Optimisation de l'épuration de la cire

Institut Gériatrique des Solex
Bernique

Abstract

Obtenir une cire la plus pure possible à partir de pains issus de la fonte de cadres retirés des ruches.

Keywords: ruche, cire, abeille, pureté, PID, Arduino UNO, SSD, relai, DS18B20

Creative Commons 3.0 by Bernique

Table des matières

Optimisation de l'épuration de la cire....1
Abstract....1
I. Remerciements aux contributeurs....3
II. Principe de l'optimisation de la pureté de la cire....3
III. Mise en équations....4
IV. Quelques éléments de dimensionnement....4
1 Hardware d'acquisition....4
2 Dimensionnement du bain-marie....4
2.1 Bain-marie industriel de récupération....4
2.2 Bain-marie maison....4
V. Design du code de traitement de l'Arduino....5
1 Architecture et bases du code....5
2 Initialisation d'un cycle de traitement....6
VI. Conditionnement du signal de T° en amont de l'Arduino....6
1 Choix du capteur de température....6
2 Schéma de câblage (DS18B20)....6
3 Test d'acquisition....7
VII. Pilotage du relais de commande du bain-marie en aval de l'Arduino....7
1 Choix du relais....7
1.1 Relais mécanique....7
1.2 Relais statique (SSR)....7
2 PID....8
3 Test d'acquisition....8
VIII. Enregistrement des courbes Tcire, Temp et M/A résistance....9
IX. De la théorie à la pratique : réglage expérimental du PID....9
1 Kp....9
2 Ki....10
3 Kd....10
X. Mises en œuvre pratiques....11
1 Enregistrement d'une courbe de fonctionnement....11
2 Cire obtenue....12
XI. La machine finie....14
Bibliographie....15
Annexe : le code....16

I. Remerciements aux contributeurs

Ce document, tant dans son contenu technique que sa rédaction, est le fruit d'un travail collaboratif entre passionnés d'électronique et d'apiculture.

II. Principe de l'optimisation de la pureté de la cire

Périodiquement, l'apiculteur procède à la fonte des cadres, opération qui consiste à faire fondre la cire présente sur les cadres anciens et désertés de la ruche.
L'objectif est multiple :

- Débarrasser les cadres des cires souillées (potentiellement source d'infection pour les abeilles) afin de pouvoir les équiper de feuilles de cire gaufrée propres en vue d'une réinsertion dans la ruche,
- Récupérer la cire afin de la valoriser (confection de bougies, vente aux laboratoires de cosmétique, fabrication de cire gaufrée pour cadre, confection d'encaustique…)

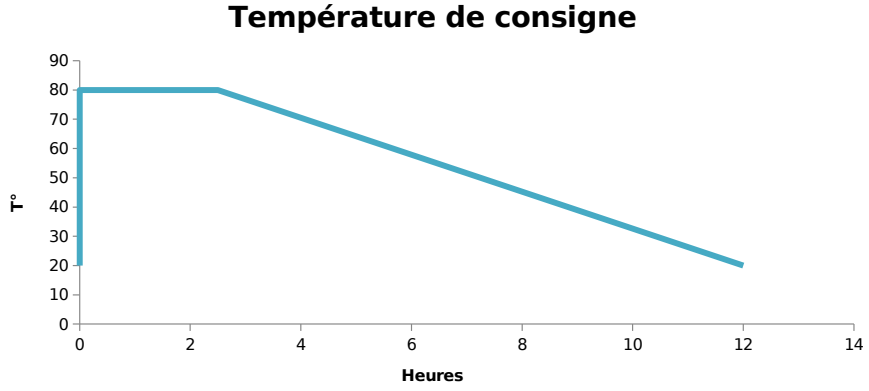
Pour fondre la cire, l'apiculteur utilise des moyens de chauffage comme des chaudières à bain-marie, des cérificateurs solaires (ci-contre)…

A la suite de cette opération, l'apiculteur est en possession de pains de cire dont la pureté n'est pas acceptable en l'état pour espérer une valorisation immédiate. Il doit effectuer une nouvelle fonte de ces pains afin d'en améliorer la pureté.
L'un des paramètres clés de cette seconde opération est le contrôle du temps de refroidissement de la cire liquide depuis sa température de liquéfaction (>64°C) jusqu'à température ambiante. Plus ce temps est long, meilleure sera la décantation des impuretés qui viendront s'accumuler au pied du pain de cire (densité de la cire de l'ordre de 0,96), laissant la partie haute débarrassée de tout corps étrangers. La suite est une séparation mécanique du talon contenant les impuretés du reste du pain de cire.

L'objectif de ce projet est de piloter la température de l'eau d'un bain-marie afin de gérer la fonte puis la décroissance lente de la température de la cire jusqu'à température ambiante.

On retiendra les principes de construction et de dimensionnement suivants :

- Bain-marie électrique en eau, dont la résistance est alimentée depuis le secteur 220V monophasé,
- Trois phases successives de gestion de la température :
  - Phase #1 : monter la température du bain-marie jusqu'à fonte complète de la cire, i.e. 80°C,
  - Phase #2 : maintien de la température de 80°C durant un palier de temps donné t_{maint},
  - Phase #3 : décroissance linéaire de la température jusqu'à température ambiante.



Creative Commons 3.0 by Bernique

- Gestion PID de la consigne de pilotage de la température :
  - Donnée d'entrée : température de la cire $Tcire_t$
  - Donnée de sortie : température de consigne de la cire $Temp_t$
  - Variable d'ajustement : marche/arrêt de la résistance du bain-marie
- Affichage sur écran LCD de la T° de la cire/eau et de la phase en cours ;
- Enregistrement et restitution des courbes de Tcire, Temp et marche/arrêt résistance. Utilisé essentiellement pour le réglage initial du PID (fonction de la géométrie du bain-marie).
- Possibilité de forcer la PHASE par un bouton manuel.

III. Mise en équations

Elle ne concerne que la gestion de la décroissance linéaire de la température en phase #3.

Soit T_{80} la température cible de fonte de la cire (hypothèse de fonctionnement), T_{amb} la température ambiante cible et DT le nombre d'heures choisi pour le refroidissement de la phase #3. A un instant donné t, la température de consigne en température de la cire sera :

$$Temp_t = \frac{T_{amb} - T_{80}}{DT} . t + 80 \qquad \text{avec t en heures}$$

IV. Quelques éléments de dimensionnement

1 Hardware d'acquisition

La fréquence de traitement (f=1/ti) de la boucle PID est relativement lente et tout à fait compatible avec une acquisition/pilotage par Arduino.

Un Arduino Uno se trouve sur le Net pour 15 à 20€ en version genuine, 3 à 5€ en clone chinois !

2 Dimensionnement du bain-marie

2.1 Bain-marie industriel de récupération

Modèle cantine en inox, avec des cavités multiples permettant de produire directement des pains de cire d'une taille compatible avec l'usage final plutôt qu'un monobloc qu'il faudrait fragmenter par la suite.

Privilégier des modèles 220V mono, d'une puissance compatible avec le contrôle commande par relais (voir §VII)

Disponible aussi en neuf sur eBay pour moins de 150€.

