Université du Littoral – Côte d'Opale – Calais
Unité d'ouverture, licence MSPI et SV, 2me et 3me année
Licence MSPI non-spécialisée, 1re année S1
Département de physique

Enseignement autour de micro-contrôleur *Arduino Uno*

Robin Bocquet, montages éléctroniques, rédaction de l'énonce 2017-2020
Arnaud Cuisset, carnet des charges et gestion du projet 2017-2019
Pierre Kulinski, réalisateur montages Arduino, soutien technique, Dk 2017
Pascal Masselin, montages éléctroniques, intervention en TP L1 S1, Calais 2021
Wilfried Montagnier, soutien technique, Calais
Dmitrií Sadovskií [*], développement et traitement informatique, rédaction de l'énonce

Calais, automne 2021

1 Introduction

Le but de ces travaux pratiques est de vous initier à l'utilisation des microcontroleurs qui ont littéralement envahis le monde technologique d'aujourd'hui. Un microcontrôleur n'est rien d'autre qu'un microprocesseur à jeu d'instructions limité spécialisé dans les communications avec l'extérieur. On retrouve ce type de composants dans les voitures par exemple où ils gèrent l'ensemble des capteurs et éléments de sécurité du véhicule, dans les drones où ils gèrent les capteurs de vitesse, d'altitude, de lacet, roulis et tangage, dans les robots où bien souvent des radars anti-collisions sont mis en place ou bien encore dans les imprimantes 3D où les moteurs ainsi que les positionnements sont gérés par ce type de composant. Les microcontrôleurs ont été développés dans les années 80 mais ont réellement diffusés dans le grand public depuis 2005 grâce au travail d'un groupe d'italiens dans le monde du logiciel libre qui a développé une plateforme logiciel simplifiant l'utilisation de ces composants. Il s'agit des développements connus sous la bannière ARDUINO et repris maintenant sous le nom GENUINO, adhérant à la charte du développement "libre". L'ensemble des documentations sur la carte ARDUINO, ainsi qu'une mine d'exemples sont consultables sur son site. Enfin, dans cette introduction, notez que vous trouvez nombre de tutoriaux sur le « net » pour vous aider à découvrir le matériel et le logiciel.

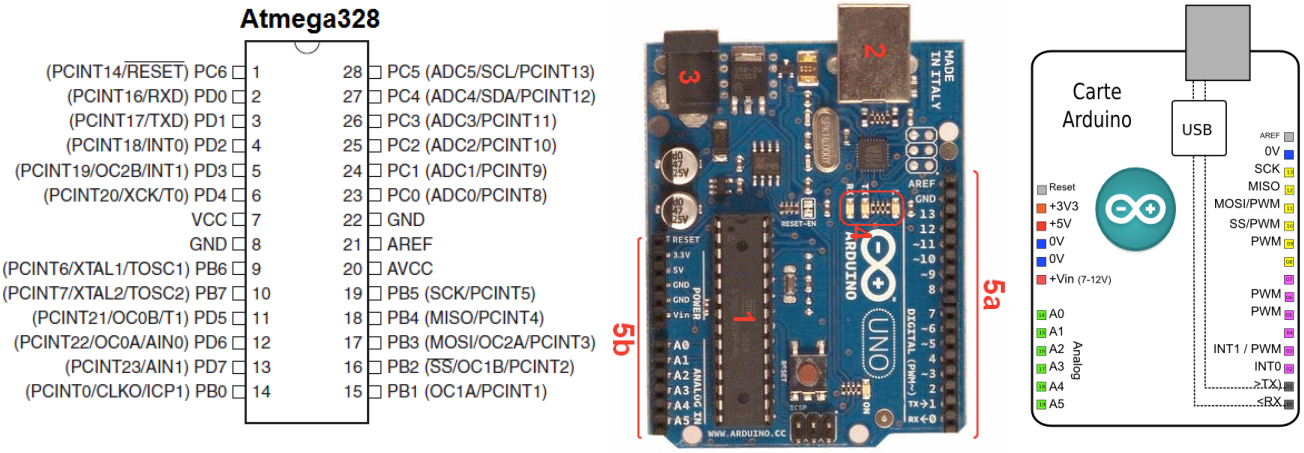


FIGURE 1 – Brochage de microcontrôleur ATMEGA328P-PU et carte ARDUINO UNO où les numérotations correspondent à https ://ctan.mines-albi.fr/macros/latex/contrib/paralist/paralist.pdf

[*] sadovski@univ-littoral.fr, responsable du projet BQE HTLC (Hight Tech Low Cost) ULCO 2017

High performance, low power AVR® 8-bit microcontroller

- Advanced RISC architecture
  - 131 powerful instructions, most single clock cycle execution
  - 32×8 general purpose working registers
  - fully static operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High endurance non-volatile memory segments
  - 4/8/16/32K Bytes of in-system self-programmable flash program memory
  - 256/512/512/1K bytes EEPROM
  - 512/1K/1K/2K bytes internal SRAM
  - Write/Erase cycles : 10,000 Flash/100,000 EEPROM
  - Data retention : 20 years at 85°C/100 years at 25°C (1)
  - Optional Boot Code Section with Independent Lock Bits
  - In-System Programming by On-chip Boot Program
  - True Read-While-Write Operation
  - Programming Lock for Software Security
- Operating Voltage : 1.8 - 5.5V
- Temperature Range : −40°C to 85°C
- Speed grade : 0 - 4 MHz@1.8 - 5.5V, 0 - 10 MHz@2.7 - 5.5.V, 0 - 20 MHz @ 4.5 - 5.5V
- Power consumption at 1 MHz, 1.8V, 25°C
  - Active Mode : 0.2 mA
  - Power-down Mode : 0.1 μA
  - Power-save Mode : 0.75 μA (Including 32 kHz RTC)
- Peripheral Features
  - Two 8-bit timer/counters with separate prescaler and compare mode
  - One 16-bit timer/counter with separate prescaler, compare mode, and capture mode
  - Real Time Counter with separate oscillator
  - Six PWM channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package temperature measurement
  - 6-channel 10-bit ADC in PDIP package temperature measurement
  - Programmable serial USART
  - Master/Slave SPI serial interface
  - Byte-oriented 2-wire serial interface (PHILIPS I2C compatible)
  - Programmable watchdog timer with separate on-chip oscillator
  - On-chip analog comparator
  - Interrupt and wake-up on pin change
- Special microcontroller features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes : Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF

