Arduino

Centre de Préparation à l'Agrégation de Montrouge

Baptiste Corrège – baptiste.correge@phys.ens.fr

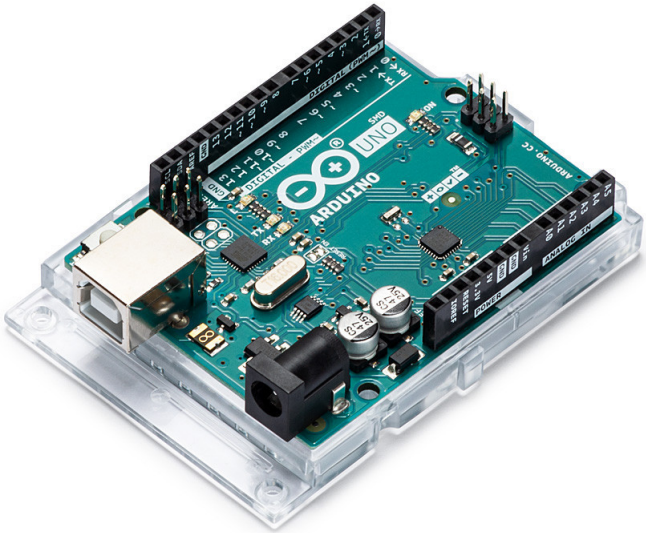


Figure 1 – Une carte Arduino Uno

Table des matières

Remarque
Ce cours est très largement inspiré, du cours de Jérémy Neveu sur l'électronique et du polycopié de TP "Microcontrôleurs" de la préparation à l'agrégation de Physique de l'ENS Montrouge.

I Introduction

1 - Qu'est-ce que l'Arduino ?

L'*Arduino* est une plateforme de microcontrôleur développée dans le but de fournir un matériel pédagogique, bon marché, et programmable avec un langage simple et open source. L'Arduino Uno utilisé en TP est une petite carte électronique composée d'un micro-contrôleur et d'un circuit imprimé. Il dispose de :

— un port USB permettant de le relier à un ordinateur
— 6 entrées analogiques (numérotées A0 à A5), généralement pour utiliser des capteurs
— 14 entrées/sorties numériques (dont 6 en PWM), pour communiquer en binaire avec l'ordinateur
— des ports d'alimentation sous 3.3V ou 5V, et des fiches GND (masse)
— un bouton Reset pour réinitialiser l'Arduino

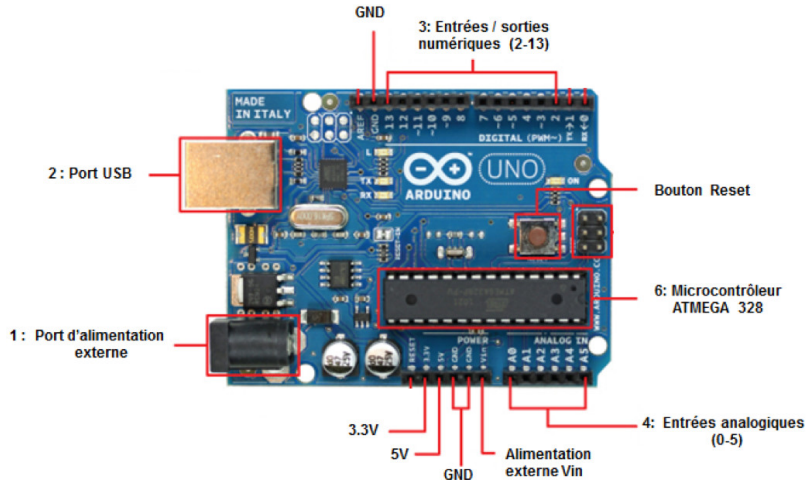


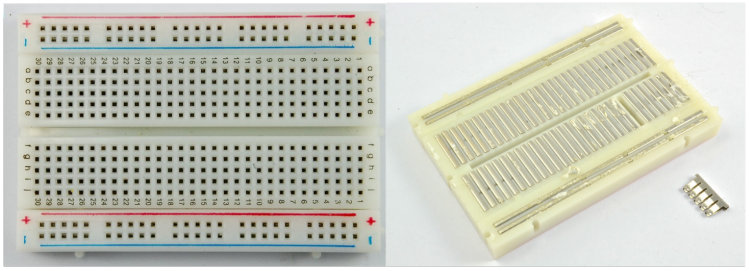
FIGURE 2 – Arduino Uno utilisé à l'agrégation

L'Arduino peut être alimenté sous une tension de 7 à 12V, qui est réduite à 5V par un régulateur de charge, et il fonctionne sous 16Mhz (fréquence d'horloge).

2 - Utiliser son Arduino

2.1 - La 'planche à pain"

La première étape pour utiliser un Arduino est de câbler les différents dipôles et capteurs au microcontrôleur. Pour cela, on utilise une plaque appelée *breadboard*. Sur un breadboard, certaines fiches sont reliées les unes aux autres : toutes les fiches d'une ligne sont électriquement liées. Les fiches sur les colonnes externes de la plaque sont également reliées, on les utilise pour l'alimentation si besoin.