2.2 Bain-marie maison

Plusieurs options :

- Plaque de cuisson électrique mono-feu (30€ chez Boulanger !) avec une cocotte-minute de récupération posées dessus (et des récipients à l'intérieur pour le bain-marie) !
- Friteuse (30€ chez Darty pour en citer d'autres !) et des récipients à l'intérieur pour le bain-marie …

Creative Commons 3.0 by Bernique

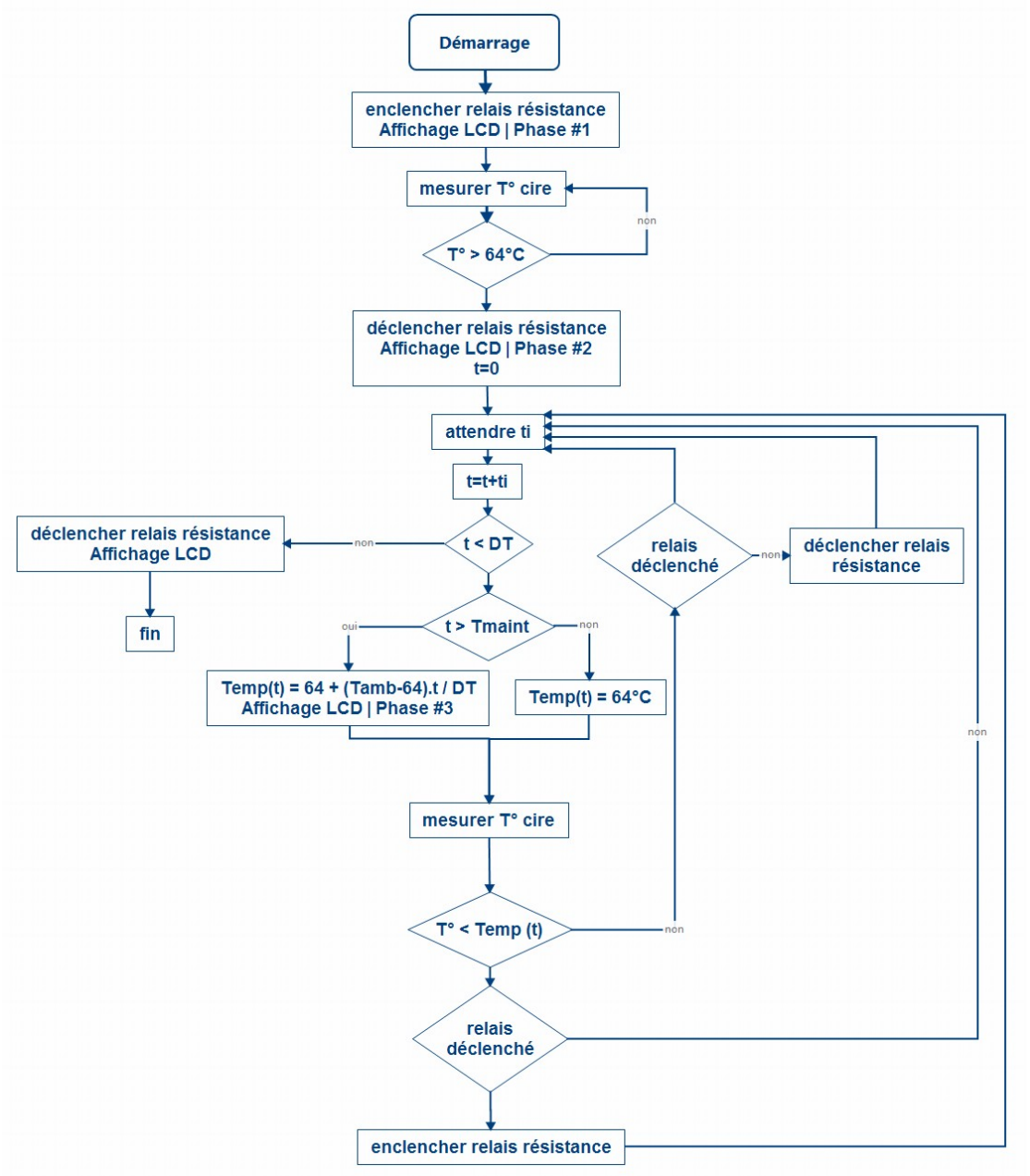
- Bidouille maison sur la base d'une résistance de machine à laver, chauffe-eau... ou encore thermoplongeur plongé dans un récipient rempli d'eau (attention à l'eau et l'électricité ensembles, parole de Cloclo !) !!!.

Calcul de la résistance électrique nécessaire :
Le raisonnement est calé sur la phase #1 de chauffe qui dimensionne le système.
La capacité thermique massique de l'eau est de 4180 J/kg/°.
Énergie nécessaire pour élever 10 litres d'eau de 60°C (=80°-20°): 4180 J/kg/° x 10kg x 60° = 2 508 000 J
Après, sachant que 1Watt=1Joule/seconde, la puissance nécessaire sur une durée de 15 minutes est : 1 839 200 / (15 x 60) = 2787 W

Autre calcul : 900W donnerait un temps de chauffe de 5 à 6min pour un volume de 1,2L (comme dans une petite friteuse).

V. Design du code de traitement de l'Arduino

1 Architecture et bases du code



La base du code repose sur :
Lecture T° : http://www.tweaking4all.com/hardware/arduino/arduino-ds18b20-temperature-sensor/
et
http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertCapteursComplexesDS18B20ThermometreSimple
PID : http://playground.arduino.cc/Code/PIDLibrary
http://playground.arduino.cc/Code/PIDAutotuneLibrary

Pilotage relais : http://www.worldofgz.com/electronique/piloter-un-relais-avec-un-arduino/
http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoInitiationCommandePuissanceRelaisULN2803

2 Initialisation d'un cycle de traitement

A la mise sous tension, machine en attente (résistance déclenchée).
Bouton poussoir de démarrage pour lancement séquence de chauffe (PHASE #1).
Une fois la température T_{80} atteinte, passage automatique en PHASE #2 de maintien.
Une fois la durée de palier atteinte, passage automatique en PHASE #3 de décroissance linéaire de la température. Arrêt automatique en fin de PHASE #3 et mise en stand-by.
Possibilité de forcer le passage en PHASE suivante avec le bouton poussoir.

VI. Conditionnement du signal de T° en amont de l'Arduino

1 Choix du capteur de température

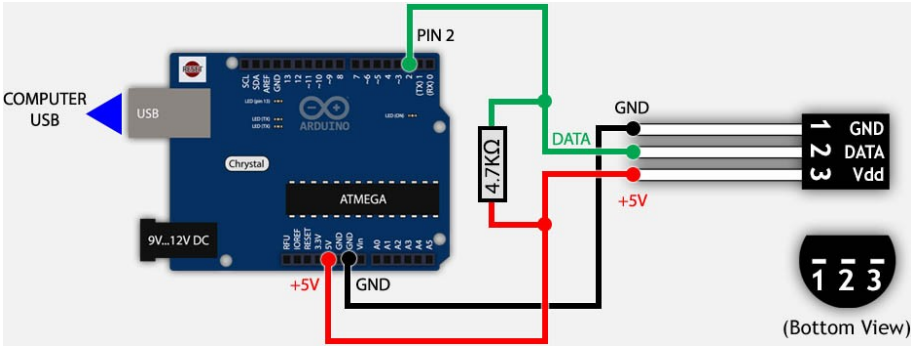
Le capteur choisi est une sonde de température DS18B20 1-wire encapsulée dans un boîtier en acier inoxydable étanche, directement alimenté par l'Arduino.

