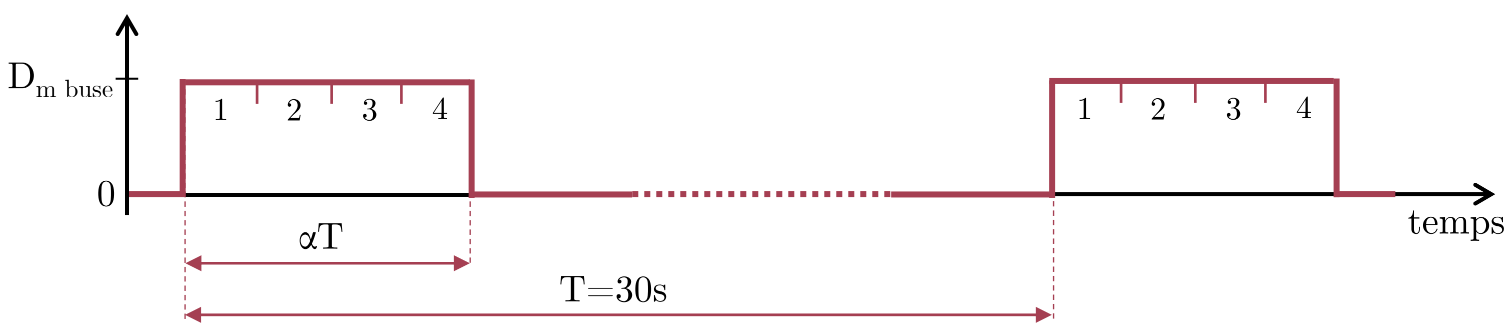


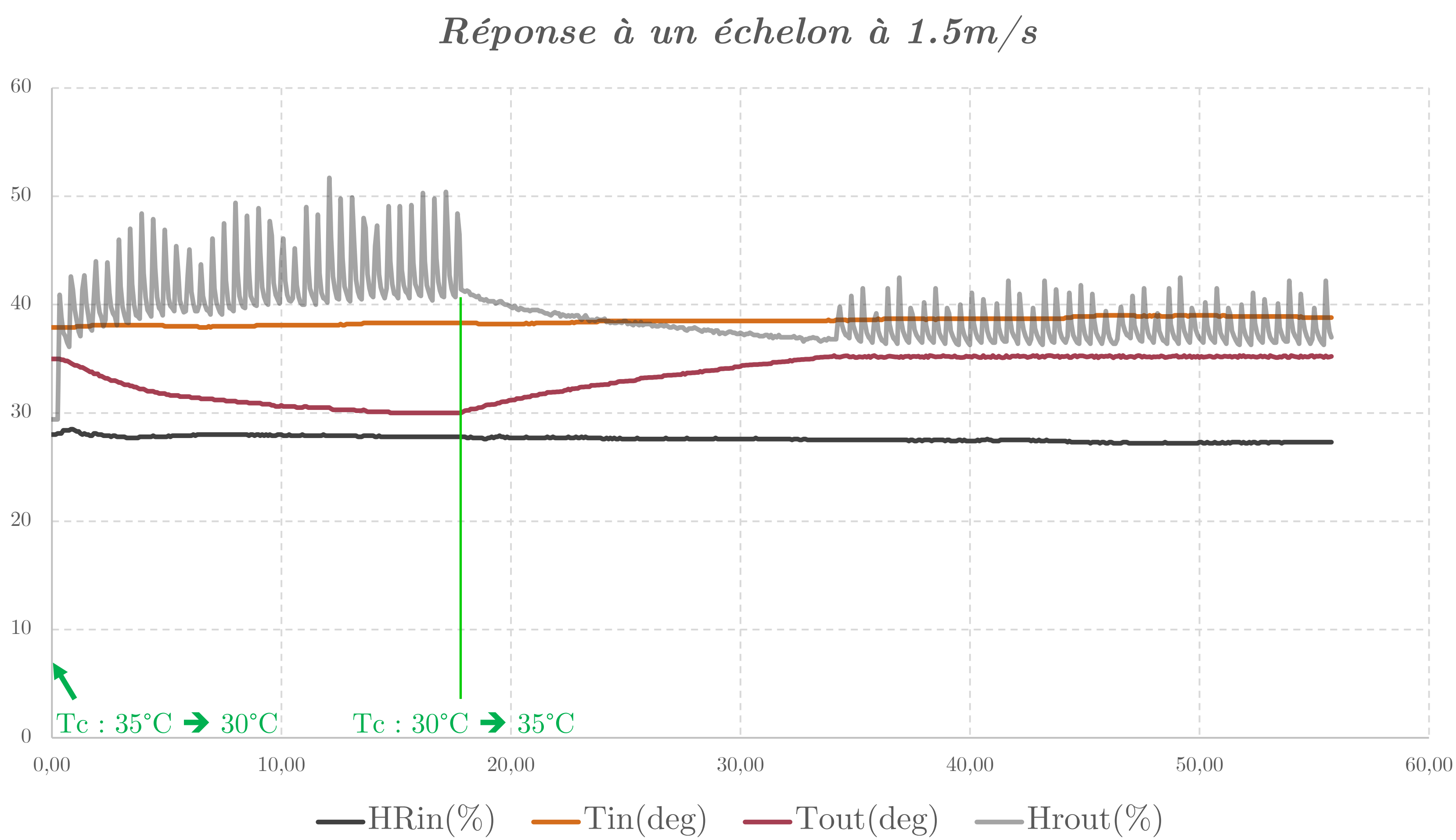

***Figure 13*** : schéma bloc du système de régulation

<u>Modélisation de la MLI</u> : exemple avec une *quantité* de 4 unités de 0,2s chacune, 8 unités max.



# 3 – RÉSULTATS ET ANALYSES

## REGULATION



# 3 – RÉSULTATS ET ANALYSES

## BILAN ENERGETIQUE

L'énergie provient du changement de phase de l'eau.

Le litre d'eau coute environ 0,3 cents. L'énergie équivalente à sa vaporisation couterait 12,5 cents en électrique.

Un bilan en puissance est très favorable:

$$\boxed{COP = \frac{P_{\text{thermo}}}{P_{\text{ventil}} + P_{\text{regul}}} \approx 1200\%}$$

Mais la puissance absolue dépend de la surface d'échange

Le COP d'une climatisation est de 500% au mieux.