(a) (b)

FIGURE 3 – Photographies d'un breadboard (a) face avant (b) face arrière.

Les composants que l'on peut utiliser avec l'Arduino sont très variés :

— des dipôles (résistance, condensateur, diode, ...) ;

— des tripôles (transistor, ...) ;

— des quadripôles (boutons poussoirs, ...) ;

— des multipôles (Amplificateurs Linéaires Intégrés, ...)

— des éléments utilisés comme des capteurs (photorésistances, thermistances...) ;

L'Arduino ne pouvant lire que des tensions, il faudra souvent convertir la grandeur de mesure en tension. Il existe également un ensemble de capteurs fournis avec l'électronique permettant de lire directement une tension. Leur connectique ("Grove") est différente et demande d'utiliser une interface supplémentaire ("shield").

2.2 - Le Software

Afin de programmer les microcontroleurs type Arduino, on utilise le logiciel IDE Arduino qui permet d'écrire le code dans le langage d'Arduino (dérivé de C), de le téléverser sur le microcontrôleur, et de récupérer les informations récupérées via le port Série. Le logiciel est téléchargeable à l'adresse suivante : https://www.arduino.cc/en/software.

Attention !
Il faut télécharger une version antérieure à la version 2.0.

Commençons par observer l'onglet "Outils". En premier lieu, il faut sélectionner le port où est connecté le microcontrôleur et le type de carte (ici uniquement des Arduino Uno). Deux outils permettent d'afficher le rendu du port série, c'est-à-dire le signal de retour de l'Arduino : le *moniteur série* et le *traceur série*. Lorsqu'on écrit un code, on peut tester sa compilation en cliquant sur l'icône *Vérifier*. Puis, lorsqu'on est prêt à le transférer au microcontroleur, on peut cliquer sur *Téléverser*.

Attention, pour pouvoir afficher les informations envoyées par une carte Arduino il faut que la carte et l'ordinateur se mettent d'accord sur le débit d'information (baud rate). Cela se fait du côté Arduino à l'aide de la commande `Serial.begin()` et du côté de l'ordinateur par un menu déroulant dans la fenêtre du moniteur ou du traceur. Si les débits ne correspondent pas, la communication ne pourra pas se faire !

Pour ajouter une librairie, il faut aller dans Sketch, Librairies et Ajouter une librairie .ZIP en sélectionnant la librairie voulue.

2.3 - Le langage

Le langage Arduino est basé sur le C. L'objectif ici n'est pas de faire un cours exhaustif sur la programmation des Arduino, mais de donner les clés de l'écriture du code. Un code Arduino est décomposé en trois parties : la déclaration des variables globales, l'initialisation et la boucle.

- **La définition des variables globales** : Elles peuvent être de différents types (dont int, float et d'éventuelles variations : long (grand entier), double (grand nombre à virgule flottante), etc.). On peut aussi si besoin y définir des fonctions. Par exemple :

```
// Declaration des variables
unsigned long startTime = 0;     // entier long non signe (positif) pour stocker le
    temps
unsigned int  tension = 0;       // entier non signe pour stocker le signal de tension
```

- **L'initialisation** : Elle est contenue dans la fonction `setup()`. On y place les définitions des fiches (les pin) utilisées sur la plaque, la vitesse de communication avec le port série... tous les éléments qui n'ont besoin d'être fait qu'une fois. Par exemple :

```
// Initialisation :
void setup() {
    Serial.begin(9600);                  // vitesse de communication avec l'ordinateur
    pinMode(8, OUTPUT);                  // le port 8 est defini comme une sortie
    pinMode(A0, INPUT);                  // le port A0 est defini comme une entree
    digitalWrite(8, HIGH);               // on fixe le potentiel de la fiche 8 a +5 V
    Serial.println("Debut des mesures"); // on communique un petit texte a l'
        ordinateur
    startTime = millis();                // on enregistre le temps du debut de l'
        experience
}
```

- **La boucle** : contenue dans la fonction `loop()`, elle contient tout le reste du programme, et qui comme son nom l'indique va se relancer à l'infini tant que l'Arduino est connecté. Cette boucle va contenir toutes les instructions ("lit la valeur du capteur", "change le statut de telle pin", "attends x millisecondes", ...) :

```
// Boucle :
void loop() {
    tension = analogRead(A0);            // on lit la valeur de la tension en A0
    Serial.println(tension);             // on communique cette tension a l'ordinateur
    delay(100);                          // on attend 100 ms
}
```

On utilise très souvent des tests avec `if` et `else`. On peut lancer une boucle un nombre déterminé de fois avec `for`, ou tant qu'une condition est vérifiée avec `while`. La syntaxe à utiliser est celle du langage C :

```
if (i == 5) {            /* test a realiser */
    ... ;
    }
    else {
    ... ;
}

for (i=0; i<10; i++){    /* variable compteur, condition et  expression d'evolution */
    ... ;
}

while (x<=10) {          /* condition a respecter */
    ... ;
}
```

Quelques fonctions très utiles :

- `Serial.println(x)` : permet d'afficher la valeur de `x` dans le moniteur série.
- `digitalWrite(pin1, HIGH)` : permet de changer le statut de `pin1`. Les deux modes possibles sont `HIGH` et `LOW`, qui correspondent respectivement à 0 et 5 V.
- `analogRead(pin2)` : permet de lire la valeur de `pin2`, une fiche a priori analogique.
- `millis()` et `micros()` : permettent d'obtenir le temps en millisecondes (resp. microsecondes) depuis l'allumage de l'arduino.
- `delay(t)` et `delayMicroseconds(t)` : permet d'attendre un temps `t` (en millisecondes / resp. en microsecondes), qui peut être un entier ou un nombre fractionnaire.