- Alimentation : 3,0 à 5,5V
- Température de fonctionnement : -55°C à +125°C
- Précision : 0,5°C (dans la gamme -10°C à 85°C)
- Sorties : Vdd (rouge) | DATA (jaune) | GND (noir)

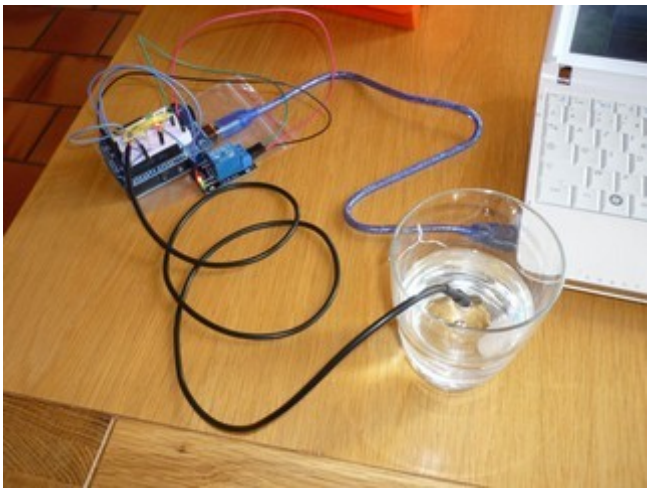
Disponible sur eBay pour quelques €uros. Sa programmation n'est pas aisée, mais il dispose d'une librairie spécifique bien garnie et on peut sur une seule broche d'entrée d'Arduino cumuler et piloter jusqu'à 128 capteurs.

C'est overkill, et une simple sonde NTC10k suffirait probablement… mais c'était dans ma boite ;-)

2 Schéma de câblage (DS18B20)



3 Test d'acquisition



La gestion de la sonde de température est testée via un petit programme spécifique dont le code sera réutilisé dans le code global final de pilotage.
La température est visualisée dans une fenêtre terminale avec un taux de rafraîchissement d'une seconde.

La sonde est plongée dans de l'eau glacée (glaçons en cours de fonte) qui nous sert de référence de température.

Après stabilisation, l'Arduino nous restitue une T° de 0,06°C. Pour un capteur donné à 0,5°C de précision, ce n'est pas trop mal !

VII. Pilotage du relais de commande du bain-marie en aval de l'Arduino

L'Arduino étant alimenté en 5V, il ne peut pas alimenter la résistance chauffante du bain-marie en direct (elle nécessite du 220V).
Il faut passer par un relais de pilotage qui servira d'interrupteur. Ceci étant, la limitation de courant en sortie de l'Arduino ne permet pas toujours de piloter directement un relais en sortie analogique (sous peine de griller la platine). Plusieurs options sont envisageables.

1 Choix du relais

1.1 Relais mécanique

Pour piloter un relais mécanique sans griller l'Arduino, il faut soit utiliser un transistor de commande intermédiaire (un 2N2222 doit convenir), soit opter pour une platine-relais pré câblée pour Arduino (eBay, quelques €uros !).
Une autre solution envisageable est le recours en lieu et place du transistor à un ULN2803 (octuple driver d'amplification de puissance ON/OFF)
On fera attention la cohérence entre la capacité maximum du relais de pilotage, i.e. 10A-250V maxi pour le modèle choisi, et la résistance du bain-marie choisie ou dimensionnée au §IV2
Ce type de relais présente l'inconvénient de devoir être utilisé à l'horizontale, au risque de ne pouvoir commuter (influence gravité sur pièces mécaniques).

1.2 Relais statique (SSR)

Il existe des relais statiques (composés d'un optocoupleur et d'un triac) qui peuvent être commandés en 5V sous des intensités de moins de 20mA (toujours dans l'optique de ne pas griller l'Arduino), avec des pouvoirs de coupure suffisants (de l'ordre de 25A comme le Crouzet GNA5).
Facile à mettre en œuvre (plug&play sur l'entrée Arduino), acceptant de grosses fréquences de commutation, ils sont parfois d'un coût peut-être prohibitif (~36€ pour des marques réputées… mais quelques €uros seulement en version asiatique!).

En revanche, ils ne souffrent pas comme les relais mécaniques lors des transitoires, fonctionnent la tête en bas et ne s'usent pas… c'est l'option qui sera retenue pour la construction de la machine.

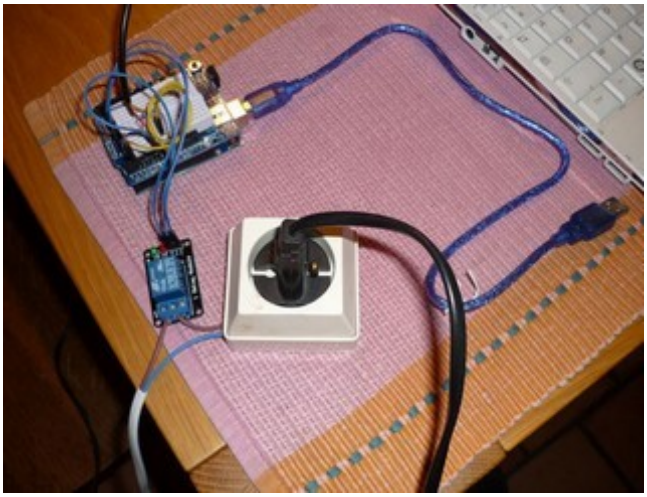
2 PID

La commande en "tout ou rien" d'une résistance de chauffe autour d'une simple consigne risque de nous générer un pilotage assez peu maîtrisé de la température avec des oscillations importantes (la faute aux inerties de l'élément chauffant et de l'eau). En effet, la résistance se déclenchera (ou démarrera) trop tard sur la base d'un seuil cible quelle ne peut pas anticiper. Le recours à une simple hystérésis autour de la consigne n'est pas non plus une solution idéale, car sa valeur dépendant fortement de la plage de fonctionnement (il y plus de pertes calorifiques vers 80°C qu'à 30°C : il faut anticiper différemment !).

L'idée est d'utiliser la librairie PID disponible autour de l'Arduino afin de gérer la consigne en anticipant le point de pilotage, limitant au mieux les oscillations suscitées. Je vous laisse voir sur Wikipédia les détails de la notion de Régulateur PID.

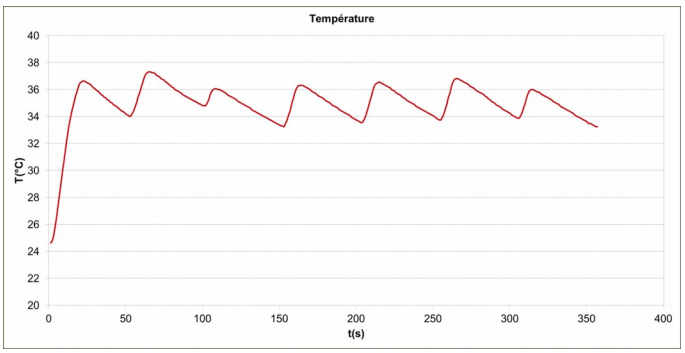
3 Test d'acquisition

Afin de tester l'ensemble du système et la programmation avant d'investir dans un bain-marie (i.e. acquisition de T° et pilotage du relais pour atteindre une consigne), le petit montage suivant a été réalisé pour un coût dérisoire (sur la base d'un relais mécanique, mais il en serait de même avec un SSR).