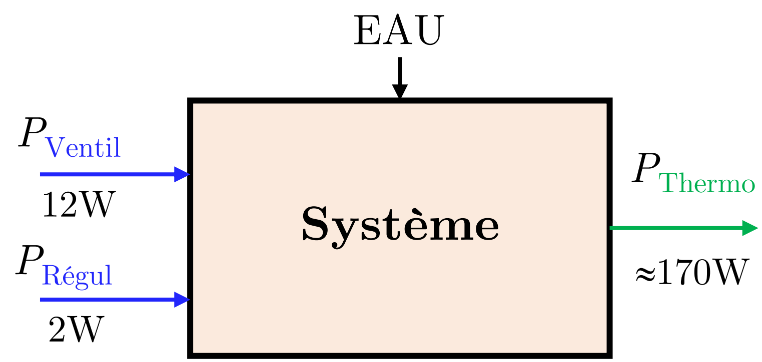



***Figure 14*** : Bilan de puissance de la machine

# 4 – CONCLUSION

## TECHNIQUE


| Id | Exigence | Critère | Niveau |
|---|---|---|---|
| 1 | Confort d'habitation en été | Surface de pièce | 20 $m^2$ |
| | | Régulation de la température intérieure | $< 25°C$ |
| | | Degré hygrométrique *NF35-102* | $HR_{\%} \in [50\ ;\ 70]$ |
| | | Précision : erreur statique | $\pm$ 1°C |
| 2 | Disposer d'un système compact | Taille du boitier | 300(L) × 300(l) × 1000(h) |
| | | Poids du dispositif | $<$ 10 kg |
| 3 | Consommation faible | Eau | $<$ 20 L/jour |
| | | Electricité | $<$ 50W |
| 4 | Bruit de fonctionnement | Niveau sonore à 1m | $<$ 42dBSPL |

# 4 – CONCLUSION

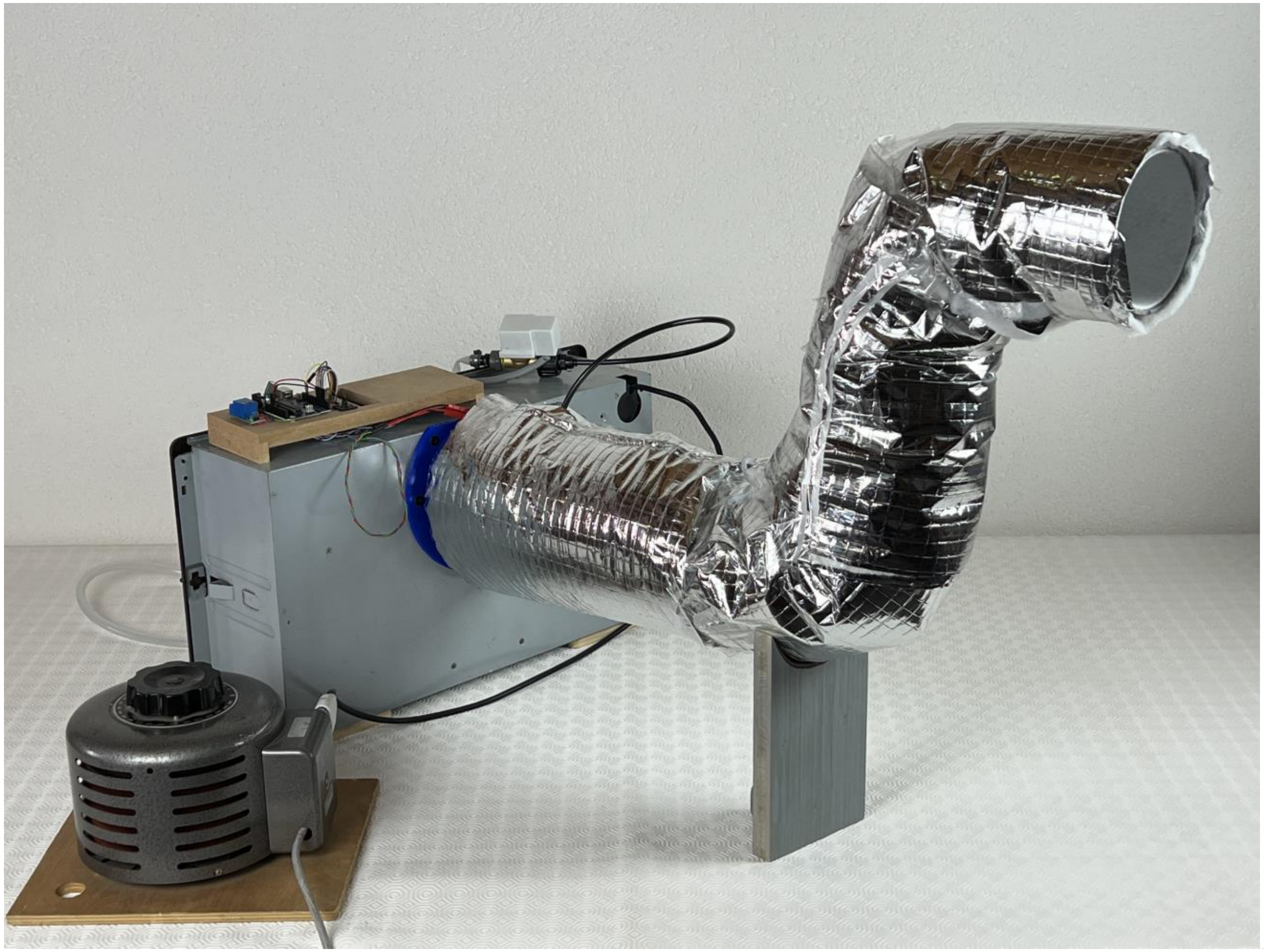
## TECHNIQUE

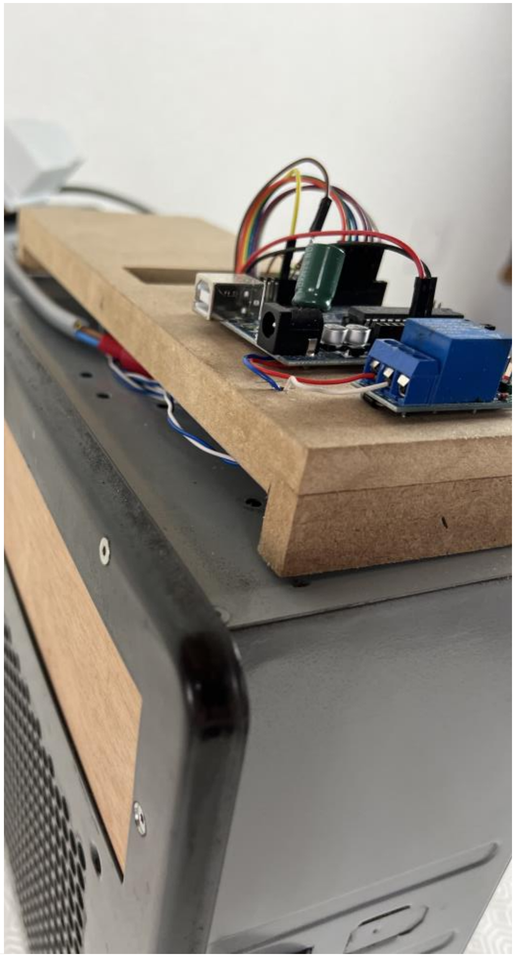

| Id | Exigence | Critère | Niveau |
|---|---|---|---|
| 1 | Confort d'habitation en été | Surface de pièce | 20 $m^2$ |
| | | Régulation de la température intérieure | $< 25°C$ |
| | | Degré hygrométrique *NF35-102* | $HR_{\%} \in [50\ ;\ 70]$ |
| | | Précision : erreur statique | $\pm$ 1°C |
| 2 | Disposer d'un système compact | Taille du boitier | 300(L) × 300(l) × 1000(h) |
| | | Poids du dispositif | $<$ 10 kg |
| 3 | Consommation faible | Eau | $<$ 20 L/jour |
| | | Electricité | $<$ 50W |
| 4 | Bruit de fonctionnement | Niveau sonore à 1m | $<$ 42dBSPL |

# 4 – CONCLUSION

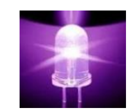
## BILAN

***Problématique*** *: par quels moyens peut-on disposer d'un système de climatisation efficace d'une pièce, compact et peu couteux ?*