3 - Les limites de l'Arduino

Les sources de limitations des microcontrôleurs sont multiples, il faut bien vérifier dans chaque situation laquelle (ou lesquelles) sont dominante(s). On peut en citer quelques unes :

- **Limitations électroniques :** comme tout système électronique, tension de sortie limitée à la tension d'alimentation (5V), courant de sortie limité (40 mA). Attention, l'Arduino n'est pas conçu pour faire de l'électronique de puissance ; veillez à bien respecter les limites constructeur.
- **Limitations numériques :** l'Arduino est une carte numérique et il y a donc des limites de transmission des données liées à la discrétisation (10 bits pour 5V en entrée analogique), la fréquence d'échantillonnage, le temps de communication avec le port sériel USB (baud rate), au temps d'exécution du code, à la mémoire vive (RAM)...

A cela s'ajoutent des limites techniques diverses : limite de portée (l'Arduino doit être câblé à ses différents capteurs et à son alimentation), limite en température et humidité (on ne peut pas l'utiliser dans tous les environnements)...

Notez que les limites ne sont pas des limites "fondamentales", mais simplement des limites matérielles liées aux compromis faits lors de la conception du microcontrôleur étudié, un Arduino étant principalement conçu pour être simple et surtout peu cher. En général, ses performances sont donc médiocres lorsqu'on les compare à celles d'un appareil dédié.

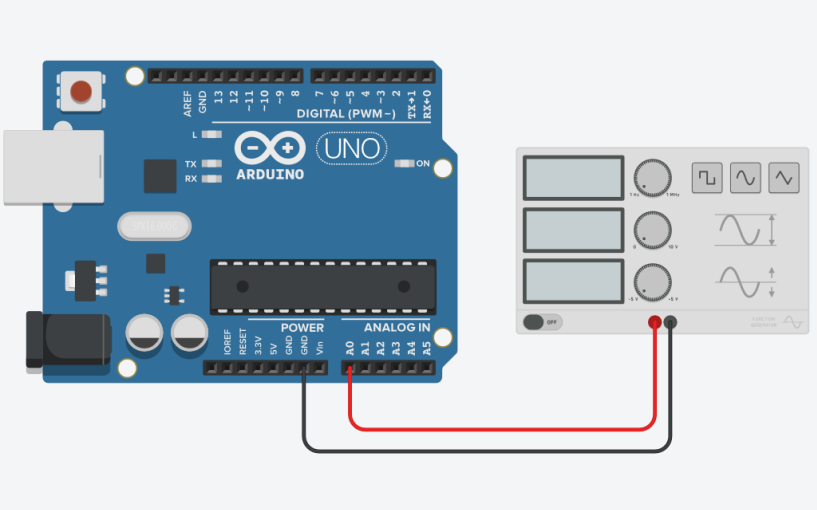
II Premières manipulations

Afin de se familiariser avec l'Arduino, nous allons réaliser quelques petites manipulations pouvant être utiles en TP. On utilisera notamment l'Arduino en tant qu'émetteur et récepteur de signaux électroniques (en tension) par le biais de ses ports I/O.

1 - Réaliser un oscilloscope

Le but est d'acquérir la tension délivrée par un GBF. On commence par créer un signal sinusoïdal de basse fréquence (10 Hz) entre 0 et 5V (ne pas se tromper sur le `Vpp` et l'offset !), et de le relier sur deux colonnes du breadboard. A l'aide des bribes de codes du paragraphe d'introduction, écrire un code pour acquérir le signal avec `analogRead` et l'afficher sur le moniteur série de l'Arduino.