TABLE 1 – Caractéristiques générales du microcontôleur ATMEGA328P

2 Initiation à ARDUINO

Phase 1 (enseignement en classe) Nous utiliserons le microcontôleur ATMEGA328 dont le brochage est montré dans la figure 1. Ses caractéristiques générales se trouve sur le site ATMEL. Rassurez vous cependant, que vous n'aurez à utiliser le microcontrôleur seul que dans les phases finales d'intégration de vos développements, lors des projets libres, si vous en avez le temps. Vous utiliserez la plupart du temps le microcontrôleur dans son environnement de développement ARDUINO. La documentation complète et les explications des différentes broches et entrées sorties sont données sur le site du son fabricant ATMEL.

D'ores et déjà vous pouvez noter (table 1) l'existence d'un port de communication série (RXD et TXD, broches 2 et 3), de convertisseurs analogiques-digitaux (broches ADCi), d'un bus I2C (SDA et SDC) très utilisé avec les capteurs, d'une interface série synchrone dénommée SPI (MISO, MOSI et SCK) et de broches dénommées digitales à collecteur ouvert. Enfin vous disposez d'une mémoire flash d'une capacité de 32 kO pour stocker votre programme de manière permanente mais, bien évidemment, effaçable. Notez que les broches digitales sur la carte ARDUINO, (fig. 1, centre, 5b) sont des niveaux TTL 5V, ce qui signifie, que le niveau logique 1 (HIGH) correspond à une tension comprise entre 2,4 et 5V, et que le niveau logique bas (LOW) correspond à une tension comprise entre 0 et 1,4 V. Chaque sortie numérique est limitée en courant à 20 mA.

2.1 Programmation de ARDUINO

Talk is cheap. Show me the code.
Linus Torvalds

Le microcontrôleur Arduino est programmé en C++ dans un environnement spécialement dédié (sec. 2.1.1). Nous allons couvrir les bases de cette programmation et de l'utilisation de cet environnement en développant un petit code qui nous permettra de façon évolutive :

1. brancher et reconnaître votre carte, faire clignoter la LED propre à Arduino (Uno)

2. modifier le code pour faire clignoter le signal SOS avec cette LED
3. allumer les 4 LED's externes une par une en séquence dans le `setup`
4. passer des messages via port série/usb (aka «Terminal» ou «Console»)
5. communiquer avec l'Arduino via son Terminal par les touches à 1 caractère
6. interpréter les touches, retourner leur code ASCII
7. pour les touches 0..9, a..f, ou A..F les interpréter comme des chiffres hexadécimaux
8. donner la valeur 0..16 du chiffre correspondant
9. donner sa représentation binaire (en 4 bits), utiliser opérations «bitshift» et «bitmask»
10. allumer les 4 LED's externes (voir point 3) selon cette représentation binaire
11. (option) interpréter l'entrée analogique d'un potentiomètre pour allumer les 4 LED's selon la tension obtenue (aka «barographe»)

2.1.1 Installer et utiliser le logiciel Arduino (IDE)

Un microcontrôleur Arduino est en fait très sommaire avec beaucoup moins de possibilités qu'un ordinateur. Pour s'en convaincre, il suffit de regarder sa quantité de mémoire. Aussi, sa programmation se fait séparément sur un ordinateur (*offline*) et le programme est ensuite transféré dans le microcontrôleur pour son exécution.

Pour travailler *offline* avec Arduino, on utilise un environnement de développement intégré, le «desktop IDE» (basé sur l'interface Java). Pour son installation sous Linux, voir `https://www.arduino.cc/en/Guide/Linux`.

```
> tar xvf arduino-1.8.3-linux32.tar.xz
> ln -s arduino-1.8.3/ arduino
> cd arduino
```

Taper la commande `arduino` ouvre la fenêtre de l'IDE (sous Java).

2.1.2 Branchement de la carte ARDUINO à l'ordinateur

Vous disposez de

— un ordinateur sur lequel le programme de développement ARDUINO est installé

— une carte ARDUINO UNO ou équivalent avec son câble USB idoine

— une plaquette d'essais électronique à trous pour les montages électroniques

— des fils dénudés à chaque bout ainsi que les composants électroniques pour les montages envisagés

Lorsque l'on insère le câble USB (type B coté carte, type A coté ordi), nous observons que les liaisons série et USB sont reconnus par l'ordinateur. La commande

```
> ls -lhtr /dev/
crw-rw---- 1 root dialout 188,   0 Aug  3 10:51 ttyUSB0
crwx-w---- 1 dima tty       4,   1 Aug  3 12:30 tty1
crw------- 1 root root    189, 513 Aug  3 12:48 usbdev5.2
crw-rw---- 1 root dialout 166,   0 Aug  3 12:48 ttyACM0
```

renvoie les interfaces usb `ttyUSB0` et série `ttyACM0`.

Pour communiquer avec l'ARDIUNO, à partir de l'IDE, nous sélectionnons le port série `ttyACM0` via TOOLS ▷ SERIAL PORT. Ensuite avec TOOLS ▷ GET BOARD INFO nous pouvons voir que l'ARDUINOest bien reconnu :

```
BN: Arduino/Genuino Uno
VID: 2A03
PID: 0043
SN: 95536333830351807052
```

Enfin, dans les OUTILS ▷ TYPES DE CARTES on choisit/confirme la carte GENUINO UNO.