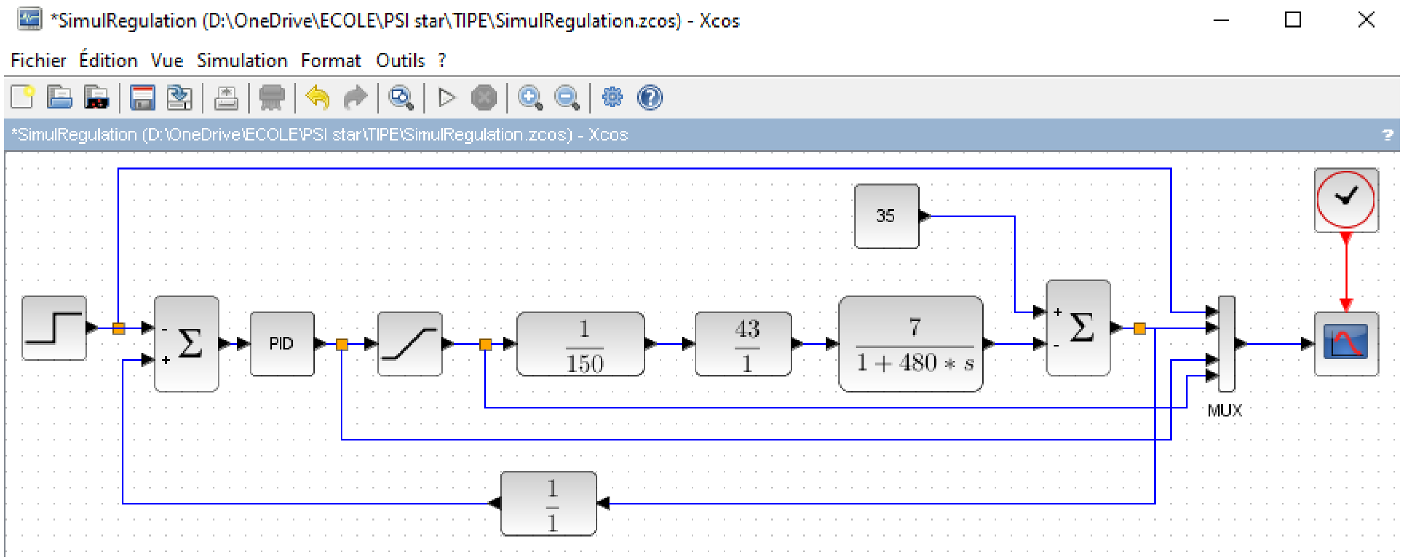


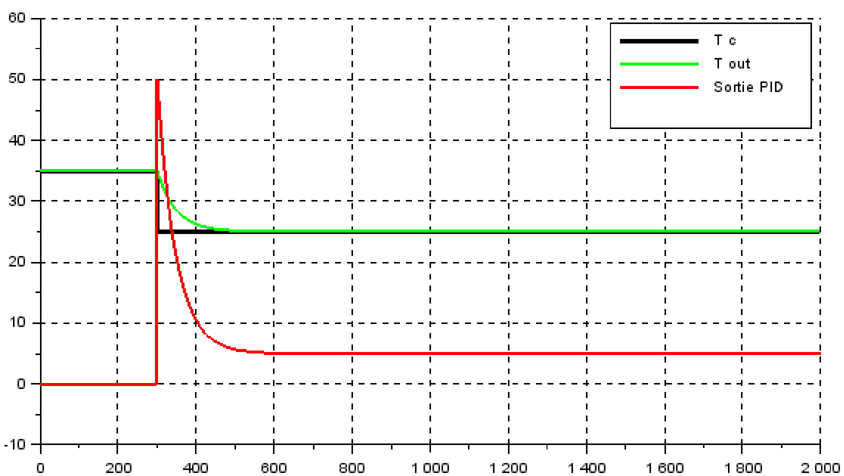


# ANNEXE 1 – AXES D'OPTIMISATION

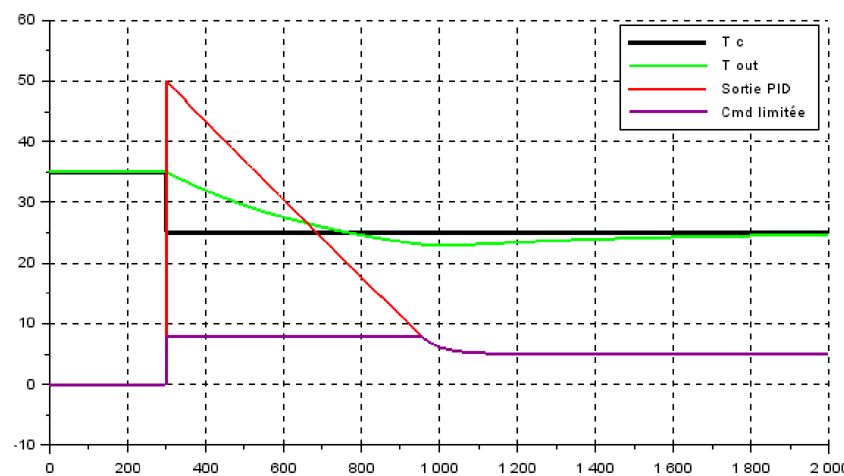
| Type | Nature | Solution | Illustration |
|---|---|---|---|
| Performance | Encombrement du produit | Plusieurs tubes en parallèle |  |
| | Taux d'évaporation baisse quand l'humidité augmente dans le tube | Tubes courts en parallèle | |
| | Dépôt de calcaire | Filtration à résine amont ou nettoyage périodique |  |
| | Puissance du ventilateur plus adaptée et aussi silencieux | Utilisation d'un ventilateur radial plus petit |  |
| | Utilisation de la fraicheur nocturne | Augmenter le débit d'air sans injection la nuit |  |
| Risque | Salmonelles | Lampe UV |  |
| Usage | Nécessité d'eau sous pression | Bac et écoulement par gravité | |
| | Entrée et sortie d'air de la pièce | Caches adaptables à une fenêtre entr'ouverte | |