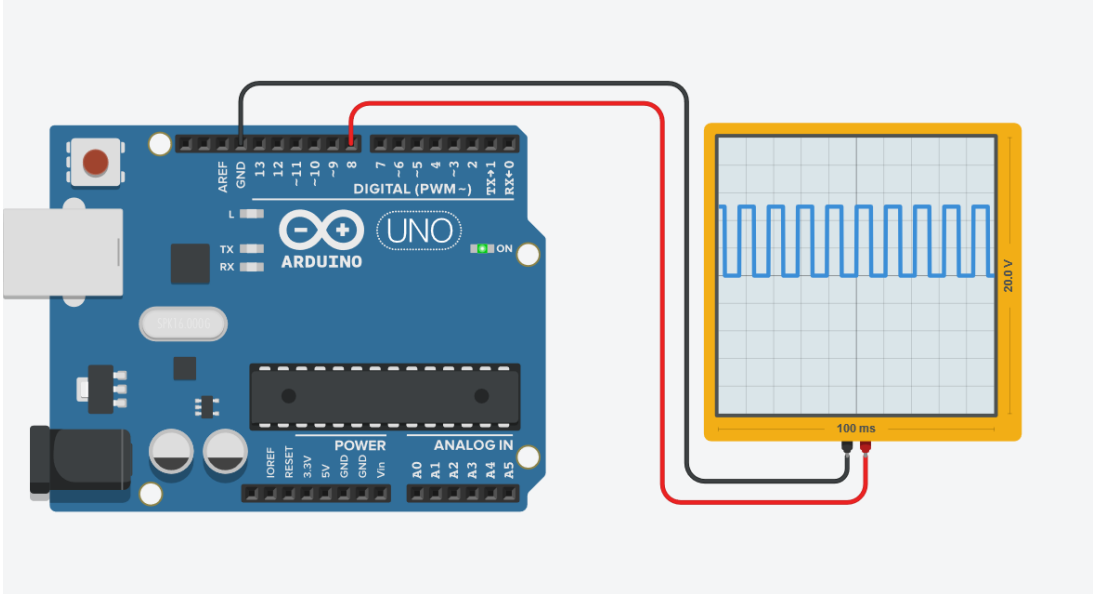
Attention !
Il faut limiter les signaux d'entrée entre 0 et 5V pour ne pas endommager l'Arduino !




```arduino
// Declaration des variables
int const nombreDeValeurs = 10000;              // Nombre de valeurs a afficher a l'
    ecran


// Initialisation :
void setup() {
    Serial.begin(                          );       // Vitesse de communication
    pinMode(                               );       // Le port A0 est defini comme une
        entree
}


// Boucle :
void loop() {
    // On ecrit toutes les valeurs, le plus vite possible.
    for (int i = 0 ; i < nombreDeValeurs ; i += 1) {
        Serial.println(                    );       // Tension mesuree
    }
    delay(10000);                                   // Le programme est termine, on attend.
}
```

2 - Réaliser un GBF

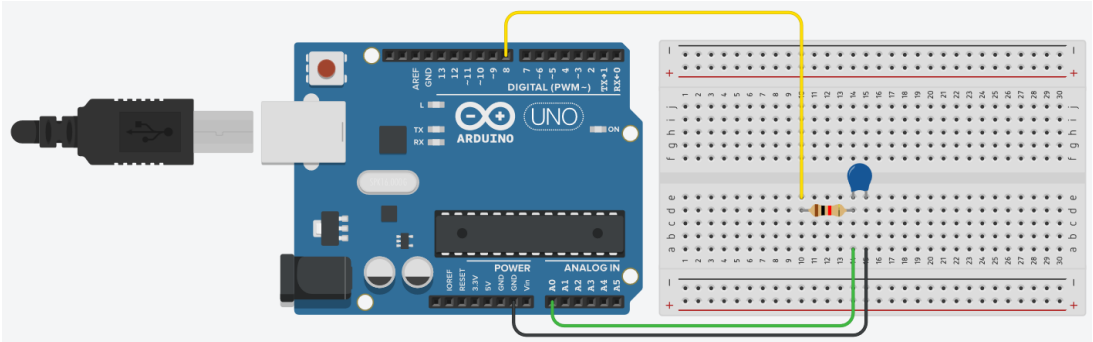
Cette fois-ci, on utilise l'Arduino comme un GBF, et on affichera le signal généré sur un oscilloscope. On peut commencer par un signal créneau en utilisant `analogWrite` mis en mode `LOW` et `HIGH` alternativement et séparé de `delay`. On pourra ajouter une DEL en série avec une résistance ($\sim 2\text{k}\Omega$) pour visualiser le signal de sortie.




```arduino
// Declaration des variables
int const nombreDeCycles =     ;                    // Nombre de cycles du signal
float const T =     ;                               // Periode du signal (en ms)


// Initialisation :
void setup() {
    Serial.begin(9600);                             // Vitesse de communication
    pinMode(                );                      // Le port 8 est defini comme une
        sortie
}


// Boucle :
void loop() {

    for (int i = 0 ; i < nombreDeCycles ; i += 1) {
                                                ;   // Tension a 5V
        delay(T/2);
                                                ;   // Tension a 0V
        delay(T/2);
    }

    delay(1000);                                    // Le programme est termine, on attend.
}
```