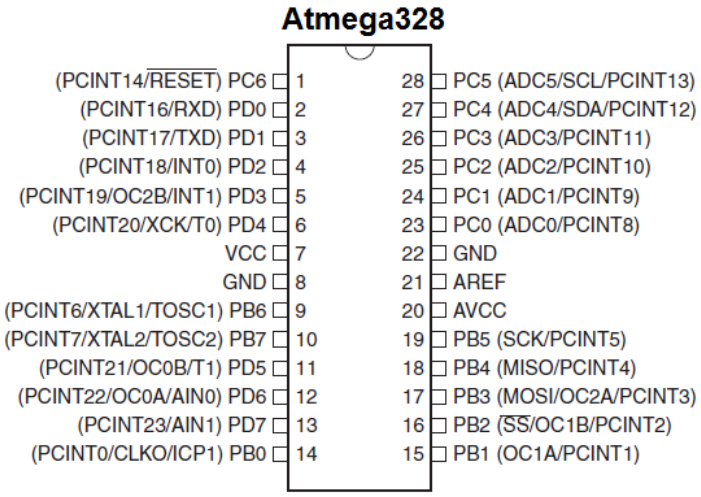
Langage de programmation

Il est basé sur `c++`, consulter le site de ARDUINO pour les informations sur ses principales fonctions. A noter la syntaxe `c,c++` de base, tel que la terminaison obligatoire de ligne par `;` et les commentaires. Tout codage ARDUINO présente la même structure :

— les définitions des constantes, des fonctions, et des variables qui seront utilisées dans tout le programme

— la partie `setup(){...}` qui n'est exécutée qu'une seule fois au tout début du programme.

— le programme principal `loop(){...}` qui est exécuté dans une boucle infinie. Repérez sur votre carte le bouton de redémarrage, dit «reboot», qui permet d'interrompre cette boucle et de revenir au `setup(){...}`.



2.1.3 Programme Blink

Nous allons tester le matériel à disposition en faisant l'expérience d'allumage de la diode électroluminescente (LED) de la carte ARDUINO. Parmi les exemples de codage les plus basiques, le programme se trouve dans FILE ▷ EXAMPLES ▷ BASICS ▷ BLINK

```
/*
  Blink: Turns on/off the onboard LED on for 200 msec
*/

void setup() {                    // runs once when you press reset or power on
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {                      // runs over and over again forever
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(200);                      // wait for a 200 milliseconds
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage LOW
  delay(200);
}
```

On charge ce programme FILE ▷ OPEN dans le logiciel IDE. Pour compiler et vérifier ce programme, nous utilisons SKETCH ▷ COMPILE AND VERIFY

```
Sketch uses 928 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048
    bytes.
```

et pour le faire tourner on passe par COMPILE-UPLOAD (téléverser le programme dans la carte) ou on utilise le raccourci clavier CTRL-U. La LED de la carte doit clignoter à une fréquence de $1/(2 \times 0,200)$ Hz confirmant le fonctionnement de la carte et du logiciel IDE. Vous êtes alors prêts à utiliser l'environnement ARDUINO.

2.1.4 Programme BlinkSOS

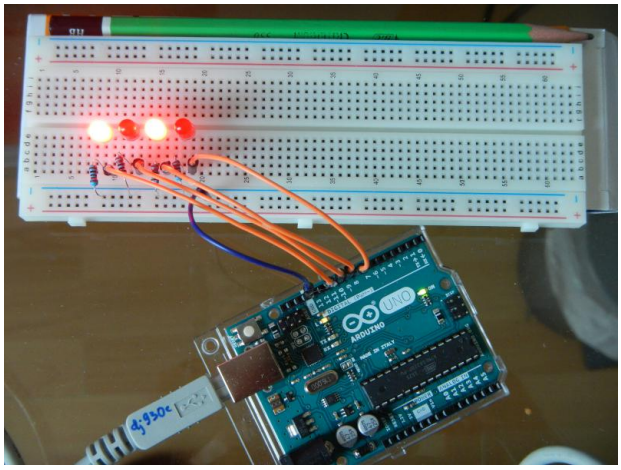
On modifie le programme précédant pour faire clignoter «SOS» en morse. Ce signal de détresse consiste de trois appels courts (lettre S) suivis par trois appels longues (lettre O) puis encore 3 courts. Pour cela, on essaie d'utiliser une boucle `for` :

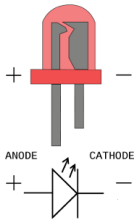
```
 for(i=3; i==0; i--) {
    digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
    delay(200);                        // wait for a 200 milliseconds
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
    delay(200);
  }
  delay(200);
```

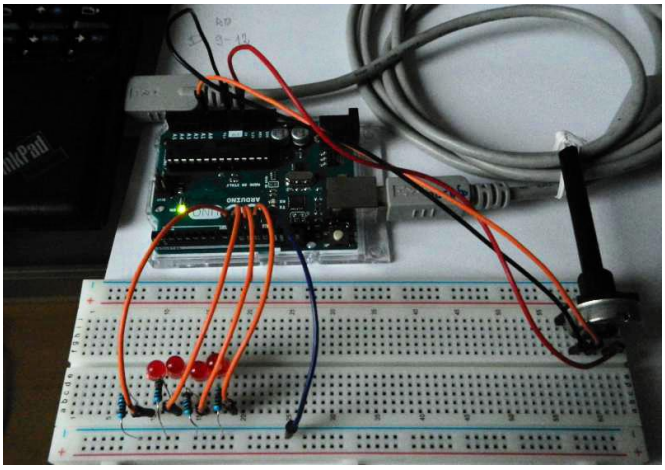
ici `i` est déclarée auparavant comme une variable du type `int` par une commande du type `int i = 0 ;` placée avant le `setup`. L'instruction `for` réalise une boucle et prend plusieurs arguments : `i = 3` indique que l'on s'intéresse aux valeurs de `i` depuis 3 jusque `i = 0`. La dernière instruction (`i--`) est exécutée à la fin de la boucle et elle peut être n'importe quoi. Ici elle décrémente la valeur de `i` par pas de 1. En fait, l'instruction `i--` est équivalente à `i = i -1` mais prend moins de place en mémoire.