# ANNEXE 2 – SIMULATIONS *SCILAB*




*PID sans saturation (limite à 8 unités)*




*PID avec saturation (limite à 8 unités)*

# ANNEXE 3 – DIAGRAMME AIR HUMIDE

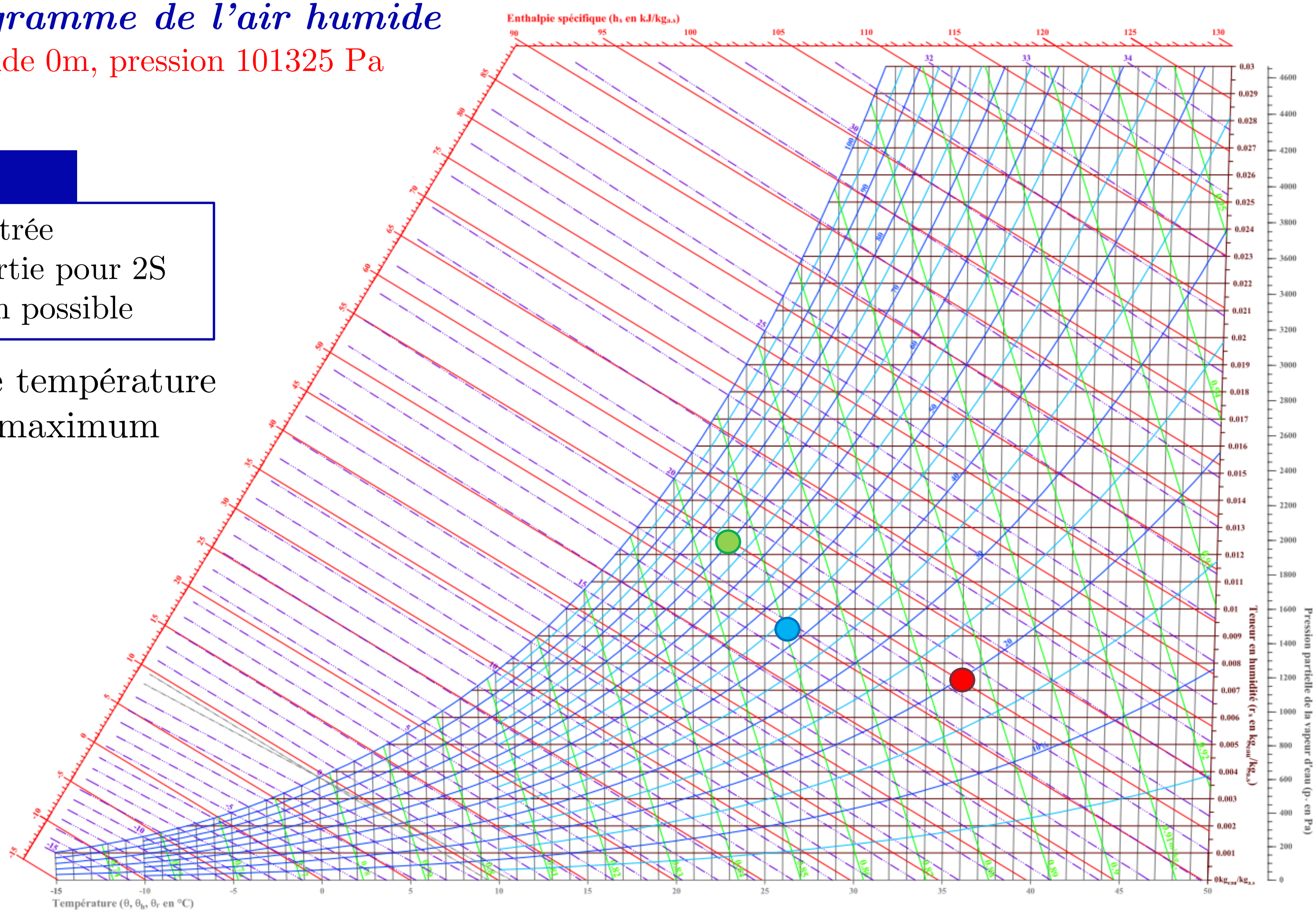
***Diagramme de l'air humide***

Altitude 0m, pression 101325 Pa

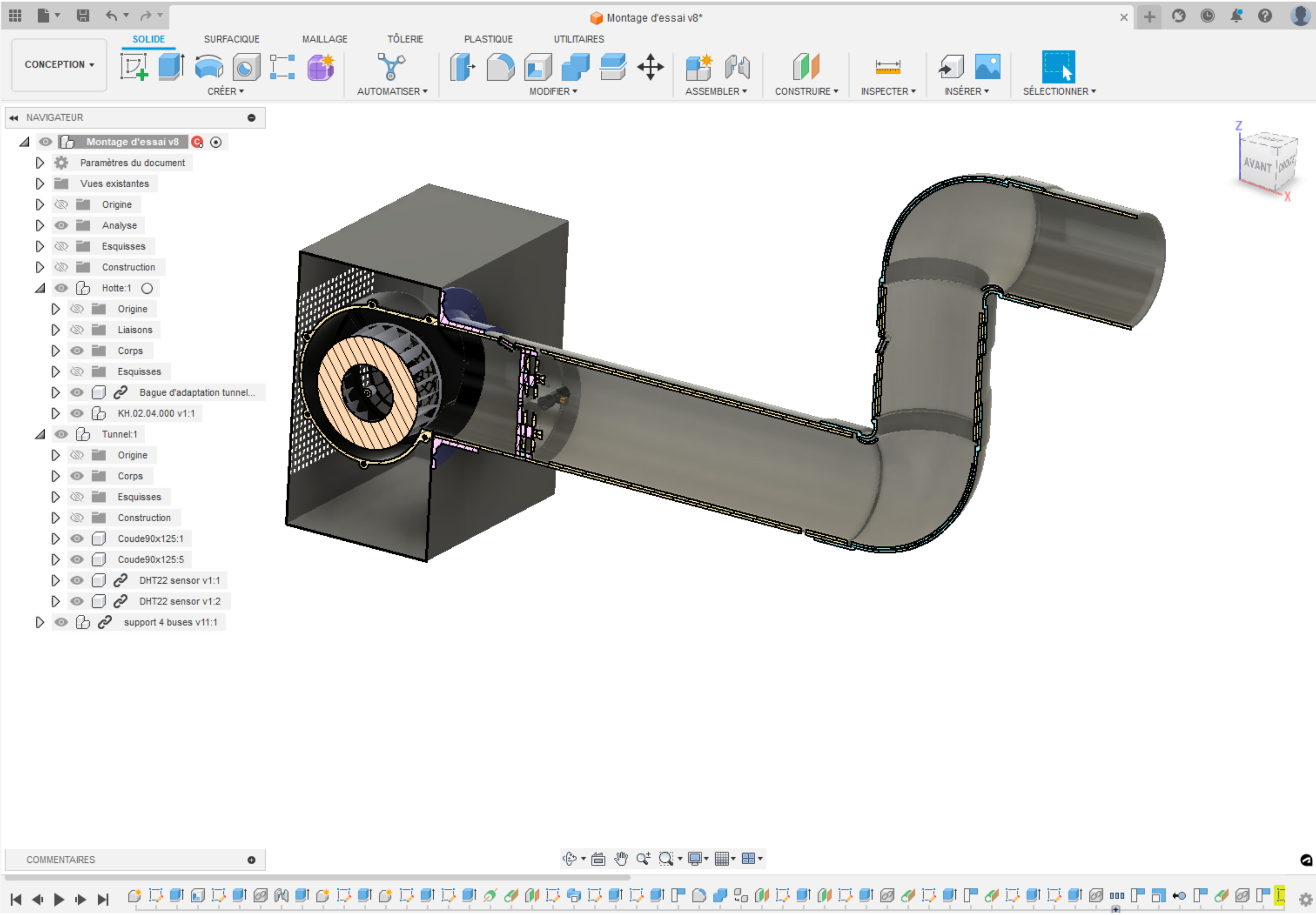
***Légende***

- Air en entrée
- Air en sortie pour 2S
- Maximum possible

Abaissement de température pas loin du maximum



# ANNEXE 4 – MODÉLISATION *FUSION360*



# ANNEXE 5 – CODE SOURCE POUR L'ARDUINO DE CONTROLE

```
/*
TIPE Air Conditioner based on water phase change

Version 1.0

  The circuit:
 * LCD RS pin to digital pin 7
 * LCD Enable pin to digital pin 6
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * LCD VSS pin to ground
 * LCD VCC pin to 5V
 * wiper to LCD VO pin (pin 3)
 * DHT1 DATA pin 8
 * DHT2 DATA pin 9
 * BUTTON pin 11
 * ELECTROVANNE pin 12
 * BACKLIGHT on/off pin 13

Pour ARDUINO UNO
 */
#include <EEPROM.h>
#include <LiquidCrystal.h>
#include <Stepper.h>
#include "DHT.h"
#include <AutoPID.h>


//#define DEBUG_TRACE     // debug with main informations
//#define DEBUG_VERBOSE   // more about details
//#define DEBUG_SIMUL     // If no sensor connected, do random values to test the code
#define DEBUG_CONSOLE     // During the measurement, display the values on the console at a different pace than
on the EEPROM
#define TRUE 1
```

```
#define FALSE 0
#define ON 1
#define OFF 0
#define MAX_QTITE 9    // maximum value for the "reglage" which is the index of water quantity. Over that, not
take into account
#define MAX_LCD_LINE 16
#define BACKLIGHT_ON LOW
#define BACKLIGHT_OFF HIGH
#define DEBOUNCE 50
#define T_INJECTION_UNITAIRE 200  // in ms
#define T_ATTENTE_INJECTION 30000 // in ms, unitary duration between two injections
#define T_MESURE 2500   // in ms time between two measurement read on DHT22 (supports 2sec minimum)
#define T_RAFRAICHISSEMENT_LCD 500
#define T_APPUI_COURT 100
#define T_APPUI_LONG  3000  // Duration to qualify a long press
#define T_APPUI_TRES_LONG 6000
#define T_APPUI_SUPER_LONG 9000
#define NB_MESURES_AVANT_STOCKAGE 12  // every N measurement, the values are stored in EEPROM
#define NB_MESURES_AVANT_PRINT_CONSOLE 2  // every N measurement, the values are printed on the console
#define T_SECONDE 1000  // One second
#define PAUSE_MEMO_DELAY 4000  // delay after which the value is stored in EEPROM. Should be less than led_delay
#define PAUSE_LED_DELAY  10000 // delay after which the backlight is swithed OFF
#define PAUSE_BLINK_ON 500
#define PAUSE_BLINK_OFF 500
#define BUTTON_PRESSED LOW
#define BUTTON_RELEASED HIGH
#define VANNE_ON  LOW
#define VANNE_OFF HIGH
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
#define COMMA 44  // the comma in ascii
#define CR  13  // carriage return
#define LF  10  // Line feed
#define EEPROM_START_OFFSET 1   // the BYTE 0 is used to store the water quantity index "reglage"
////////////////
// PARAMETERS
////////////////
//#define COMP_OFFSET                    // comment if the offset compensation must be disabled
#define ZERO_OFFSET_TEMPERATURE 15.5   // temperature at which the two sensors show the same value
#define OFFSET_TEMPERATURE  36.7       // wrong temperature where the offset is defined
#define OFFSET_VALUE 1.7               // ofset True value - Bad value
// formule: Treel = Tmes + ( OFFSET_VALUE / (OFFSET_TEMPERATURE-ZERO_OFFSET_TEMPERATURE) ) * ( Tmes-
ZERO_OFFSET_TEMPERATURE)

/////////////////////////
// ASSIGNATIONS OF PINS
```

```
////////////////////////
#define LCD_RS_PIN 7
#define LCD_EN_PIN 6
#define LCD_D4_PIN 5
#define LCD_D5_PIN 4
#define LCD_D6_PIN 3
#define LCD_D7_PIN 2
#define DHT1_PIN 8     // Digital pin connected to the DHT sensor
#define DHT2_PIN 9     // Digital pin connected to the DHT sensor
#define DHT3_PIN 10     // Digital pin connected to the DHT sensor
#define BUTTON_PIN 11  // For Menu and value change

#define VANNE_PIN 12   // for water inflow
#define BACKLIGHT_PIN 13

// initialize the sensor lib
// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 3 (on the right) of the sensor to GROUND (if your sensor has 3 pins)
// Connect pin 4 (on the right) of the sensor to GROUND and leave the pin 3 EMPTY (if your sensor has 4 pins)
DHT dht1(DHT1_PIN, DHTTYPE);
DHT dht2(DHT2_PIN, DHTTYPE);
// DHT dht3(DHT3_PIN, DHTTYPE);
// The process has long time characteristics (TAU= 8 minutes). No need to update too fast for PID
// Code source for PID: https://reference.arduino.cc/reference/en/libraries/autopid/

#define PID_UPDATE_DELAY 10000   // regulation recalculation interval.

//pid settings and gains
#define OUTPUT_MIN 0
#define OUTPUT_MAX 255
#define KP 5
#define KI .01
#define KD 0
#define REGLAGE_INJECTION_MIN 0   // Minimum water injection units of unitary time
#define REGLAGE_INJECTION_MAX 8   // Maximum value corresponding to the exchange surface to avoid flood
#define MANUAL 0
#define AUTO 1
#define TARGET_OUTPUT_TEMP 25     // So far fixed value for tests.

double t1,t2,h1,h2,setPoint,outputVal;

//input/output variables passed by reference, so they are updated automatically
```

```
AutoPID myPID(&t1, &setPoint, &outputVal, OUTPUT_MIN, OUTPUT_MAX, KP, KI, KD);
unsigned long lastTempUpdate; //tracks clock time of last temp update

// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
// Code source for LCD: https://docs.arduino.cc/learn/electronics/lcd-displays

LiquidCrystal lcd(LCD_RS_PIN, LCD_EN_PIN, LCD_D4_PIN, LCD_D5_PIN, LCD_D6_PIN, LCD_D7_PIN);


////////////
// VARIABLES
////////////
int reglage = 0;                       // Default value of water quantity is NONE
int duree_injection = 0;
char niveau_precedent_inc = BUTTON_RELEASED;  // to detect a edge of button press
unsigned long temps_aff = 0;       // Time reference from which the different functions estimate the passed time
to trigger the execution
unsigned long temps_mesure = 0;    // Measurement
unsigned long temps_memo = 0;      // "reglage" value stored in EEPROM
unsigned long temps_distri = 0;    // water injection
unsigned long temps_blink = 0;     // blink pace of black square