3 - Charge d'un circuit RC

L'intérêt des microcontrôleurs type Arduino réside dans le fait de pouvoir être utilisé avec des dipôles simples d'utilisation courante, et établir des connexions directement sur le breadboard. On propose une expérience très simple permettant de mesurer la capacité d'un condensateur à l'aide d'un circuit RC. Sur le breadboard, on branche en série une résistance et un condensateur, en les choisissants de façon à avoir $\tau \sim 0.1$s. On relie ensuite l'entrée du circuit à la PIN 8 de l'Arduino, et la sortie à une broche de masse (Ground). On envoie la tension aux bornes du condensateur à l'entrée "Analog In" A0.




```
float ti, tf ;                          // Deux variables temps pour mesurer tau
float tau ;                             // Constante de temps (decimal)


void setup(){
  Serial.begin(9600) ;                  // Initialisation de la communication serie
  pinMode(8, OUTPUT) ;                  // On definit la borne de sortie
  digitalWrite(8, 0) ;                  // On applique une tension nulle en entree
}

void loop(){
  Serial.println("Decharge du condensateur") ;
  digitalWrite(8, 0) ;

  while ( analogRead(A0) > 1 ){         // On s'assure que le condensateur se decharge
    delay(500) ;
  }

  Serial.println("Mesure en cours") ;
  ti = micros() ;                       // On releve ti
  digitalWrite(8, HIGH) ;               // On applique 5V en entree du circuit RC

  while (analogRead(A0) < 647){         // Tant que le niveau lu est inferieur a 647
  }                                     // (i.e. 63.2% de 1023), on boucle sans rien
      faire

  tf = micros() ;                       // Une fois sorti de la boucle on mesure tf
  tau = (tf - ti) * 1e-6 ;              // Calcul de tau, converti en secondes

  Serial.print("Constante RC en s : ") ;
  Serial.println(tau, 3) ;              // On affiche tau avec 3 chiffres apres la
      virgule

  delay(5000) ;                         // Nouvelle mesure dans 5 secondes
}
```

III La régulation

Dans le domaine industriel, la régulation est l'ensemble des techniques visant à maintenir une grandeur physique constante, malgré l'influence de perturbations extérieures. On peut résumer le fonctionnement de la régulation en trois étapes : mesurer, comparer et corriger. Pour faire de la régulation de manière automatique, on réalise un système en boucle fermée.

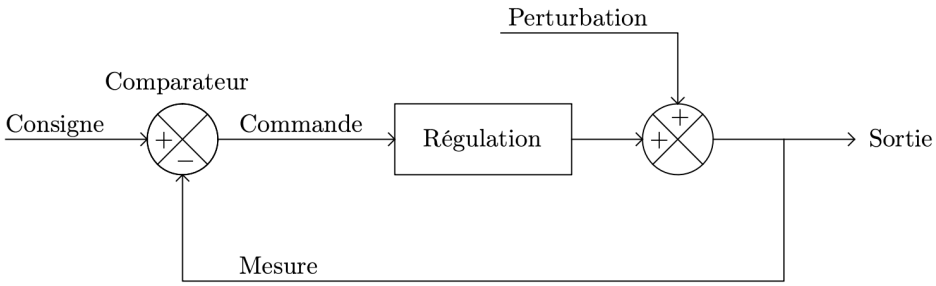


FIGURE 4 – Schéma de principe du fonctionnement de la régulation en boucle fermée.

On utilise trois critères pour caractériser la qualité d'un asservissement :

- **La précision :** elle est quantifiée par l'écart entre la sortie et la mesure, et on distingue deux types de précisions ;

  Définition
  — **Erreur statique :** la différence entre la sortie demandée et la sortie réalisée en régime permanent.
  — **Erreur dynamique :** erreur avec laquelle la sortie suit la consigne imposée au système.

- **La stabilité :** Il existe plusieurs définitions, pas forcément équivalentes ;

  Définition
  Deux définitions possibles :
  — Un système est stable à une entrée bornée correspond une sortie bornée.
  — Un système est stable si sa réponse impulsionnelle tend vers zéro.

- **La rapidité :** Le système régulé doit répondre le plus rapidement possible à un échelon de la consigne. En général, on utilise le temps de réponse à 95%.

Propriété
De manière générale, il y a une compétition entre rapidité et stabilité.

Attention !
Lorsqu'on souhaite que la sortie suive une loi variable, on ne parle pas de *régulation* mais de *suivi de consigne*. Le terme d'*asservissement* désigne les deux cas.

L'étude expérimentale de la régulation sera effectué sur un système de régulation en température, décrit en annexe.

1 - La régulation TOR

Une méthode simple de régulation est la régulation de type "Tout-ou-Rien" : la commande est à 100% de ses capacités, ou bien à 0%. C'est par exemple ce qui est utilisé avec la plupart des convecteurs thermiques.