La prise de récupération est pilotée par le relais qui agit comme un interrupteur sur le câble d'alimentation secteur. Elle alimente un sèche-cheveux (sur position "marche") positionné en face de la sonde de température DS18B20.

Une consigne de 35°C est introduite dans le programme de l'Arduino… et le code fait le reste! Croyez-moi sur parole, lors du premier front montant sur le graphe ci-contre, la chauffe se coupe avant d'arriver à la consigne et tente de s'ajuster par de multiples petits redémarrages du sèche-cheveux… preuve que le PID (non optimisé, ce n'est pas l'objet ici) essaye de faire son boulot (voir §IX pour son réglage) !



VIII. Enregistrement des courbes Tcire, Temp et M/A résistance

Utilise essentiellement dans la phase de réglage du PID (voir §IX). Ne sera pas utilisé en fonctionnement normal, les infos relayées par l'écran LCD étant suffisantes.

L'enregistrement est réalisé grâce le la macro Parallax sous Excel (voir projet *Banc dynamique de puissance*).

IX. De la théorie à la pratique : réglage expérimental du PID

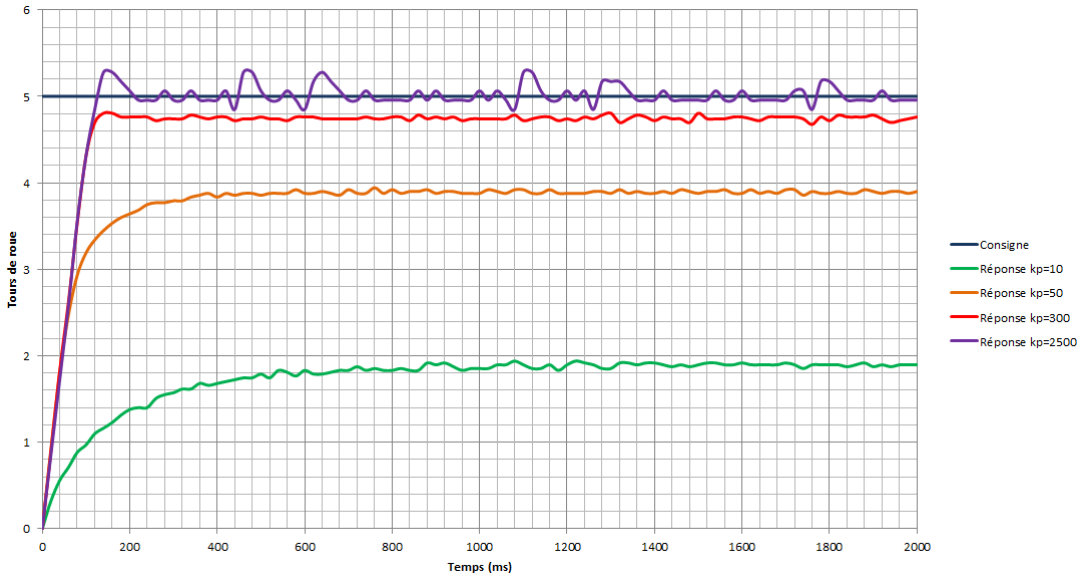
Réglage PID: http://www.ferdinandpiette.com/blog/2011/08/implementer-un-pid-sans-faire-de-calculs/
et http://www.ferdinandpiette.com/blog/2012/04/asservissement-en-vitesse-dun-moteur-avec-arduino/

Note : les courbes ci-dessous sont tirées de ces pages internet et servent à illustrer le propos, l'objet de la consigne/mesure (axe Y) étant totalement déconnecté de notre sujet.

Pour régler nos coefficients du PID, on va utiliser le montage sur les phases #1 et #2 de fonctionnement : démarrage de la chauffe pour atteindre une température fixe, i.e 80°C en stabilisé.

1 Kp

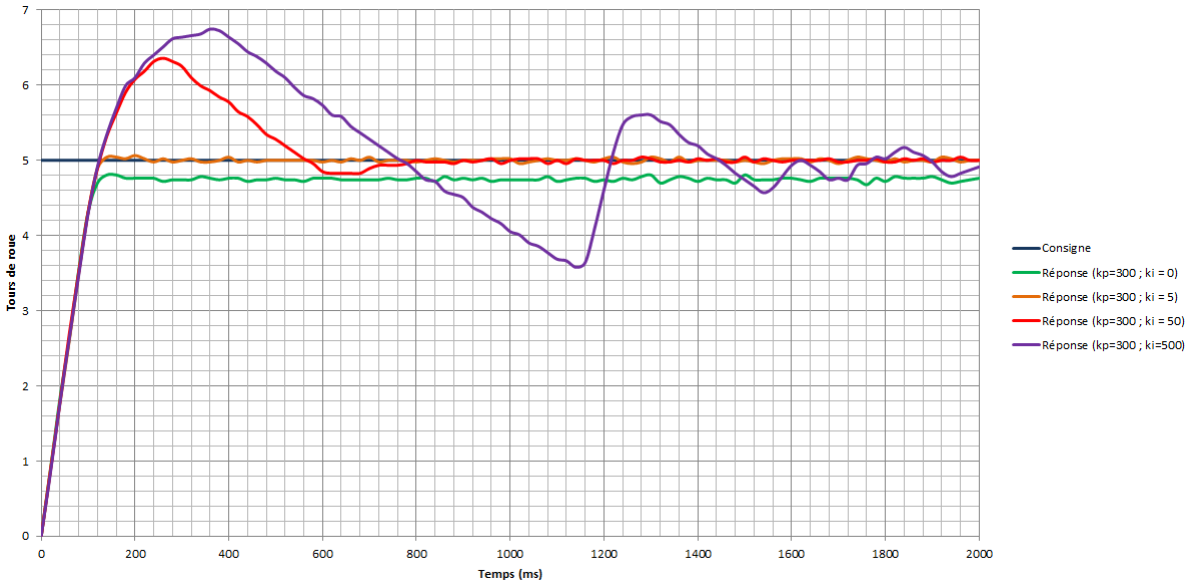
Première étape du réglage expérimental du PID, le coefficient de proportionnalité Kp.
Dans le code de l'Arduino, on pose Ki et Kd égaux à zéro. Puis, par essais successifs, on teste différentes valeurs de Kp jusqu'à trouver la valeur maximum qui nous fait arriver le plus proche possible de la consigne sans toutefois générer d'oscillations.
Sur le graphe ci-dessous, la valeur de Kp=300 est un bon compromis.