unsigned long temps_appui_long = 0;     // long press
unsigned long temps_aff_lcd = 0;  // LCD value writing to avoid massive writes in the main loop
enum t_appui {AUCUN,COURT,LONG,TRES_LONG,SUPER_LONG};  // Button press types
t_appui type_appui = AUCUN;
unsigned long decompte = T_ATTENTE_INJECTION;  // countdown between injection
char blink_state = ON;
char afficheur_eteint=TRUE;
char vanne_ouverte = FALSE;
char valeur_memorisee = TRUE;
char valeur_affichee = FALSE;
char injection_eau = FALSE ;
byte probleme_capteurs = FALSE;
byte go_record = FALSE;
byte regulation_mode = MANUAL;
int eeprom_current_address = EEPROM_START_OFFSET ;
int nb_mesures = 0;
byte nb_mesures_stockees = 0;

//////////////
// FUNCTIONS
//////////////

void Display_valeur(int valeur){
```

```
  //debug=valeur;
  if (valeur > MAX_QTITE) valeur=0;  // if out of allowed values range, display a 0 (abnormal)
  digitalWrite(BACKLIGHT_PIN,BACKLIGHT_ON);  // switch backlight LCD on
  lcd.setCursor(10,1);
  lcd.print(valeur);
  afficheur_eteint = FALSE;
  valeur_affichee = TRUE;
  // new time reference for the backlight
  temps_aff = millis();
        #if defined(DEBUG_TRACE)
          Serial.print("Display: ");
          Serial.println(valeur);
        #endif
}

void Display_sensors(float HRin, float Tin, float HRout, float Tout)
{
  String FirstLineString = "";
  digitalWrite(BACKLIGHT_PIN,BACKLIGHT_ON);
  // new time reference for the backlight
  temps_aff = millis();
  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  if (isnan(HRin) || isnan(Tin) || isnan(HRout) || isnan(Tout))
  {
    FirstLineString+="Err capteurs";
        #if defined(DEBUG_TRACE)
          Serial.println(F("Failed to read from DHT sensors!"));
        #endif
  }
  else
  {
      FirstLineString+=String(int(HRin))+ "%" + String(Tin,1)+ "  " + String(int(HRout))+ "%" + String(Tout,1) ;
  }
  lcd.setCursor(0,0);
  lcd.print(FirstLineString);
  afficheur_eteint = FALSE;
        #if defined(DEBUG_VERBOSE)
          Serial.print("FirstLineString: ");
          Serial.println(FirstLineString);
        #endif
}

void Display_message(String msg, byte x, byte y)
{
```

```
  if ((msg.length()> MAX_LCD_LINE-x) || (y>1) ) msg="ERREUR";  // the string is too long or out of the screen
coordinates
                                                   // if the string may overwrite the water injection value, go to second
line
  if ( ((msg.length()+x) > 9) && (y==1) ) valeur_affichee = FALSE;
  digitalWrite(BACKLIGHT_PIN,BACKLIGHT_ON);
  //lcd.clear();
  lcd.setCursor(x,y);
  lcd.print(msg);
  afficheur_eteint = FALSE;
  temps_aff = millis();
}

void Display_square(char state){
  //debug=state;
    lcd.setCursor(11,1);
    if (state == ON) lcd.print((char)255);
    if (state == OFF) lcd.print(" ");
  }

void Display_countdown(void)
{
    int tmp=0;
    tmp=(T_ATTENTE_INJECTION-(millis()-temps_distri))/1000;
    lcd.setCursor(14,1);
    if(tmp>9)
    {
      lcd.print(tmp);
    }
    else
    {
      lcd.print(" ");
      lcd.print(tmp);
    }
}

void Eeprom_to_console(void)
{
     int addr = EEPROM_START_OFFSET;
     int i=0;
     Serial.println(F("Programme " __FILE__ " le " __DATE__ "\r\n"));
     Serial.print("Instants espacés de (s): ");
     Serial.println(String(NB_MESURES_AVANT_STOCKAGE*T_MESURE/1000));
     Serial.println("Mesures: ");
     Serial.println("Instant %HRin Tin %HRout Tout");
```

```
        #if defined(DEBUG_VERBOSE)
          Serial.println("addr=" + String(addr) + " Length=" + String(EEPROM.length()));
          for(i=0;i<26;i++)
          {
             Serial.print(EEPROM.read(addr+i));
             Serial.print(",");
          }
          Serial.println("");
          i=0;
        #endif
    while ((EEPROM.read(addr)!=0) && (addr < EEPROM.length()))
    {             // after each group of values, there may be an end of record NULL
         if(i==5)
         {        // end of the bloc of 5 BYTES
             Serial.println("");     // next line printed
             i=0;
         }
         if (i==2 || i==4)
         {   // position 2 and 4: temperature coded one one byte multiplied by 5 for less precision loss
           Serial.print(String(EEPROM.read(addr)/5.0,1));
         }
         else
         {   // other values written with no change
           Serial.print(String(EEPROM.read(addr)));
         }
           Serial.print(",");
           addr++;
           i++;
           delay(20);
    }
    Serial.println("");
}

void Values_to_eeprom(byte instant, float HRin, float Tin, float HRout, float Tout)
{
      byte hr1, temp1, hr2, temp2;
      // encoding the FLOAT into ONE BYTE
      hr1=(byte)(int)(HRin+.5); // round to the near integer value, then cast to one BYTE
      temp1=(byte)(int)(Tin+.5);
      hr2=(byte)(int)(HRout+.5);
      temp2=(byte)(int)(Tout+.5);
      if( (eeprom_current_address + 5) < EEPROM.length())
      {   // there are still places for one set of data
              EEPROM.update(eeprom_current_address, instant);
              eeprom_current_address++;
```

```
                EEPROM.update(eeprom_current_address, hr1);
                eeprom_current_address++;
                EEPROM.update(eeprom_current_address, temp1);
                eeprom_current_address++;
                EEPROM.update(eeprom_current_address, hr2);
                eeprom_current_address++;
                EEPROM.update(eeprom_current_address, temp2);
                eeprom_current_address++;
                   #if defined(DEBUG_VERBOSE)
                        Serial.println("Next addr=" + String(eeprom_current_address));
                   #endif
      }
      else
      {   // stop needed
                   #if defined(DEBUG_TRACE)
                        Serial.println(F("EEPROM pleine. On arrete..."));
                   #endif
              if(eeprom_current_address = EEPROM.length())
              { // no more storage available
                EEPROM.update(eeprom_current_address-1, 0); // overwrite the last measurement to end the record
with NULL
              }
              else
              { // still one place free