2.1.5 Montage à 4 LED's

Les résistances entre les cathodes de LED's et GND sont de 220 Ohm, voir sec. 2.1.5. Dans la partie `setup`, nous pouvons initialiser







les quatre sorties digitales avec une petite boucle :

```
for(j=PIN_BASE,i=4; i--; pinMode(j++, OUTPUT));
```

où l'utilisation des opérations ++ et −− est à noter. Et par la suite, nous pouvons aussi allumer nos LED's une par une

```
for(j=PIN_BASE,i=4; i--; j++) {
   digitalWrite(j, HIGH);   // turn the LED on (HIGH is the voltage level)
   delay(200);              // wait for a 200 milliseconds
   digitalWrite(j, LOW);    // turn the LED off by making the voltage LOW
   delay(200);
 }
```

pour tester notre montage.

Remarque : Le courant maximal qu'on peut tirer des pins digitaux de Arduino à 5V

L'ARDUINOest conçu pour être alimenté de deux manières : soit par le port USB soit par le connecteur externe. Pour le port USB, la limite maximale du courant est de 500 mA. Pour une alimentation externe qui utilise le régulateur +5VDC de la carte, la limite est plutôt fixée par la dissipation de chaleur de ce régulateur et en conséquence, est déterminée par la valeur spécifique de la tension sur le connecteur d'alimentation externe. 3MM mA sera certainement suffisant pour les deux types de sources.

Cependant, une autre limitation est la quantité de courant que les broches d'E/S peuvent gérer à la fois individuellement et la consommation totale de courant de toutes les broches d'E/S ensemble. 200mA est une limite totale et 40mA est une limite maximale individuelle. Ainsi, 200mA pour toutes les broches de sortie est la limite à laquelle vous vous heurtez le plus souvent, et il est préférable de faire fonctionner les broches d'E/S en toute sécurité à environ 20mA maximum.

Remarque : Connexion de LED's

Pour ceux qui n'ont pas suivi d'enseignement d'électronique, une LED est une diode électroluminescente, c'est à dire qui s'éclaire lorsqu'un courant la parcourt. Elle sert beaucoup pour visualiser de façon très simple l'état logique d'une ligne (haut/bas ou vrai/faux). C'est un dipôle asymétrique constitué d'une anode et d'une cathode. Lorsque la diode est polarisée dans le sens direct (+V sur l'anode et 0V sur la cathode), elle est passante, et un courant circule dans la diode. Lorsqu'elle est polarisée en inverse, elle est bloquée, aucun courant ne circule. La figure 2 donne les caractéristiques courant-tension de LED de diverses couleur. Vous remarquerez que lorsque la tension aux bornes de la diode est inférieure à ∼1,5 V, le courant est nul : la diode est bloquée. Il faut que la tension dépasse une certaine valeur appelée *tension de coude* pour que le courant puisse passer. Typiquement, pour les LED que vous avez en TP cette *tension de coude* est de 1,8 V. On notera qu'au delà de la tension de coude, une petite variation de la tension induit une forte variation du courant.

Afin de connecter une LED à un pin digital de l'ARDUINO (tension de sortie 5V), on utilise une résistance R en série avec la LED pour limiter le courant (25 mA). Calculons la valeur de R.

On voit sur la figure 2 que la tension d'alimentation ($V = +5$ V) est égale à la tension V_R aux bornes de la résistance R plus la tension aux bornes de la LED V_{LED}. Une LED rouge typique doit avoir une différence de potentiel V_{LED} de 1.6-1.8V entre ses pins pour un courant, approximativement, de 25mA. Donc

$$V_R = V - V_{LED} = 5V - 1.8V = 3.2V$$

En appliquant la loi d'Ohm, pour $I = 25$mA nous obtenons une résistance de $R = V_R/I = 128$Ohm. Cependant, les tutoriels recommandent d'utiliser 220 Ohm. Pourquoi ? La raison est dans la pratique : en regardant les résistances disponibles facilement, on

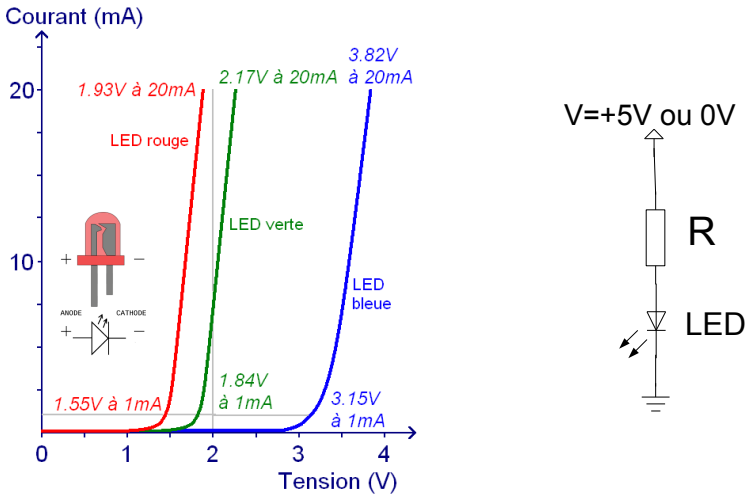


FIGURE 2 – Caractéristiques courant-tension de LED et son montage de base. La cathode est répérée par la patte la plus courte de la diode.

trouve une de 100Ω qui est inférieure à 128Ω que nous avons besoin et est risquée. En effet le courant serait alors de $3,2/100 = 32$ mA soit une valeur de I trop élevée. La valeur usuelle suivante est de 220Ω ce qui nous donne un courant $I = 3.2/220$ d'environs 14mA. Les DEL's sont opérationnelles pour des courants de 10–25 mA. Elles sont des composants non-linéaires, et la différence de courant entre 14 mA et 25 mA n'est pas nécessairement proportionnelle à la différence de luminosité. Dans la plupart des cas, on ne sera même pas capable de voir cette différence.