Creative Commons 3.0 by Bernique

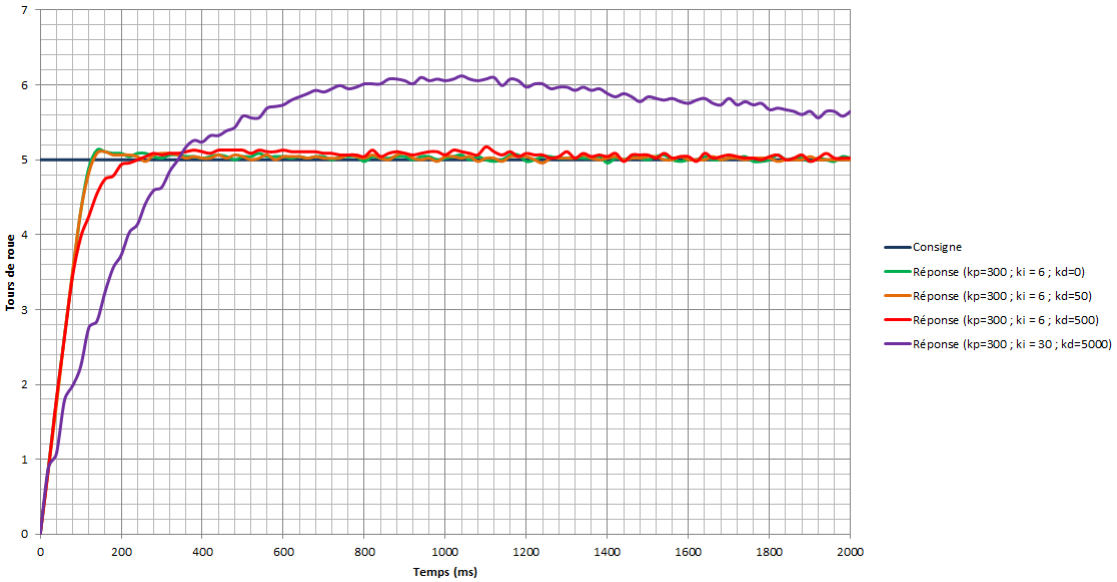
2 Ki

Kp étant déterminé, on voit qu'il demeure une erreur statique (écart entre la consigne et la courbe rouge).
On va jouer avec le coefficient intégrateur Ki pour améliorer la situation, Kp étant figé à sa valeur précédente.
Le principe est le même : on teste plusieurs valeurs de Ki, et on retient celle qui offre le meilleur compromis (i.e. temps de réponse rapide et peu d'oscillations).
Sur le graphe ci-dessous, c'est la courbe orange (Kp=300 | Ki=5). Un essai plus poussé fera choisir Ki=6.



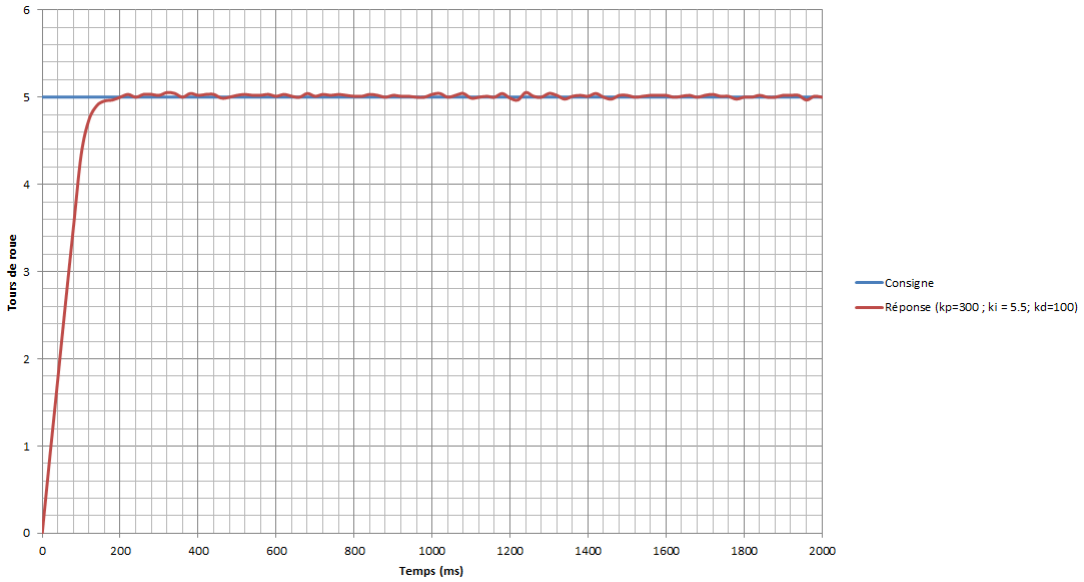
3 Kd

Enfin, pour parfaire le système, on applique la même méthodologie avec le coefficient dérivateur Kd.
Pour se faire, on a figé le couple Kp=300 | Ki=6, et on teste plusieurs valeurs de Kd (bien que la version précédente soit déjà très stable !).



Creative Commons 3.0 by Bernique

Pour finir, on peut affiner avec quelques itérations autour de ce triptyque de valeurs… pour aboutir au tiercé gagnant ci-dessous :



Avouez que c'est tout de même plus satisfaisant que la courbe d'essai obtenue au §VII3 !