                EEPROM.update(eeprom_current_address, 0);   // ending the record with NULL
              }
              go_record = FALSE;  // and we stop writing to the EEPROM
              Display_message("Stop Enreg",0,1);
              nb_mesures=0;
      }
}

void check_button(void)
{
  // When the button is pressed, LCD backlight is switched ON and the current "reglage" is displayed
  // To increment the reglage, a second press is needed. As long as the button is pressed, skip this test
  // Next press with LCD ON does increment the value, and reaching MAX_QTITE loops to 0
  // LONG press on the button shows successively other options (Menu)

  // ACTION WHEN THE BUTTON IS PRESSED

  if((niveau_precedent_inc == BUTTON_RELEASED) && digitalRead(BUTTON_PIN) == BUTTON_PRESSED )
  {   // button now pressed
    delay(DEBOUNCE);  // waiting some short ms to confirm the button is still in same state, to avoid a bounce
    if(digitalRead(BUTTON_PIN) == BUTTON_PRESSED)
```

```
    {
        temps_appui_long=millis();  // réference de temps pour qualifier l'appui long
        niveau_precedent_inc = BUTTON_PRESSED;
            #if defined(DEBUG_TRACE)
              Serial.println("appui");
            #endif
    }
  }

  // ACTIONS ASSOCIATED TO THE BUTTON RELEASE TIME (AFTER LONG PRESS)

  if((niveau_precedent_inc == BUTTON_PRESSED) && digitalRead(BUTTON_PIN) == BUTTON_RELEASED )
  {   // button now released
      delay(DEBOUNCE);  // waiting some short ms to confirm the button is still in same state, to avoid a bounce
      if(digitalRead(BUTTON_PIN) == BUTTON_RELEASED)
      { // confirmed ! Now we qualify the press type based on passed time
          niveau_precedent_inc = BUTTON_RELEASED;
            #if defined(DEBUG_TRACE)
              Serial.print("lache (ms): ");
              Serial.println(String(millis()-temps_appui_long));
            #endif
          if((millis()-temps_appui_long) > T_APPUI_SUPER_LONG)
          {
             type_appui = SUPER_LONG;
                              #if defined(DEBUG_TRACE)
                                Serial.println("appui SUPER long");
                              #endif
          }
          else
          {
              if((millis()-temps_appui_long) > T_APPUI_TRES_LONG)
              {
                 type_appui = TRES_LONG;
                              #if defined(DEBUG_TRACE)
                                Serial.println("appui TRES long");
                              #endif
              }
              else
              {
                  if((millis()-temps_appui_long) > T_APPUI_LONG)
                  {
                     type_appui = LONG;
                              #if defined(DEBUG_TRACE)
                                Serial.println("appui long");
                              #endif
```

```
                }
                else
                {
                    if((millis()-temps_appui_long) > T_APPUI_COURT)
                    {
                        type_appui = COURT;
                            #if defined(DEBUG_TRACE)
                              Serial.println("appui court");
                            #endif
                    }
                }
            }
        }
    }
}

// MENU PRESENTATION DEPENDING ON BUTTON PRESS TIME

if((niveau_precedent_inc == BUTTON_PRESSED) && digitalRead(BUTTON_PIN) == BUTTON_PRESSED )
{   // The button is still pressed: display successively the menu options which will be validated at button
release
         if(millis()-temps_appui_long> T_APPUI_SUPER_LONG)
         {   // SUPER LONG press time passed => Mode toggle
            if( millis()-temps_aff_lcd > T_RAFRAICHISSEMENT_LCD)
            {
                 Display_message("Mode       ",0,1);
                 temps_aff_lcd=millis();
                         #if defined(DEBUG_TRACE)
                           Serial.println("Mode");
                         #endif
            }
         }
         else
         {
              if(millis()-temps_appui_long> T_APPUI_TRES_LONG)
              {   // VERY LONG press time passed => Console
                 if( millis()-temps_aff_lcd > T_RAFRAICHISSEMENT_LCD)
                 {
                      Display_message("Console Print",0,1);
                      temps_aff_lcd=millis();
                          #if defined(DEBUG_TRACE)
                            Serial.println("Console Print");
                          #endif
                 }
              }
```

```
            else
            {   // c'est peut -être un appui long seulement
                if(millis()-temps_appui_long> T_APPUI_LONG)
                {   // LONG press time passed => Recording
                    if(go_record==TRUE)
                    {
                         if( millis()-temps_aff_lcd > T_RAFRAICHISSEMENT_LCD)
                         {
                              Display_message("Stop Enreg ",0,1);
                              temps_aff_lcd=millis();
                                #if defined(DEBUG_TRACE)
                                  Serial.println("Stop Enreg");
                                #endif
                         }
                    }
                    else
                    {
                         if( millis()-temps_aff_lcd > T_RAFRAICHISSEMENT_LCD)
                         {
                              Display_message("Go Enreg...",0,1);
                              temps_aff_lcd=millis();
                                #if defined(DEBUG_TRACE)
                                  Serial.println("Go Enreg...");
                                #endif
                         }
                    }
                }
                else
                {
                    if(millis()-temps_appui_long> T_APPUI_COURT+T_RAFRAICHISSEMENT_LCD)
                    {   // Short press time passed => Quantity
                         if( millis()-temps_aff_lcd > T_RAFRAICHISSEMENT_LCD)
                         {
                              Display_message("Quantite   ",0,1);
                              temps_aff_lcd=millis();
                              #if defined(DEBUG_TRACE)