2.1.6 Interaction sur le port série

Le port série classique est émulé par l'ARDUINO à travers de son interface USB. Pour initier ce port à la vitesse de transfert de 9600 baud (la vitesse typique des anciens terminaux et modems sur le port série RS232), nous ajoutons dans le programme d'initialisation `setup`

```
Serial.begin(9600);
```

Pour communiquer les informations par ce port, une fois que notre carte a été identifiée (comme `ttyACM0` dans ce document, ou `COM1`, `COM2`, etc sous WINDOWS), il faut ouvrir le terminal de l'environment Arduino IDE. Notez, qu'à l'ouverture du terminal, ainsi qu'après le téléchargement d'un nouvel exécutable, la carte se redémarre en exécutant le `setup`.

Sortir les informations : le texte (string) Nous pouvons sortir un message de démarrage, par exemple

```
Serial.println("\nSOS blinking and Serial interaction via keys");
```

Notez, qu'à la différence de `Serial.print` la commande `Serial.println` y ajoute automatiquement le changement de ligne. Pour ces deux commandes, le texte, dit «string», doit être entouré par les signes double de citation `"`.

Le caractère, le byte, le bit Chaque string consiste en un ou plusieurs caractères, et chaque caractère est représenté par un nombre 0..255 (où les caractères imprimables/lisibles démarrent à 32). Un tel nombre occupe un *byte ou octet*, c. à d., *huit* registres binaires 0/1 dits *bits* ($255 = 2^8 - 1$). Dans le langage `c, c++`, on déclare ces nombres avec le type `char` ou plus précisément `unsigned char`. Il existe plusieurs façons de définir la correspondance (codage) entre les caractères et ces nombres. Actuellement, le plus répandu et simple est le codage ASCII.

NB : les types représentant les nombres entiers A la différence de `unsigned char`, le type `char` définit les nombres entiers dans l'intervalle $-127 \ldots 127$ qui occupent aussi un byte dont les 7 bits inférieures gardent la valeur ($127 = 2^7 - 1$) et le $8^{\text{ième}}$ bit sert à donner le signe. Les nombres entiers, quant à eux, sont définis par la classe `int` ou encore `unsigned int` et occupent *deux* bytes. Par conséquent, ces nombres vont jusqu'à $\pm(2^{15} - 1)$ ou bien $2^{16} - 1$.

Entrée d'information Pour envoyer des informations à la carte, nous les entrons dans la ligne de commande du terminal suivies par la touche «Enter» qui les envoie. Les informations sont transmises comme un texte qui est mis en attente dans le tampon de l'interface série/usb dit «buffer». Pour savoir si les informations nous attendent à l'entrée, on appelle `Serial.available()`

Montage du potentiomètre et Convertisseur A/N (ADC) On branche les extrémités d'un potentiomètre de $R = 10$ à 100 KOhm entre GND et la sortie 5V coté analogique ; le connecteur amovible se branche à l'entrée analogique, par exemple A0. Le courant consommé de $V/R = 5/10^4 = 5\text{mA}$ est inférieur à celui que Arduino peut soutenir, voir sec. 2.1.5. La programmation est simple, voir l'exemple code «AnalogInput» dans FILE ▷ EXAMPLES ▷ ANALOG. La partie essentielle du code est

```
int sensorPin = A0, sensorValue = 0;
sensorValue = analogRead(sensorPin);
```

où nous appelons `analogRead` pour lire la tension présente sur la broche A0 (l'entrée analogique). Cette fonction effectue la conversion de la tension analogique en signal numérique (A/N) avec un convertisseur à 10 bits pour la plupart des microcontrôleurs ARDUINO. Cela signifie que le signal numérisé peut prendre $2^{10} = 1024$ valeurs comprises entre 0 et 1023, 1023 correspondant à 5V.

On transforme ainsi la tension présente sur l'entrée A0 en une valeur lisible et exploitable par le microcontrôleur. La conversion consiste à comparer cette tension à une valeur de référence fixée à 5 V par défaut. Il est possible de comparer par rapport à une autre valeur de tension comprise entre 0 et 5 V qui est appliquée à la borne Aref de la carte. Il faut alors le spécifier lors de l'utilisation de la commande `analogReference(reftype)`.

Programme final (avec option potentiomètre) On trouve les modifications sauvegardées dans `~/Arduino/BlinkSOS/BlinkSOS.ino`.