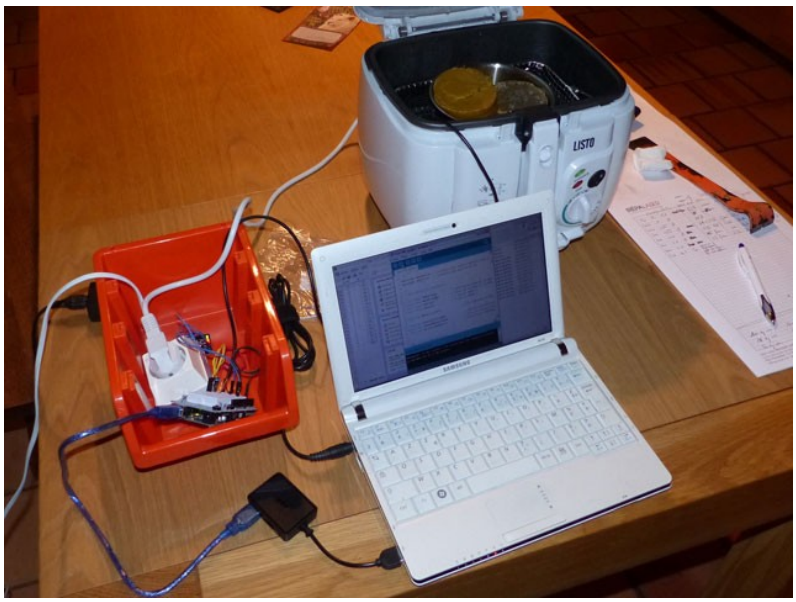
X. Mises en œuvre pratiques

1 Enregistrement d'une courbe de fonctionnement

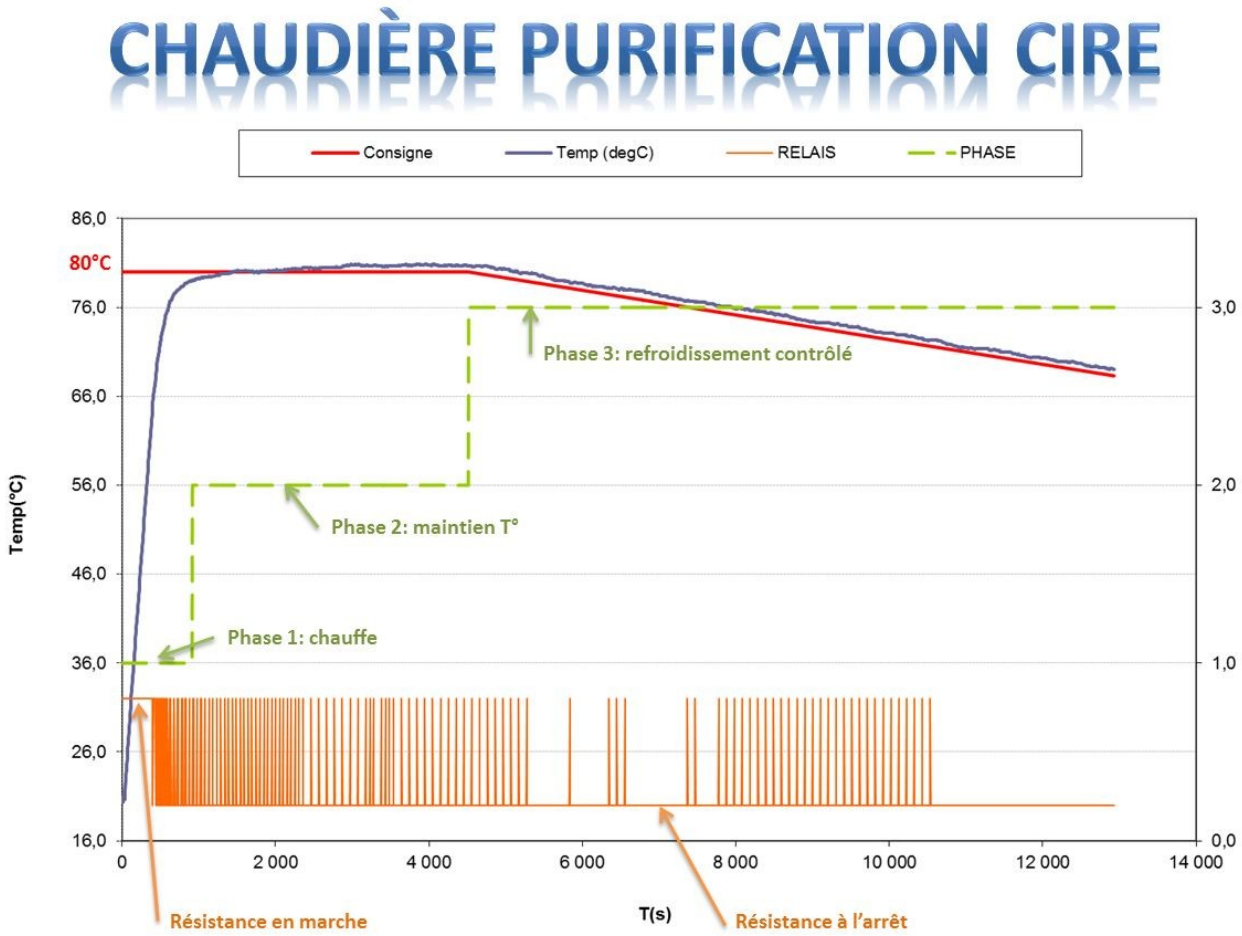
La graphique commenté ci-dessous a été obtenu avec le système et les paramètres suivants :

- une friteuse 1800W (modèle LISTO FCF L2 de chez Boulanger – 29,99€ en Nov.2015) ;
- un Arduino UNO et sa platine relais, relié au PC pour enregistrer les courbes de T° ;
- la charge en cire (qualité sortie de chaudière, voir §10.2) est de 0,48kg ;
- temps de maintien à 80°C de 1h, et le refroidissement de 12h (non représenté en totalité sur le graphique).
- le PID utilisé est : [100 | 0,1 | 30].

La température maxi atteinte est 80,8°C, ce qui est satisfaisant pour notre usage.



Creative Commons 3.0 by Bernique



2 Cire obtenue

Cire en sortie de chaudière, avant épuration (avec les talons d'impuretés et une couleur encore trop sombre dans certains cas, indiquant une pureté insuffisante) :

Cire après épuration avec le système (cire de départ très sombre).
Le dessus est bien plan et non craquelé, ce qui indique un refroidissement suffisamment lent :

Le dessous avec les crasses :

XI. La machine finie

Le proto ayant fait ses preuves et le PID étant réglé pour le bain-marie choisi, l'ensemble des éléments a été intégré dans une grosse boite de dérivation à l'intérieur de laquelle on trouve :

- L'Arduino ;
- Son alimentation 9VDC depuis le secteur 220V (transformateur), l'alimentation par port USB depuis le PC des tests n'étant plus disponible ;
- Un écran LCD 2x16 caractères ;
- Un SSR 25A ;
- Une sonde de température DS18B20 qui sort de la boite ;
- Et une prise 220V permettant d'alimenter le bain-marie et pilotée par le SSR.

Bibliographie

- Site et forum Arduino
- Cours Arduino
- LOCODUINO
- ArduinoInfo
- Tweaking4All
- Sciences et Techniques : from chaos to control
- World of GZ
- Mon Club Elec
- Lejardindeloizo
- Le blog d'Eskimon