                                Serial.println("Quantite");
                              #endif
                         }
                    }
                }
            }
        }
    }
}
```

```
void manage_quantity_input(void)
{
  // A square is blinking during the time the changed value is not stored yet in EEPROM.
  if ((valeur_memorisee == FALSE) && (blink_state == ON) && (millis()-temps_blink > PAUSE_BLINK_ON))
  {
      Display_square(OFF);
      temps_blink=millis();
      blink_state = OFF;
        #if defined(DEBUG_VERBOSE)
          Serial.println("Blink OFF");
        #endif
  }
  if ((valeur_memorisee == FALSE) && (blink_state == OFF) && (millis()-temps_blink > PAUSE_BLINK_OFF))
  {
      Display_square(ON);
      temps_blink=millis();
      blink_state = ON;
        #if defined(DEBUG_VERBOSE)
          Serial.println("Blink ON");
        #endif
  }
  // LCD backlight is switched OFF after a delay after last press
  if ((afficheur_eteint == FALSE) && (millis()-temps_aff > PAUSE_LED_DELAY))
  {
      digitalWrite(BACKLIGHT_PIN,BACKLIGHT_OFF);
      afficheur_eteint = TRUE;
        #if defined(DEBUG_VERBOSE)
          Serial.println("Extinction affichage");
        #endif
  }
  // Storage of the value after a given delay following the last change
  if ((valeur_memorisee == FALSE) && (millis()-temps_memo > PAUSE_MEMO_DELAY) )
  {
      if (EEPROM.read(0) != reglage)
      {
        EEPROM.write(0, reglage);  // update of the chosen value in EEPROM if changed
      }
      valeur_memorisee = TRUE;
        #if defined(DEBUG_TRACE)
          Serial.println("Memorisation valeur");
        #endif
      Display_square(OFF);
      afficheur_eteint = FALSE;
  }
```

```
}



void manage_injection(void)
{
    if(millis()-temps_distri < T_INJECTION_UNITAIRE*reglage)
    {   // start of injection
      if(vanne_ouverte==FALSE)
      {
        digitalWrite(VANNE_PIN,VANNE_ON);
        vanne_ouverte=TRUE;
          #if defined(DEBUG_TRACE)
            Serial.println("Ouverture vanne");
          #endif
      }
    }
    else
    {   // injection duration is passed
      digitalWrite(VANNE_PIN,VANNE_OFF);
      injection_eau = FALSE;  // work is done
      vanne_ouverte = FALSE;
      temps_distri = millis();  // New time reference for the next distribution instant.
          #if defined(DEBUG_TRACE)
            Serial.println("fin distri");
            Serial.println("");
          #endif
    }
}

void manage_display(void)
{
      t1 = dht1.readTemperature();
      t2 = dht2.readTemperature();
      h1 = dht1.readHumidity();
      h2 = dht2.readHumidity();

      if (isnan(h1) || isnan(t1) || isnan(h2) || isnan(t2))
      {
          Display_message("Err capteurs", 0, 0);
          probleme_capteurs = TRUE;
            #if defined(DEBUG_TRACE)
              Serial.println(F("Erreur lecture des capteurs DHT!"));
            #endif
```

```
      }
      else
      {
          probleme_capteurs = FALSE;
          Display_sensors(h1, t1, h2, t2);
            #if defined(DEBUG_VERBOSE)
              Serial.println("nbmes" + String(nb_mesures) +" reste " + String(nb_mesures %
NB_MESURES_AVANT_STOCKAGE)
                                + " Go_Rec " + String(go_record) + " PB Capteur" + String(probleme_capteurs));
            #endif
          if( (probleme_capteurs==FALSE) && (go_record == TRUE) && ((nb_mesures % NB_MESURES_AVANT_STOCKAGE) ==
0))
          {     //  Store one over N values
                nb_mesures_stockees++;
                Values_to_eeprom(nb_mesures_stockees, h1, (t1*5.0), h2, (t2*5.0));
                // store in a BYTE (on 255) a value with a comma: 25.3 => 253 / 2 => 126
          }
      }
      nb_mesures++;
    }
}

///////////////////
// INITIALISATION
///////////////////

// the setup routine runs once when you press reset:
void setup() {
  // make the pushbutton's pin an input:
  pinMode(BUTTON_PIN,INPUT_PULLUP);
  // LCD backlight
  pinMode(BACKLIGHT_PIN,OUTPUT);
  digitalWrite(BACKLIGHT_PIN,BACKLIGHT_OFF);
  // water injection
  pinMode(VANNE_PIN,OUTPUT);
  digitalWrite(VANNE_PIN,VANNE_OFF);
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // the two temp & HR sensors
  dht1.begin();
  dht2.begin();

  // if temperature is more than 4 degrees below or above setpoint, OUTPUT will be set to min or max
respectively
  myPID.setBangBang(4);
```

```
  //set PID update interval
  myPID.setTimeStep(PID_UPDATE_DELAY);

  setPoint = TARGET_OUTPUT_TEMP;  // For the tests, set the target to a fixed value. Do a menu else.

  Serial.begin(9600);
  // Read the previous setting value from EEPROM
  reglage = EEPROM.read(0); // the manual setting of water injection is stored in address 0.
        #if defined(DEBUG_TRACE)
          Serial.println("Init faite");
        #endif
  temps_mesure=millis();
  temps_distri=millis();
  decompte=millis()-temps_distri;
  valeur_affichee=FALSE;
  type_appui=AUCUN;
  lcd.clear();
  Display_message("Quantite: ",0,1);
  Display_valeur(reglage);
  Display_countdown();
  eeprom_current_address = EEPROM_START_OFFSET ;  // The available storage pointer starts after the storage of
the internal constants.

}



////////////////////////
// MAIN LOOP FOREVER ///
////////////////////////

void loop() {

  check_button(); // sets the "type_appui" depending on the user action

  // Setup several state values based on the press type
  switch(type_appui)
  {
      case AUCUN:
      break;
      case COURT:   // Backlight ON and display current state OR increase the quantity of water if press again
                      if (afficheur_eteint == TRUE || valeur_affichee==FALSE)
                      {  // immediate action: display switched ON with current value displayed
                         Display_message("Quantite: ",0,1);
                         Display_valeur(reglage);
```

```
                              #if defined(DEBUG_TRACE)
                                Serial.println("Bouton Appuye");
                              #endif
                           afficheur_eteint = FALSE;
                           valeur_affichee = TRUE;
                        }
                        else
