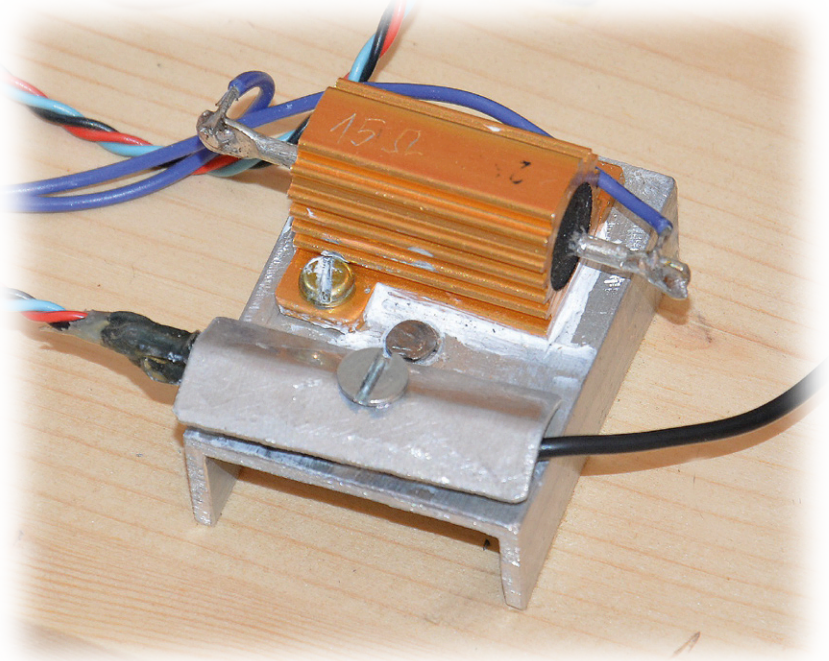
banc thermostaté de test de capteurs

Arduino Uno aux commandes



Pierre Commarmot
(France)

Arduino pilote ici un dispositif destiné à mesurer les principales caractéristiques de la plupart des capteurs de température au silicium, des thermistances CTN et CTP, à détecter les défauts et à trouver des paires correspondantes.

C'est pour trouver ce qui déraillait dans mon chauffe-eau solaire que j'ai entamé ce projet, basé sur l'effet Joule. On chauffe à une température constante, réglable, une petite plaque métallique à laquelle on couple thermiquement le capteur à tester (*DUT*). Après stabilisation, vous pouvez mesurer les caractéristiques du capteur à différentes températures à l'aide de votre Arduino Uno.

Électronique et mécanique

Le socle est constitué d'un morceau de profilé d'aluminium sur laquelle est vissée une résistance de puissance de 15 Ω. Le schéma de la **figure 1** montre le MOSFET de puissance à canal N commandé en MLI (modulation en largeur d'impulsion ou *PWM*) par Arduino pour fournir le courant à la résistance de chauffage. On passe par un convertisseur de niveau entre la sortie TTL d'Arduino et la grille du MOSFET. Même s'il est possible, dans certaines circonstances, d'attaquer directement la grille d'un MOSFET en TTL, ce n'est pas le cas ici. J'ai préféré utiliser deux transistors complémentaires à signaux faibles, bon marché, T1 et T2, pour piloter le MOSFET. D'autres paires de transistors complémentaires feraient l'affaire, le gain et la fréquence de transition ne sont pas critiques ici. Simplification aussi pour le logiciel, puisque le niveau haut TTL (le 1 logique) correspond à la puissance maximale fournie par le MOSFET. J'ai pris une alimentation externe de 12 V continus et 2 A pour le circuit, chauffage compris, parce que je l'avais sous la main !

L'interface entre Arduino et le système de puissance, je l'ai construite sur une plaque perforée. Les deux sondes (LM35 et résistance CTN)