Annexe : le code

```
/*****************************************************************************************************************************************************
 *  Bain-marie épuration cire
 *
 *  (C) Bernique - Nov.2015
 *
 *  Under GPL V3 license, http://www.gnu.org/licenses/gpl-3.0.en.html
 *
 *  Paramètres de fonctionnement réglables (voir PDF pour définition des PHASES 1, 2 et 3)
 *   ligne 39: durée de la PHASE 2 en seconde (à adapter si besoin, fonction du volume de cire à fondre)
 *   ligne 40: durée de la PHASE 3 en seconde (à adapter si besoin, fonction de la vitesse de solidification souhaitée)
 *   ligne 41: température de fonte souhaitée (pré-réglé à 80°C); peut être modifié entre 64 et 100°C
 *   ligne 42: température cible de fin de PHASE 3 (pré-réglée à 20°C); peut être modifiée selon usage, par exemple = Tmax pour maintien cire fondue
 *   ligne 66: coefficient Kp proportionnel du PID (à régler en dur selon le matériel de chauffe utilisé)
 *   ligne 67: coefficient Ki intégral du PID (à régler en dur selon le matériel de chauffe utilisé)
 *   ligne 68: coefficient Kd dérivé du PID (à régler en dur selon le matériel de chauffe utilisé)
 *   ligne 124: intervale de temps en secondes entre deux enregistrements dans le tableur de restitution. N'influe pas sur l'échantillonnage du PID!
 *****************************************************************************************************************************************************/

/*
 * Inclusion des librairies utilisées ---
 */
#include <OneWire.h>                    // librairie pour capteur OneWire, dont DS18B20
#include <PID_v1.h>                     // librairie pour gestion du PID
#include <LiquidCrystal.h>              // librairie LCD

/*
 * Déclaration des Constantes
 */
#define RS 12                           // Ecran LCD: Register select
#define E 11                            // Ecran LCD: Enable
#define D4 5                            // Ecran LCD: ligne de données 4
#define D5 4                            // Ecran LCD: ligne de données 5
#define D6 3                            // Ecran LCD: ligne de données 6
#define D7 2                            // Ecran LCD: ligne de données 7
#define COLS 16                         // Ecran LCD: Nbr de colonnes
#define ROWS 2                          // Ecran LCD: Nbr de lignes
LiquidCrystal lcd(RS, E, D4, D5, D6, D7);   // Instanciation de l'objet LCD

const float TEMPO=7200;                 // durée en secondes de la PHASE 2 (7200=2h)
const float MAINT=64800;                // durée en secondes de la PHASE 3 (64800=18h)
const float Tmax=80;                    // T° palier PHASE 2
const float Tmin=20;                    // T° fin de PHASE 3

const int RELAY_1_PIN=7;                // pin connectée au Relais ou SSR
const int MAN=6;                        // pin connectée à l'interrupteur de forçage de la PHASE en cours

const int modeLecture=0xBE;             // Code des instructions du capteur
const int lancerMesure=0x44;            // Code des instructions du capteur
const int broche_OneWire=8;             // declaration constante de broche pour capteur T°

/*
 * Déclaration des Variables
 */
int PHASE=0;                            // utilisé pour savoir dans quelle phase de conduite de la T° on se trouve
float tn, t0, T, Tr;                    // références temporelles pour les PHASES
```

```
float R=0.8;                        // indicateur de fonctionnement de la résistance pour graphique: 0.2=>OFF; 0.8=>ON
int i=1;                            // compteur pour envoi échantillonné des mesures vers tableur de mise en forme
char message1[16] = "";             // stockage pour affichage LCD
char message2[16] = "";             // stockage pour affichage LCD
int H;                              // durée en heures, pour affichage LCD
int M;                              // durée en minutes, pour affichage LCD

int WindowSize = 2000;              // taille de la fenêtre pour fonctionnement du PID
unsigned long windowStartTime;      // référence temporelle pour PID
double Input, Output, Setpoint;     // entrée, sortie et consigne du PID
float Kp = 100;                     // tuning du PID: à régler selon chaque cas d'usage
float Ki = 0.1;                     // tuning du PID: à régler selon chaque cas d'usage
float Kd = 30;                      // tuning du PID: à régler selon chaque cas d'usage
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

byte data[12];                      // Tableau de 12 octets pour lecture des 9 registres de RAM et des 3 registres d'EEPROM du capteur One Wire
byte adresse[8];                    // Tableau de 8 octets pour stockage du code d'adresse 64 bits du composant One Wire
float ma_tempetf=0.0;               // température retournée pa rle capteur en décimal
OneWire  capteur(broche_OneWire);   // crée un objet One Wire sur la broche voulue

/*
 * ********** RUN once to SETUP **********
 */
void setup() {

Serial.begin(115200);               // initialise connexion série à 128000 bauds (=valeur côté PC)

windowStartTime = millis();         // base de temps pour PID
Setpoint = Tmax;                    // Consigne du PID pour PHASE1 et PHASE2
myPID.SetOutputLimits(0, WindowSize); // tell the PID to range between 0 and the full window size
myPID.SetMode(AUTOMATIC);           // turn the PID on

pinMode(RELAY_1_PIN,OUTPUT);        // le relais est une sortie sur RELAY_1_PIN
digitalWrite(RELAY_1_PIN,LOW);      // le relais est au repos (adapter selon câblage du relais)
pinMode(MAN,INPUT);                 // l'interrupteur de forçage de la PHASE est une entrée sur MANUAL
pinMode(MAN,INPUT_PULLUP);          // utilisation de la résistance de PULLUP interne pour éviter les interférences sur une pin flottante!

CapteurInit();                      // appel de la fonction d'initialisation du capteur de T°; prise en compte du premier capteur trouvé

Serial.println("MSG,Starting ...");                                  // Préparation du tableur de mise en forme des mesures
Serial.println("CLEARDATA");                                         // Clear data already in the spreadsheet (starting from row 2)
Serial.println("LABEL,PHASE,T(s),Consigne,Temp (degC),RELAIS");      // Column title, goes to row 1

t0 = millis();                      // base de temps pour graphique

// Gestion de l'écran
lcd.begin(COLS, ROWS);              // Nbr de colonnes et de lignes
lcd.clear();                        // Effacer l'écran LCD
delay (500);
lcd.print("Initialisation");        // on fait patienter l'utilisateur

}

/*
 * Main loop
 *  is used to compute measurements and output values to the serial port.
 *  We print statements which the PLX-DAQ macro is able to use
 *  Refer to :
```

```
 *    https://www.parallax.com/downloads/plx-daq
 *  for more details
 */
void loop(){

// gestion de la T°
ma_tempetf=capteurMesureTemp();         // appel de la fonction de mesure - renvoie une valeur float
tn = millis() - t0;                     // base de temps

// envoi les mesures au tableur toutes les 15 secondes seulement; changer i==15 pour adapter
if (i==1)
{
 Serial.print("DATA,");
 Serial.print(PHASE,DEC); Serial.print(',');
 Serial.print(tn/1e3,DEC); Serial.print(',');
 Serial.print(Setpoint,DEC); Serial.print(',');
 Serial.print(ma_tempetf,DEC); Serial.print(',');
 Serial.println(R,DEC);
 i=1;
}
else
{
 i++;
}

// gestion du forçage manuel de la PHASE
if (digitalRead(MAN)==LOW)              // mise à la masse de D6
{
 PHASE++;                               // on passe à la PHASE suivante
 T = millis();                          // mise à zéro de comptabilités temporelles
 Tr = millis();
 if (PHASE == 5) PHASE=0;               // la boucle est bouclée, on se remets à disposition pour commencer un nouveau cycle
 delay(2000);                           // on patiente, histoire de ne pas doubler l'appui
}

// gestion de la consigne en fonction de la PHASE
if (PHASE == 0)
{
 // il ne se passe rien, on fige la consigne et on attends!
 // pour démarrer un cycle, il faut appuyer sur le bouton de forçage de PHASE
   Setpoint=0;

 // Affichage à l'écran
 lcd.setCursor(0, 0);                           //met le curseur en position (0;0) sur l'écran