                        {
                           reglage+= 1;
                              #if defined(DEBUG_TRACE)
                                Serial.println("Bouton Réappuyé");
                              #endif
                           valeur_memorisee = FALSE;  // The value just changed, now need to store after a delay 
less than LCD swich-on time
                           temps_aff = millis();  // reference of time for the LCD OFF
                           temps_memo = millis(); // reference of time for the setting being stored
                           if (reglage > MAX_QTITE) reglage=0;
                           Display_valeur(reglage);
                           delay(200);
                        }
                        type_appui = AUCUN; // processing finised, reset button state.
      break;
      case LONG:    // start/stop of the recording
                        if(go_record==TRUE)
                        {   // STOPPE
                              go_record = FALSE;
                              // still need to finish the record with the NULL character
                              if(eeprom_current_address == EEPROM.length())
                              { // eeprom_current_address points on the next storage address, if equal to EEPROM 
size, no more space
                                EEPROM.update(eeprom_current_address-1, 0); // Overwrite the last value with the 
NULL character (end of record)
                                    #if defined(DEBUG_VERBOSE)
                                      Serial.println("NULL ecrit en(-1):" + String(eeprom_current_address-1));
                                    #endif
                              }
                              else
                              { // there is still one place left
                                EEPROM.update(eeprom_current_address, 0);
                                eeprom_current_address++;
                                   #if defined(DEBUG_VERBOSE)
                                      Serial.println("NULL ecrit en:" + String(eeprom_current_address));
                                    #endif
                              }
                              Display_message("Fin Enrg OK",0,1);
```

```
                    nb_mesures=0;
                    nb_mesures_stockees=0;
                   // We could restart next record from here. In this case, comment this next
instruction
                    eeprom_current_address=EEPROM_START_OFFSET;
                    delay(1000);
                    type_appui = COURT; // Processing finished, trigger the quantity display by setting
the relevant keypress type
                }
                else
                {   // LET'S START
                    go_record = TRUE;
                    nb_mesures=0;       // start of the measurement counter
                     // We could restart next record from here. In this case, comment this next
instruction
                     nb_mesures_stockees=0;
                    Display_message("Enrg actif",0,1);
                    type_appui = AUCUN; // processing finised, reset button state..
                }
      break;
      case TRES_LONG: // Print the data from EEPROM to the console (USB)
                Eeprom_to_console();
                Display_message("Aff Termine  ",0,1);
                delay(1000);
                type_appui = COURT; // Processing finished, trigger the quantity display by setting the
relevant keypress type
      break;
      case SUPER_LONG: // Switch between AUTO and MANUAL
                if(regulation_mode == AUTO)
                {
                    Display_message("Manuel       ",0,1);
                    delay(1000);  // needed to ensure readability. Rare exception of delay() use.
                    regulation_mode = MANUAL;
                    reglage = EEPROM.read(0);   // reload the manual value from memory
                    type_appui = COURT;
                }
                else
                {
                    Display_message("Automatique  ",0,1);
                    delay(1000);
                    regulation_mode = AUTO;
                    type_appui = COURT;
                }
      break;
      default:
```

```
            #if defined(DEBUG_TRACE)
              Serial.println("switch default");
            #endif
      break;
  }

  manage_quantity_input();  // simple press of the button does increase the quantity of water to inject.
                            // Does also store this value in EEPROM and display it.

  // As soon as the injection period is passed, the distribution is triggered
  // Principle: opening of the valve is a multiple named "reglage" of a fixed value T_INJECTION_UNITAIRE
  // and the time between two injections is set by T_ATTENTE_INJECTION.
  // If reglage = 0, no injection => deactivated.

  if( (millis()-temps_distri) > T_ATTENTE_INJECTION )
  {
          #if defined(DEBUG_TRACE)
            Serial.println("Faut distribuer!");
          #endif
      // Here the water injection is done
      // define the "reglage" value between REGLAGE_INJECTION_MIN and REGLAGE_INJECTION_MAX
      // to correspond to PID 0 and 255
      if(regulation_mode == AUTO)
      {   // take the PID output value as reglage instead of the manual one.
        reglage = (uint8_t)(outputVal*REGLAGE_INJECTION_MAX/255 + REGLAGE_INJECTION_MIN);
      }
      manage_injection();   // Does open the water valve the right duration
      // The reference time is set in this function once the injection is done
  }

  // Display the temperature and humidity values periodically at T_MESURE
  if(millis()-temps_mesure > T_MESURE)
  {
      manage_display();   // Display values
      temps_mesure=millis();
  }

  // Update the countdown value every second
  if( (millis()-decompte) > T_SECONDE)
  {   // more than 1 second passed
      Display_countdown();  // count down between two injections
      decompte=millis();
  }

  //call every loop, updates automatically at certain time interval
```

```
  if(regulation_mode == AUTO)  myPID.run();
}
```