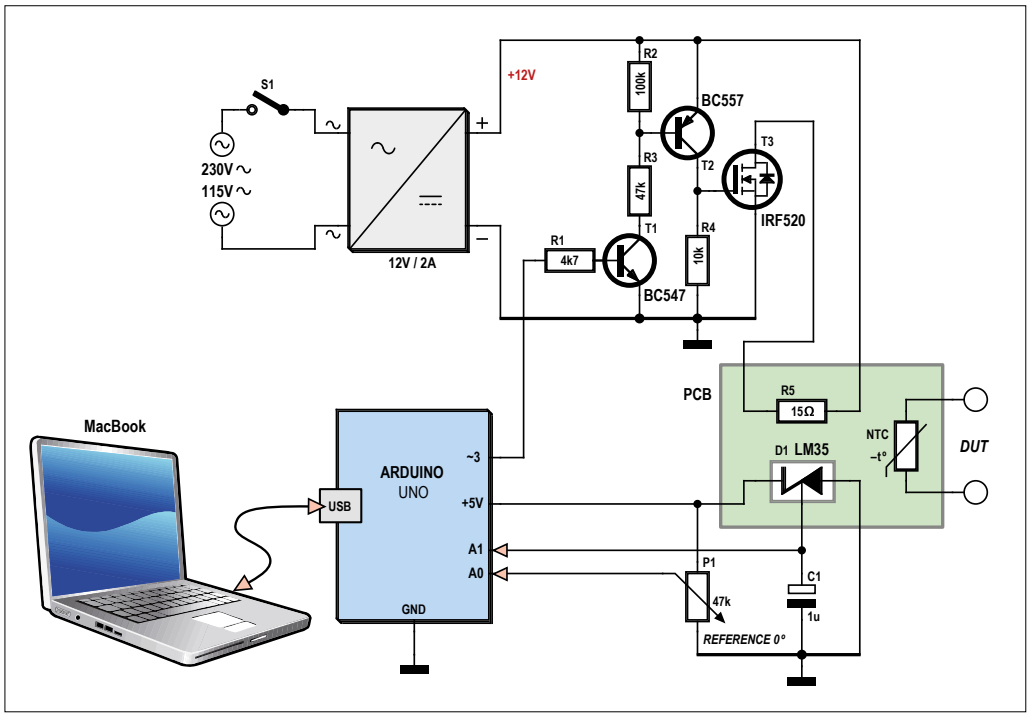


Figure 1.
Schéma du banc de test de capteurs de température.

sont fixées ensemble, avec la résistance de puissance de 15 Ω, sur le morceau d'alu, la « plaque de cuisson ». On réduit ainsi les erreurs dues au gradient de température du banc. Le capteur à tester doit être installé le plus près possible du LM35. Pour réduire le bruit, C1, un condensateur de 1 µF, filtre le signal de sortie du LM35. On règle la température à atteindre sur le banc au moyen d'un potentiomètre connecté directement à l'entrée du CA/N à 10 bits d'Arduino.

Le logiciel

En ce qui concerne le logiciel, nul besoin de musculation, d'ailleurs, nous n'utiliserons que quelques broches de l'Uno. Si vous employez un autre modèle d'Arduino, vérifiez simplement que le câblage est en concordance avec les fonctions d'E/S de votre carte.

Le logiciel est en deux parties, l'une écrite avec l'environnement de développement (*IDE*) Arduino [1], l'autre avec *Processing* [2] pour l'ordinateur hôte qui affiche les données. Notez que pour les deux, je me suis servi des versions en vigueur en septembre 2014. J'utilise le Mac depuis longtemps, ce qui m'a conduit naturellement à *Processing* ; je suppose qu'il faudra l'un ou l'autre ajustement pour le portage vers Linux ou Windows.

Le ***sketch* Arduino** pour ce projet est dans le **listage 1**, il est aussi disponible sur [3]. Voyons ce qu'Arduino y fait. Il commence par prendre sur le potentiomètre la température de consigne pour la mesure. Puis il interroge le capteur LM35 par une autre voie du CA/N qui lui donne la température actuelle du banc. Le LM35 délivre une tension proportionnelle à la température à l'échelle de 10 mV/° C avec une précision de ±0,5 °C. Arduino calcule alors la puissance de chauffe requise pour arriver en température à l'aide d'un logiciel rudimentaire de contrôleur PID (**p**roportionnel, **i**ntégration et **d**érivation). Deux modes se succèdent, l'un « agressif » pour converger rapidement, l'autre « conservatif » à l'approche immédiate de la consigne. Le mode conservatif dispense des longues oscillations autour du point d'équilibre PID. Mon intention était de mesurer automatiquement le capteur à tester, mais finalement, cette fonction n'est pas implémentée dans la version actuelle, il faudra lire les résultats sur un multimètre.
Passons à la suite ! Arduino envoie à peu près toutes les 500 ms les paramètres suivants, séparés par des virgules et terminés par un retour chariot (CR) :

- température de consigne (virgule flottante, gamme de 0 à 100 °C)
- température actuelle du banc par le LM35 (entier, gamme de 0 à 100 °C)

- puissance de chauffe calculée (entier, gamme de 0 à 100 °C)
- température du capteur à tester (virgule flottante, gamme de 0 à 100 °C)

ciale est lancée pour traiter les événements de l'ordinateur. L'excellent site « G4P » est une mine d'information sur *Processing*.

Figure 2.
Mon banc au banc de travail.

Le câble USB entre l'ordinateur hôte et Arduino Uno transporte la tension d'alimentation pour Arduino et le LM35 ainsi que les paramètres transmis. Le petit logiciel *Processing* ne fait que tracer un rectangle autour de quatre curseurs, un pour chaque paramètre.