 sprintf(message1,"Press to start  ");
 lcd.print(message1);
 lcd.setCursor(0, 1);                           // Passer à la ligne 2
 sprintf(message2,"Tr:%2d",(int)ma_tempetf);
 lcd.print(message2);
 lcd.print((char)223);
 lcd.print("C");
 lcd.print("       ");
}

if (PHASE == 1)
{
 Setpoint=Tmax;
```

```
  // Affichage à l'écran
  lcd.setCursor(0, 0);                    //met le curseur en position (0;0) sur l'écran
  sprintf(message1,"PHASE:%d  Tc:%2d",PHASE,(int)Setpoint);
  lcd.print(message1);
  lcd.print((char)223);                   // affichage du signe "°"
  lcd.print("C");
  lcd.setCursor(0, 1);                    // Passer à la ligne 2
  sprintf(message2,"Chauffe  Tr:%2d",(int)ma_tempetf);
  lcd.print(message2);
  lcd.print((char)223);
  lcd.print("C");

  if(ma_tempetf > (Tmax-1))               // on change de PHASE 1°C avant la consigne pour limiter le temps
  {
  Tr = millis();
  PHASE=2;
  }
}

if (PHASE == 2)
{
  // Affichage à l'écran
  H=int((TEMPO+MAINT-(millis()-Tr)/1e3)/3600);
  M=int((TEMPO+MAINT-(millis()-Tr)/1e3)/60 -H*60);
  lcd.setCursor(0, 0);                    //met le curseur en position (0;0) sur l'écran
  sprintf(message1,"PHASE:%d  Tc:%2d",PHASE,(int)Setpoint);
  lcd.print(message1);
  lcd.print((char)223);
  lcd.print("C");
  lcd.setCursor(0, 1);                    // Passer à la ligne 2
  sprintf(message2,"%2dh%2dmn  Tr:%2d",H,M,(int)ma_tempetf);
  lcd.print(message2);
  lcd.print((char)223);
  lcd.print("C");

  if((millis()-Tr) > TEMPO*1e3)
  {
  T = millis();
  PHASE=3;
  }
}

if (PHASE == 3)
{
  // Affichage à l'écran
  H=int((MAINT-(millis()-T)/1e3)/3600);
  M=int((MAINT-(millis()-T)/1e3)/60 -H*60);
  lcd.setCursor(0, 0);                    //met le curseur en position (0;0) sur l'écran
  sprintf(message1,"PHASE:%d  Tc:%2d",PHASE,(int)Setpoint);
  lcd.print(message1);
  lcd.print((char)223);
  lcd.print("C");
  lcd.setCursor(0, 1);                    // Passer à la ligne 2
  sprintf(message2,"%2dh%2dmn  Tr:%2d",H,M,(int)ma_tempetf);
  lcd.print(message2);
  lcd.print((char)223);
  lcd.print("C");
```

```
  Setpoint = (Tmin-Tmax)*(millis()-T)/(MAINT*1e3)+Tmax;
  if((millis()-T) > MAINT*1e3)
  {
    PHASE=4;
  }
}

if (PHASE == 4)
{
  // il ne se passe rien, on fige la consigne et on attends!
  Setpoint=0;

  // Affichage à l'écran
  lcd.setCursor(0, 0);                    //met le curseur en position (0;0) sur l'écran
  sprintf(message1,"Arret du cycle  ");
  lcd.print(message1);
  lcd.setCursor(0, 1);                    // Passer à la ligne 2
  sprintf(message2,"Tr:%2d",(int)ma_tempetf);
  lcd.print(message2);
  lcd.print((char)223);
  lcd.print("C");
  lcd.print("       ");

}

//  Entrée et lancement du PID
Input = ma_tempetf;
myPID.Compute();

/************************************************
* turn the output pin on/off based on pid output
************************************************/
if(millis() - windowStartTime>WindowSize)
{ //time to shift the Relay Window
windowStartTime += WindowSize;
}
if(Output < millis() - windowStartTime)
{
  digitalWrite(RELAY_1_PIN,HIGH);
  R=0.2;
}
else
{
  digitalWrite(RELAY_1_PIN,LOW);
  R=0.8;
}

}
/************************************************
*        fin de la fonction loop
************************************************/

/*
 * *************** Autres Fonctions du programme *************
 */

//************************** fonction d'initialisation du capteur ****************
```

```
void capteurInit(void) // fonction qui ne reçoit rien et ne renvoie rien
{
// XXXXXXXXXXXXXXXXXXXXX Détection du capteur présent sur la broche XXXXXXXXXXXXXXXXXXXXXX
Serial.println("**** Detection du capteur **** ");

while (capteur.search(adresse)== false) // tant qu'aucun nouveau capteur est détecté
{

 // la fonction search renvoie la valeur FAUX si aucun élément 1-wire est trouvé.

 Serial.println("Aucun capteur 1-wire present sur la broche ! "); // affiche message + saut de ligne
 delay (1000); // pause 1 seconde
}

//la suite est exécutée seulement si un capteur est détecté

 // la fonction search renvoie la valeur VRAI si un élément 1-wire est trouvé.
 // Stocke son code d'adresse 16 bits dans le tableau adresse[8]
 // adresse envoyé à la fonction correspond à l'adresse de début du tableau adresse[8] déclaré ...

 Serial.print ("1 capteur 1-wire present avec code adresse 64 bits : ");


 //--- affichage des 64 bits d'adresse au format hexadécimal
  for(int i = 0; i < 8; i++) { // l'adresse renvoyée par la fonction search est stockée sur 8 octets

   if (adresse[i]<16) Serial.print('0'); // pour affichage des O poids fort au format hexadécimal
   Serial.print(adresse[i], HEX); // affiche 1 à 1 les 8 octets du tableau adresse au format hexadécimal
   Serial.print(" ");
 }

 Serial.println();

 //---- test du type de capteur ----
 // le type du capteur est donné par le 1er octet du code adresse 64 bits
 // Valeur 0x28 pour capteur type DS18B20, 0x10 pour type DS18S20, 0x22 pour type DS1820
 if (adresse[0]==0x28)
 {
  Serial.println ("Type du capteur present : Capteur temperature DS18B20.");
 }
 else
 {
  Serial.println ("Le capteur present n'est pas un capteur de temperature DS18B20.");
 }


 //----- contrôle du code CRC ----
 // le dernier octet de l'adresse 64bits est un code de contrôle CRC
 // à l'aide de la fonction crc8 on peut vérifier si ce code est valide
 if (capteur.crc8( adresse, 7) == adresse[7]) // vérification validité code CRC de l'adresse 64 bits
 // le code CRC de l'adresse 64 bits est le 8ème octet de l'adresse (index 7 du tableau)
 {
  Serial.println ("Verification du code CRC de l'adresse 64 bits de ce capteur : VALIDE !");
 }
 else
 {
  Serial.println ("Verification du code CRC de l'adresse 64 bits de ce capteur : NON VALIDE !");
 }
```

```
  //------- message final détection ----
  Serial.println("----- fin de la recherche du capteur ----");
  Serial.println("");
}
//---------- fin de la fonction d'initialisation du capteur ---------


//------------- fonction de mesure de la température --------------

float capteurMesureTemp(void) { //fonction qui renvoie résultat float et ne reçoit rien

  //-------- variable locale de la fonction ----------
  int tempet=0; // variable pour resultat brute  de la mesure
  float tempetf=0.0; // variable pour resultat à virgule de la mesure

// XXXXXXXXXXXXXXXXXXXXX Lancement d'une mesure et lecture du résultat XXXXXXXXXXXXXXXXXXXXXX

 // Serial.println("**** Acquisition d'une mesure de la temperature **** ");

  // avant chaque nouvelle instruction, il faut :
  //   * initialiser le bus 1-wire
  //   * sélectionner le capteur détecté
  //   * envoyer l'instruction

//-------- lancer une mesure --------
  capteur.reset(); // initialise le bus 1-wire avant la communication avec un capteur donné
  capteur.select(adresse); // sélectionne le capteur ayant l'adresse 64 bits contenue dans le tableau envoyé à la fonction
  capteur.write(lancerMesure,1); // lance la mesure et alimente le capteur par la broche de donnée

//-------- pause d'une seconde -----
delay(1000);    // au moins 750 ms
        // il faudrait mettre une instruction capteur.depower ici, mais le reset va le faire


//---------- passer en mode LECTURE ------------
  capteur.reset(); // initialise le bus 1-wire avant la communication avec un capteur donné
  capteur.select(adresse); // sélectionne le capteur ayant l'adresse 64 bits contenue dans le tableau envoyé à la fonction
  capteur.write(modeLecture,1); // passe en mode lecture de la RAM du capteur

// ---------- lire les 9 octets de la RAM (appelé Scratchpad) ----

for ( int i = 0; i < 9; i++) {          // 9 octets de RAM stockés dans 9 octets
    data[i] = capteur.read();           // lecture de l'octet de rang i stocké dans tableau data
  }


//----- caclul de la température mesurée (enfin!) ---------

data[1]=data[1] & B10000111; // met à 0 les bits de signes inutiles
tempet=data[1]; // bits de poids fort
tempet=tempet<<8;
tempet=tempet+data[0]; // bits de poids faible

// --- en mode 12 bits, la résolution est de 0.0625°C - cf datasheet DS18B20
tempetf=float(tempet)*6.25;
tempetf=tempetf/100.0;
```

```
return (tempetf);

}
// ---------- fin de la fonction de mesure de la température ----------

// --- Fin programme ---
```