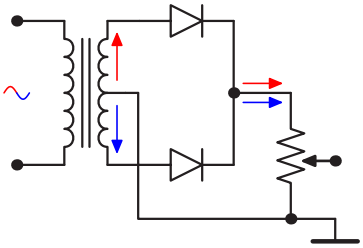
```
/*
  Blink SOS
  Turns on/off the onboard LED on for 200 msec three times, then for 1 sec three times, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino model, check
  the Technical Specs of your board  at https://www.arduino.cc/en/Main/Products

  This example code is in the public domain.

  modified 8 May 2014
  by Scott Fitzgerald

  modified 2 Sep 2016
  by Arturo Guadalupi

  modified 8 Sep 2016
  by Colby Newman

  modified 3 Aug 2017
  by Dima Sadovskii
*/

// NB: use a PWM (~) pin to see analog fading effect, pin 8 is not suitable
#define PIN_BASE 9  // starting digital pin number for 4 external LED's
#define PIN_ANLG A0 // pin number for digital entry
// the setup function runs once when you press reset or power the board
void setup() {
  int j,i;
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
  // initialize 4 external digital pins starting in sequence from PIN_BASE as output
  for(j=PIN_BASE,i=4; i--; pinMode(j++, OUTPUT));
  // light the LED's in a sequence
  for(j=PIN_BASE,i=4; i--; j++) {
    digitalWrite(j, HIGH);   // turn the LED on (HIGH is the voltage level)
    delay(200);              // wait for a 200 milliseconds
    digitalWrite(j, LOW);    // turn the LED off by making the voltage LOW
    delay(200);
  }
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  // send an intro:
  Serial.println("\nSOS blinking and Serial interaction via keys");
  // Serial.println();
}

// parse a single byte character key as a hexadecimal digit
unsigned int hexToByte (char c) {
  if ( (c >= '0') && (c <= '9') ) {
```

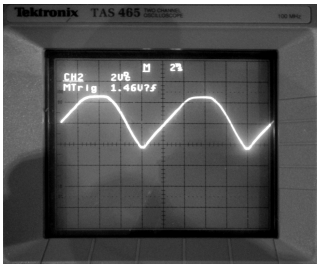
```
    return (int)(c - '0');
  }
  if ( (c >= 'A') && (c <= 'F') ) {
    return (int)(c - 'A')+10;
  }
  if ( (c >= 'a') && (c <= 'f') ) {
    return (int)(c - 'a')+10;
  }
}

int aold=-1; // last read analog input

// the loop function runs over and over again forever
void loop() {
  int i,j,hex,aval=0;
  char key;
  aval = analogRead(PIN_ANLG);
  if(aval>0) {                          // treat nonzero analog input
    Serial.print(aold);
    Serial.print(" V=");
    Serial.println(aval);
    aold = aval;                        // keep track of last analog input
    for(i=4; i--; digitalWrite(i+PIN_BASE,(aval>=i*256)?HIGH:LOW));
  }
  else { // show SOS signal
    if(aold>0) {
      aold=0;
      Serial.println("stop analog entry");
      for (i=255 ; i >= 0; i -= 5) {    // fade out from max to min in increments of 5 points (range from 0 to
          255):
        analogWrite(PIN_BASE, i);       // NB: only possible for a PWM pin, e.g. 9 (but not 8!)
        delay(20);                      // wait for 20 msec to see the dimming effect
      }
      //digitalWrite(PIN_BASE,LOW);
    }
    for(i=3; i; i--) {
      digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
      if(analogRead(PIN_ANLG)||Serial.available()) break;
      delay(200);                       // wait for a 200 milliseconds
      digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
      delay(200);
    }
    delay(200);
    for(i=3; i; i--) {
      digitalWrite(LED_BUILTIN, HIGH);
      if(analogRead(PIN_ANLG)||Serial.available()) break;
      delay(1000);                      // wait for a second
      digitalWrite(LED_BUILTIN, LOW);
      if(analogRead(PIN_ANLG)||Serial.available()) break;
      delay(1000);
    }
  }
  // Read serial input:
  if(Serial.available() > 0) {
    key = Serial.read();
    Serial.print("key ");
    Serial.print(key);
    Serial.print(" with ASCII ");
    Serial.print(int(key));
    Serial.print("=0x");
    Serial.print(key,HEX);
    if(isHexadecimalDigit(key)) {     // use isDigit for decimal input
      //if Hexadecimal display the numerical value of the key
      Serial.print(" gives HEX value ");
      Serial.print(hex=hexToByte(key));
      Serial.print(" and BIN code ");
      for(i=8,j=PIN_BASE+4;i>0;i=i>>1) {
        Serial.print(((hex&i)?1:0));
        digitalWrite(--j,((hex&i)?HIGH:LOW));
      }
    }
    Serial.println();
  }
}
```

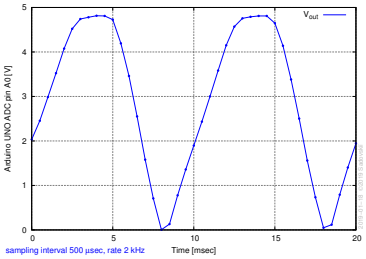
2.1.7 Échantillonnage rapide avec Arduino ADC

L'échantillonnage en temps réel avec des intervalles de temps équidistants est une tache très courante dans de nombreuses applications [1]. Le temps de mesure d'un échantillon par l'ADC de Arduino UNO étant de 100μsec, le taux maximal théorique peut atteindre 10kHz. Cependant, le microcontrôleur possède peu de mémoire (2K) pour sauvegarder les données acquises. L'alternative est de transférer les mesures immédiatement par son port série. Dans ce cas, on doit compter le temps de transfert. A la vitesse de 115200 baud, ce temps est estimé à 70 μsec/byte. Ainsi, si on utilise le mode texte (ASCII), un nombre inférieur à 1024 est représenté par quatre caractères, chacun étant codé sur un byte. Il nous faudra donc *quatre* bytes maximum en format décimal, ou bien *trois* bytes en format hexadécimal pour les valeurs inférieures à 1024 [2]. A ceci on ajoute le caractère de terminaison de ligne (LF sous Linux). Donc en tout, nous allons avoir besoin de quatre bytes minimum, et le temps pour un échantillon s'élève à $100 + 4 \cdot 70 \approx 400$ μsec. En conclusion, dans cette méthode, il nous sera possible d'échantillonner aux fréquences inférieures ou égales à 2.5kHz. Dans le cas de transfert binaire, avec seulement *deux* bytes par échantillon, ce taux remonte à 4kHz.