Le principe de fonctionnement de la régulation TOR est le suivant :

- Si la sortie est inférieure au seuil minimal, on met la commande à 100%
- Si la sortie est supérieure au seuil maximal, on met la commande à 0%

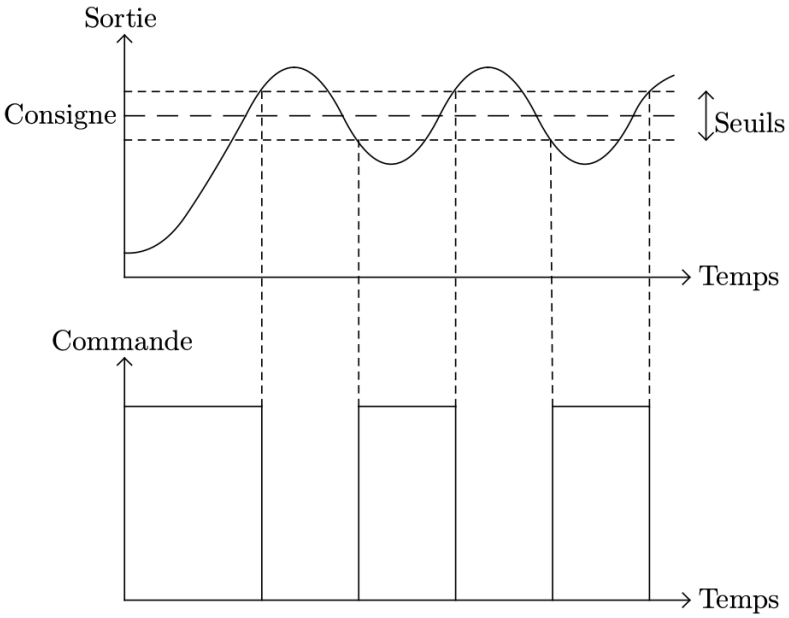


FIGURE 5 – Schéma de principe du fonctionnement de la régulation Tout-ou-Rien.

On observe alors un cycle d'hystérésis dans le réglage de la commande. Elle permet d'éviter une usure prématurée et une réduction notable de la durée de vie de l'instrument due à la grande fréquence de basculement de la commande.

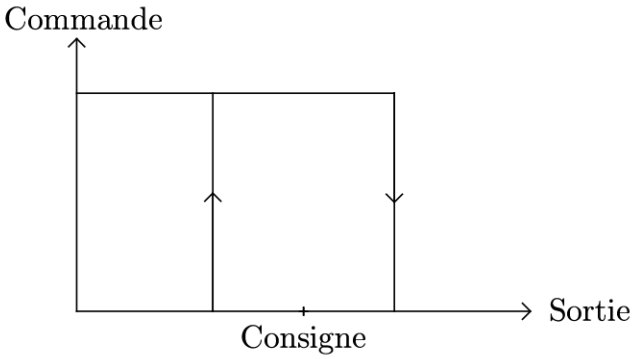


FIGURE 6

La régulation à un seuil est rapide, mais elle n'est pas stable (on observe des oscillations de fréquences élevées) et est peu précise.

```
// Consigne de temperature
float consigne  =     ;               // Consigne en temperature
float seuil     =     ;               // Seuil TOR

// Seuils pour la regulation TOR
float seuilbas  = consigne - seuil ;  // Seuil bas
float seuilhaut = consigne + seuil ;  // Seuil haut

// Variables utiles
float commande = 0 ;                  // Tension de commande
float mesure ;                        // Mesure de temperature
float ecart ;                         // Ecart a la consigne


void setup() {
  Serial.begin(9600) ;                // Initialisation de la communication serie
  pinMode(3, OUTPUT) ;                // On definit la borne de commande
  pinMode(A0, INPUT) ;                // On definit la borne de mesure
  digitalWrite(3, LOW) ;              // On applique une tension nulle en commande
}

void TOR() {
  /* Algorithme de regulation en Tout-ou-rien */
  mesure =              ;             // Mesure de la temperature

    if( mesure > seuilhaut ){         // Si on chauffe trop on allume
    commande =          ;
  }
  if( mesure < seuilbas ){            // Si on refroidit trop on eteint
    commande =          ;
  }

  digitalWrite(3, commande);          // On envoie la commande en sortie

  delay(200);                         // Delais entre chaque mesure
}

void loop() {
  TOR()  ;
}
```

2 - La régulation PID

2.1 - La correction proportionnelle (P)

La correction proportionnelle consiste simplement à ajouter un amplificateur (gain K_p) après le comparateur. La commande est alors proportionnelle à l'écart entre la consigne et la mesure. La commande après correction est alors :

$$C(t) = K_p \times e(t) \qquad \underline{H_p}(p) = K_p \tag{III.1}$$

La correction proportionnelle à plusieurs effets sur le comportement du système. Lorsque le gain K_p augmente :

— l'erreur statique diminue

— la sortie est de plus en plus oscillante, les dépassement de plus en plus grands

— le temps de montée diminue

— la fréquence des oscillations augmente

2.2 - La correction intégrale (I)

La correction intégrale consiste à ajouter un amplificateur intégrateur (gain K_i) après le comparateur. La commande après correction est alors :

$$C(t) = K_i \times \int_0^t e(t)\mathrm{d}t \qquad \underline{H_i}(p) = \frac{K_i}{p} \tag{III.2}$$

Tant qu'une erreur subsiste, elle va être intégrée ce qui va conduire le signal en sortie du correcteur à être de plus en plus important et donc forcer l'actionneur à résorber cette erreur. La commande intégrale est donc progressive mais persévérante.

SI la correction intégrale permet d'annuler l'erreur statique pour une réponse à un échelon, elle introduit déphasage de -90° et risque donc de rendre le système instable. Ce correcteur est donc rarement utilisé seul.