À la réception d'une chaîne de caractères, les paramètres sont triés et répercutés sur chaque curseur. Après quoi, une tâche spé-

Comment utiliser le banc de test

Branchez la carte Uno sur votre ordinateur (MacBook) et lancez l'IDE Arduino. Définissez le type de carte et le port sériel utilisé, puis ouvrez le fichier TestSondeTemp.ino et transférez-le sur votre Uno.
Câblez l'Uno comme indiqué ici. Il vous faut maintenant créer l'application d'affichage sur l'ordinateur hôte. Lancez *Processing* et ouvrez TestSondeTemp.pde ; vous pouvez

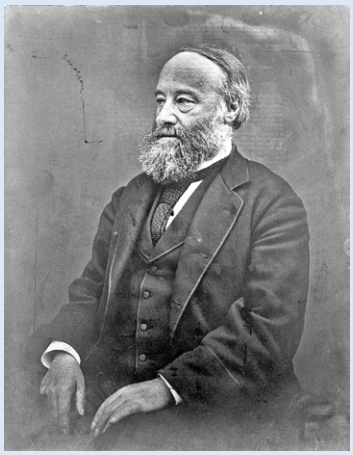
La **loi de Joule**

disait déjà en 1840 que la chaleur Q produite par un courant I dans une résistance R entraînant une chute de potentiel U était égale à :

$Q = P \times t = U \times I \times t = R \times I^2 \times t$ [J/s ou joules/seconde]

ce qui chez les électroniciens se ramène souvent à :

$P = I^2 \times R$ [watts]



créer une application autonome ou choisir de la lancer depuis l'IDE Processing. La police ArialMT-20 doit se trouver dans le fichier Data au même niveau que le fichier source TestSondeTemp.pde.

Après une soigneuse vérification du câblage et des connexions, vous pouvez allumer l'appareil, régler la température, attendre deux minutes la stabilisation thermique et commencer vos mesures avec le multimètre.

Une représentation graphique vous donnera vite un aperçu des caractéristiques de votre capteur. Bon amusement !

(150062 – version française : Robert Grignard)

Liens

[1] IDE Arduino : www.arduino.cc/en/Main/Software

[2] Processing : https://processing.org/download/?processing

[3] Logiciel du projet : www.elektormagazine.com/articles

Listage 1. *Sketch* Arduino pour le banc de test de capteurs de température (Réf. [3])

```
// Test bench PID temperature regulation
// Inputs:  potentiometer temperature reference (0..5V) input A0
//          bench temperature sensor LM35 (0..5V) input A1
// Outputs: heating PWM command out 3
//          USB port: data output every 500 ms

// Check Github website
#include <PID_v1.h>

double Consigne, Input, Output;

// Aggressive and conservative settings
double aggKp=4, aggKi=0.2, aggKd=1;
double consKp=1, consKi=0.05, consKd=0.25;

PID MonPID(&Input, &Output, &Consigne, consKp, consKi, consKd, DIRECT);

// Wiring
int BrocheRefTemp = 0;   // potentiometer on A0
int BrocheSondeRef = 1;  // LM35 on A1
int BrocheSondeTest = 2; // DUT on A2, not used
int BrocheMosFET = 3;    // MOSFET gate

// Variables initialisation
int PotValue;         // Potentiometer value
int ValPot1024 = 0;   // 0-1023 PotValue
int ValPWM256 = 0;    // PWM output value

// Defining inputs-outputs, initialisation
void setup() {
  Serial.begin(19200);
  pinMode(BrocheRefTemp, INPUT);
  pinMode(BrocheSondeRef, INPUT);
```

```
 pinMode(BrocheSondeTest, INPUT);
 pinMode(BrocheMosFET, OUTPUT);
 MonPID.SetMode(AUTOMATIC);
}

void loop() {

 // Getting reference temperature
 PotValue = analogRead(BrocheRefTemp);
 PotValue = map(PotValue, 0, 1023, 0, 100);
 Consigne = double(PotValue); // Target temperature
 Serial.print(float(PotValue));
 Serial.print(“,”);

 // LM35 probe reading
 ValPot1024 = analogRead(BrocheSondeRef);
 ValPot1024 = map(ValPot1024, 0, 205, 0, 100);
 Input = ValPot1024;
 Serial.print(float(ValPot1024));
 Serial.print(“,”);

 // Computing required power
 double gap = abs(Consigne-Input); // Target temp distance
 if(gap<10)
 {  // approaching target temp we use conservative parameters
   MonPID.SetTunings(consKp, consKi, consKd);
 }
 else
 {
    // far from target temp, use aggressive parameters
    MonPID.SetTunings(aggKp, aggKi, aggKd);
 }

 MonPID.Compute();
 ValPWM256 = int(Output);  // output = 1 -->Max power
 if (Consigne<21) {
 ValPWM256 = 0;};
 analogWrite(BrocheMosFET, ValPWM256);
 Serial.print(map(ValPWM256, 0, 255, 0, 100));
 Serial.print(“,”);


 // DUT reading
 ValPot1024 = analogRead(BrocheSondeTest);
 ValPot1024 = map(ValPot1024, 0,1023, 0, 100);
 Serial.print(float(ValPot1024));
 Serial.println(“”);

 delay(500);
}
```