Nous allons échantillonner un signal AC de 0..5V et de fréquence 100 Hz avec l'ADC. On peut utiliser un GBF comme source, mais à l'occasion ici, pour prendre un exemple, nous avons pris un ancien chargeur des piles NiCd et un potentiomètre variable («Load», voir la figure) de 10kΩ pour régler la tension de sortie V_{out} à 0..5V. Le chargeur utilise un transfo et deux diodes (two diode rectifier), les bornes A0 (entrée analogique 0) et GND de Arduino sont branchées sur le potentiomètre.





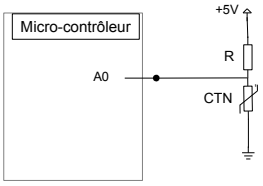


Une fois téléchargé dans le microcontrôleur, le programme `~/Arduino/adcsampler/adcsampler.ino` ci-dessous permet de communiquer avec lui [3] en utilisant les clés T, N, H, et S, pour définir l'intervalle de l'échantillonnage (`READ_PERIOD`), le nombre des échantillons (`npnt`), ainsi que le format (décimal ou hex), puis démarrer les mesures. Par exemple, avec `T500N100HS` on définit l'intervalle de 500μsec, 100 échantillons et format hex. La figure ci-dessus à droite montre le résultat sous forme graphique.

3 Les capteurs

Le microcontrôleur est réellement adapté pour l'utilisation de capteurs au sens large. Il s'agit bien souvent de mesurer une grandeur électrique (tension ou courant) ou un temps qui sont proportionnels à la grandeur physique que mesure le capteur. Nous vous proposons d'utiliser et de mettre en oeuvre un capteur de température basé sur une thermistance et un capteur de lumière basé sur l'utilisation d'une photo-résistance. Vous mettrez en place la liaison série vers le moniteur série pour visualiser les données de votre capteur.

3.1 Thermistance



La thermistance (thermistor en anglais) dont vous disposez est une CTN (Coefficient de Température Négative), c'est à dire que la résistance diminue avec l'augmentation de la température. Il existe également des CTP (Coefficient de Température Positif). Elle a une résistance qui varie en fonction de la température suivant une loi définie par un polynôme de degré 3 donné par la relation de *Steinhart-Hart* :

$$\frac{1}{T} = a + b\,\log(R_T) + c\,\left[\log(R_T)\right]^3$$

avec trois coefficients phénoménologiques du polynôme (a, b, c). On peut également utiliser une approximation de cette formule, donnée pour une plage de température plus restreinte :

$$\frac{R_T}{R_0} = \exp\left[B\left(\frac{1}{T} - \frac{1}{T_0}\right)\right]$$

Les valeurs de B sont référencées par le fabricant pour une gamme de température donnée en °K. La thermistance que vous avez à votre disposition présente une résistance de $100\,k\Omega$ à 25°C

En utilisant le $+5$V de la carte, réalisez un montage «diviseur de tension» comme indiqué sur la figure ci-dessus, en prenant $R = 100\,k\Omega$. Vous utiliserez l'entrée analogique A0 de la carte pour mesurer la tension aux bornes de la thermistance CTN. Notez que ce

1. comme une application possible, voir TP «réaction oscillante»
2. il s'agit ici du nombre maximal de bytes nécessaires, pour les petites valeurs ce nombre décroît
3. en utilisant le moniteur série, la vitesse du port série est réglée à 115200 baud

montage est déjà rencontré dans la sec. 2.1.6. En fait, vous pouvez remplacer le potentiomètre de l'ancien montage, et exploitez par la suite le même code en le modifiant pour afficher des valeurs de T. Pour une variation de température de 25 à 30°C, calculez les valeurs de thermistance attendues et les valeurs de tension correspondantes sur l'entrée A0 de la carte.

3.2 Photo-résistance

La conductivité d'un matériau et donc sa résistance, est fonction du matériau lui-même mais également de son environnement : la température comme dans le cas de la thermistance (sec. 3.1) ou encore de la luminosité. C'est cette propriété que nous utilisons ici avec la photo-résistance.

La luminosité ambiante peut être mesurée en lux. 1 lux correspond à un niveau de lumière en pleine nuit et 1000 lux en pleine journée. La valeur de la résistance d'une photo-résistance suit une loi en fonction de la luminosité L du type :

$$R(L) = R_0\, L^{-k}$$

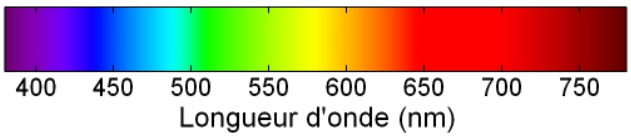
où R_0 et k sont des constantes. Ainsi la courbe $\log R = f(\log L)$ est une droite avec une pente négative si bien que la valeur de la résistance diminue avec L depuis quelques centaines de kΩ dans l'obscurité jusque quelques centaines d'Ω en plein jour, ceci bien évidemment, en fonction des modèles. Comme les thermistances, les photorésistances sont le plus souvent utilisées dans un pont diviseur de tension.

4 Colorimétrie

Nous combinons les expériences et le code développé dans les sec. 2.1.5, 2.1.7, et 3.2 pour construire et tester un colorimètre (spectrophotomètre monochromatique) permettant de détecter des molécules spécifiques et de mesurer leurs concentrations en temps réel.

4.1 Absorption de la lumière, loi de *Beer-Lambert*

Considérons un faisceau de lumière de jour traversant un milieux actif tel qu'une solution aqueuse *colorée*. La lumière de jour étant blanche, elle possède toutes les couleurs (toutes les longueurs d'onde λ) du spectre visible ci-dessous.



On conclut que le milieux retient ou *absorbe* la partie du spectre complémentaire à sa couleur : un objet transparent vert laissera passer le vert et arrêtera toutes les autres couleurs. Par ailleurs, il est clair que l'intensité I de la lumière absorbée diminue avec l'épaisseur l du milieux. La relation entre I et l est donnée par la loi *Beer-Lambert*.