2.3 - La correction dérivative (D)

La correction dérivative consiste à ajouter un amplificateur dérivateur (gain K_d) après le comparateur. La commande après correction est alors :

$$C(t) = K_d \times \frac{\mathrm{d}e}{\mathrm{d}t} \qquad \underline{H_d}(p) = pK_d \tag{III.3}$$

Sans rentrer dans les détails, ce type d'action peut permettre d'éviter des comportements oscillatoires et des dépassements de consignes en introduisant un déphasage de +90°. Il a un ofrt effet stabilisant sans déterorer la précision. Il faut noter que ce type de correcteur est purement théorique et n'est pas réalisable physiquement à cause de la condition de causalité. De plus il amplifie les hautes fréquences donc potentiellement du bruit.

2.4 - La correction PID

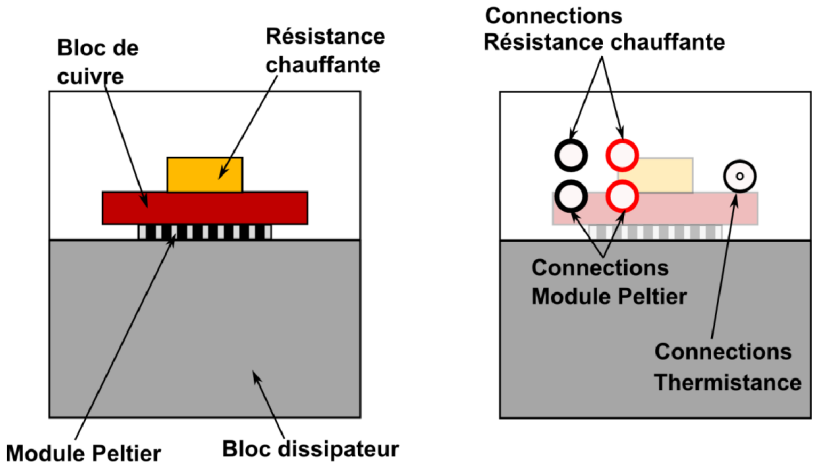
La correction PID pour proportionnelle-intégrale-dérivateur est la somme des correcteurs précédents et dépend donc de trois paramètres à ajuster pour améliorer la vitesse et la précision tout en conservant un système stable.

$$C_{PID}(t) = K_p + K_i \times \int_0^t e(t)\mathrm{d}t + K_d \times \frac{\mathrm{d}e}{\mathrm{d}t} \tag{III.4}$$

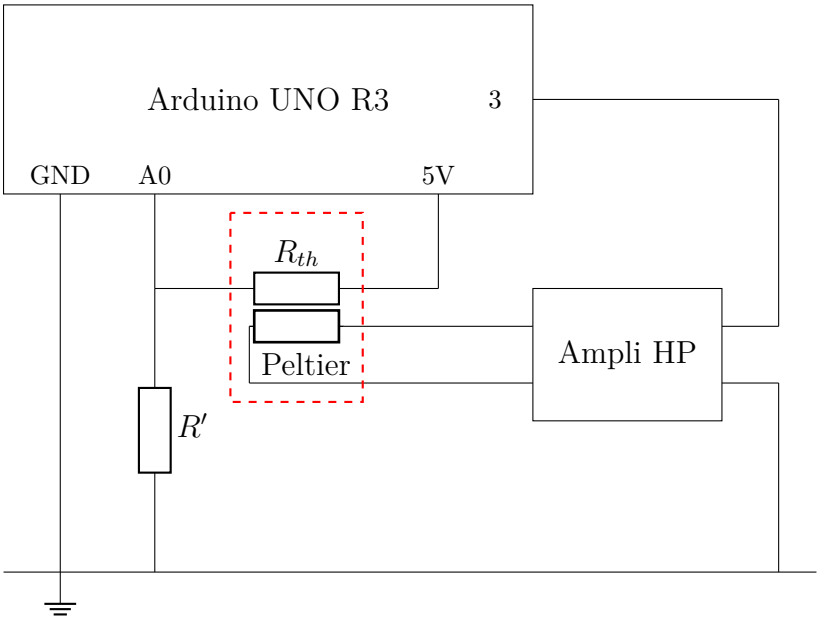
Attention !
Prendre en compte les limites de l'Arduino : max à 255, min à 0

3 - Régulation en température

Une expérience réalisable à Montrouge est la régulation en température. Le principe est de maintenir constante la température d'un bloc de cuivre à l'aide d'un module Peltier. On mesure alors la température à l'aide d'une thermistance, et les perturbations sont créées à partir d'une résistance chauffante. La caractéristique de la thermistance est donnée en annexe.



Après avoir mesuré la température du bloc de cuivre, on modifie donc la tension envoyée dans le module Peltier. Cependant, le module Peltier nécessite d'être alimenté avec une puissance supérieure à celle pouvant être délivrée par l'Arduino : on utilise donc un amplificateur haute puissance.



Enfin, on peut alimenter la résistance chauffante avec une alimentation continue dont on modifie la tension à la main.

```
// Consigne de temperature
float consigne  = 18. ;                 // Consigne en temperature
float seuil     = 0.5 ;                 // Seuil TOR

// Seuils pour la regulation TOR
float seuilbas  = consigne - seuil ;    // Seuil bas
float seuilhaut = consigne + seuil ;    // Seuil haut

// Variables utiles
float commande = 0 ;                    // Tension de commande
float mesure ;                          // Mesure de temperature
float ecart ;                           // Ecart a la consigne


void setup() {
  Serial.begin(9600) ;                  // Initialisation de la communication serie
  pinMode(3, OUTPUT) ;                  // On definit la borne de commande
  pinMode(A0, INPUT) ;                  // On definit la borne de mesure
  pinMode(LED_BUILTIN, OUTPUT) ;        // On definit la LED sur la carte
  digitalWrite(3, LOW) ;                // On applique une tension nulle en commande


Serial.println("Mesure,Commande,Seuil_bas,Seuil_haut");
}


float temperature(float tension) {
  /* Conversion tension-temperature (en Celsius) */
  return 1./(1./298.+1./4718.*log(1024./tension - 1.)) - 273.15 ;
}


void TOR() {
  /* Algorithme de regulation en Tout-ou-rien */
  mesure = temperature(analogRead(A0)) ;        // Mesure de la temperature

    if( mesure > seuilhaut ){                   // Si on chauffe trop on allume
    commande = HIGH ;
  }
  if( mesure < seuilbas ){                      // Si on refroidit trop on eteint
    commande = LOW ;
  }

  digitalWrite(3, commande) ;                   // On envoie la commande en sortie
  digitalWrite(LED_BUILTIN, commande) ;         // On affiche la commande via la LED de
      la carte

  // Affichage :
  Serial.print(mesure);
  Serial.print(",");
  Serial.print(commande);
  Serial.print(",");
  Serial.print(seuilbas);
  Serial.print(",");
  Serial.println(seuilhaut);

  delay(200);                                   // Delais entre chaque mesure
}

void loop() {
  TOR()  ;
}
```

A Utilisation de la thermistance

Dans le montage utilisé, une thermistance qui permet de calculer la température du bloc de cuivre. Il s'agit d'une thermistance à coefficient de température négatif (CNT), dont la valeur varie en fonction de la température selon une loi exponentielle de la forme

$$R(T) = R(T_0) \exp\left[\beta\left(\frac{1}{T} - \frac{1}{T_0}\right)\right] \tag{A.1}$$

où $R(T_0)$ est la résistance à $T_0 = 298\text{K}$ et β est un coefficient considéré comme constant sur la plage de température utilisée. Pour la thermistance utilisée, $R(T_0) = 10\text{k}\Omega$ et le coefficient β a été calibré expérimentalement à la valeur de $\beta = (4718 \pm 40)\text{K}$. La caractéristique résistance-température est alors la suivante :

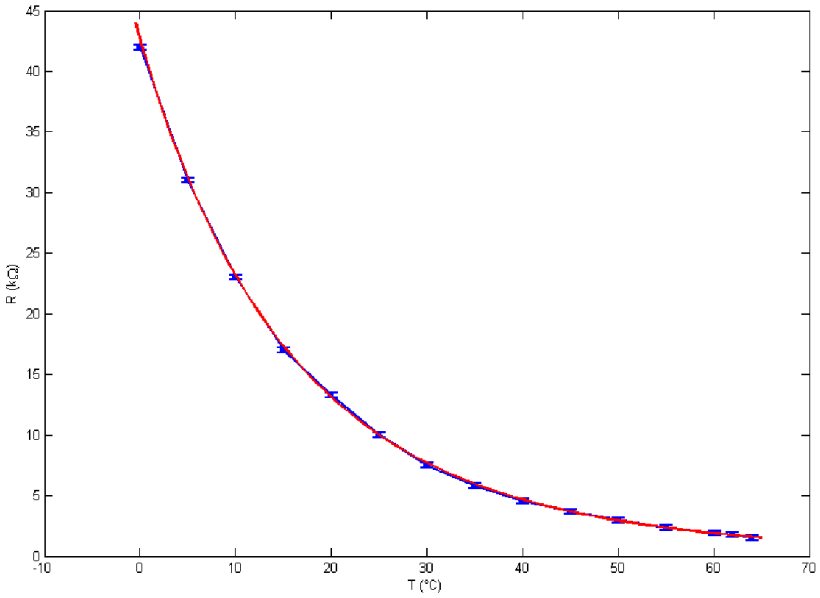


FIGURE 7 – Calibration de la thermistance.

Pour mesurer la résistance de la thermistance avec l'Arduino, la méthode la plus simple est d'utiliser un pont diviseur de tension. On branche alors la thermistance R_{th} et une résistance R' en série entre les ports `5V` et `GND` de l'arduino, et on mesure la tension U aux bornes de la résistance R'. On a alors :

$$U = \frac{R'}{R' + R_{\text{th}}} \times 5\text{V}.$$