Sous sa forme différentielle, cette loi prédit que la quantité de la lumière absorbée $\mathrm{d}I$ par l'épaisseur $\mathrm{d}l$ est

$$\mathrm{d}I = I(l+\mathrm{d}l) - I(l) = -\varepsilon\, C\, I(l)\, \mathrm{d}l\ . \tag{4.1a}$$

On constate que l'absorption est un phénomène forcé, provoqué, induit par l'intensité I de la lumière traversant le milieux : plus on a de la lumière, plus il y a de la lumière $\mathrm{d}I$ absorbée. Le coefficient de proportionnalité comprend naturellement la concentration C [Mol/L] des molécules actives et leur caractéristique individuelle ε [L/Mol/cm] appelée le *coefficient d'absorption* ou l'*absorbance*[4]. La forme (4.1a) s'applique aux petites épaisseurs $\mathrm{d}L \approx \Delta l$. Sinon, on doit utiliser la loi intégrale

$$\log I(l) - \log I(0) = -\varepsilon\, C\, l \quad \Rightarrow \quad I(l) = I_0 \exp(-\varepsilon\, C\, l)\ . \tag{4.1b}$$

Cependant, pour des petites variations de $x = l\, C$, la forme (4.1b) est excessive et difficilement applicable, surtout en présence d'erreurs de mesure. Ainsi, si x varie entre $x_{\min}$ et $x_{\max}$, nous utiliserons plutôt une *linéarisation* de (4.1b)

$$I(l) \approx I_0\ \exp(-\varepsilon\, \bar{x})\big(1 - \varepsilon\,(x - \bar{x}) + \dots\big) \quad \text{avec } \bar{x} = (x_{\max} - x_{\min})/2. \tag{4.1c}$$

Enfin, dans une étude avec une épaisseur fixe et une concentration C qui est fonction de I, on réécrit (4.1c) sous forme

$$C(I) = a_0 + a_1\, I \tag{4.1d}$$

4. selon les domaines d'applications spécifiques, on rencontre aussi l'*absorptivité*, la densité optique du milieu, l'opacité ou l'extinction.

4.2 Régression linéaire

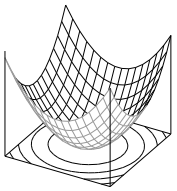
Considérons les mesures (x_i, y_i) avec $i = 1, \ldots, N$ pour chaque valeur de x. Dans notre cas (sec. 4.1), y et x représentent l'intensité I et la concentration C. On estime les incertitudes de y_i égales à Δy_i (ou à ces écart types σ_i), et on suppose dans le cas de la *régression*, que les valeurs de x sont "exactes". Ainsi x est appelée *variable de contrôle*. On étude le phénomène décrit par la loi théorique $y = f(x;\alpha)$ avec des paramètres phénoménologiques à déterminer $\alpha = \alpha_1, \alpha_2, \ldots$. Dans notre cas, il s'agit de l'approximation linéaire (4.1d) avec ses deux paramètres $\alpha = (a_0, a_1)$. En d'autres termes, nous cherchons à faire passer une *ligne droite* appelée aussi «courbe de calibration» par les points $(x_i, y_i) = (C_i, I_i)$ à partir de mesures de I_i pour des échantillons de concentration C_i connue. Ces échantillons de concentration connue sont appelés les «standards». Bien entendu, leur nombre N doit être égale ou être supérieur au nombre des paramètres α (donc pour nous $N \geq 2$). Par la suite, en connaissant les α, nous pourrons déterminer la valeur de C pour toute valeur de I dans l'intervalle $I_{\min} \ldots I_{\max}$ couvert par (4.1d).

Méthode des moindres carrés L'idée centrale de cette méthode (appelée *least squares fit* en anglais) est de trouver les valeurs de paramètres α, pour lesquelles la déviation moyenne quadratique entre la théorie et l'expérience

$$F(\alpha) = \sum_{i=1}^{N} \left(\frac{y_i - f(x_i, \alpha)}{\sigma_i} \right)^2$$

devient minimale. Ici les écarts types σ_i de chaque y_i jouent le rôle de "poids" : ils permettent de donner plus "d'importance" aux mesures plus précises. Dans notre cas

$$\min_{a_1,a_0} F(a_1, a_0) = \min_{a_1,a_0} \sum_{i=1}^{N} \left(\frac{y_i - (a_1 x_i + a_0)}{\sigma_i} \right)^2 .$$



En introduisant les «moyennes pondérées»

$$[g] := \frac{1}{S} \sum_{i=1}^{N} \frac{g_i}{\sigma_i^2}, \quad \text{où} \quad S = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \quad \text{et} \quad g_i = x_i, y_i, x_i y_i, x_i^2 \tag{4.2a}$$

(devenant les moyennes $[g] = \bar{g}$ dans le cas simple où tous $\sigma_i \equiv \sigma$), on trouve les valeurs (voir le code `linreg` dans la sec. 4.3)

$$a_1 = \frac{[xy] - [x][y]}{[xx] - [x][x]}, \quad a_0 = [y] - [x]\, a_1 \tag{4.2b}$$

qui minimisent $F(a_1, a_0)$ avec leurs écarts types correspondant [5]

$$\sigma_{a_1}^2 = \frac{S^{-1}}{[xx] - [x][x]}, \quad \sigma_{a_0}^2 = \sigma_{a_1}^2 [xx]. \tag{4.2c}$$

Démonstration. Au minimum de $F(a_1, a_0)$, on a $dF = 0$. On obtient le système de deux équations linéaires de variables (a_1, a_0)

$$\left\{ \begin{array}{l} \dfrac{\partial F}{\partial a_1} = 0 \\ \dfrac{\partial F}{\partial a_0} = 0 \end{array} \right. \Rightarrow \left\{ \begin{array}{l} \displaystyle\sum_{i=1}^{N} x_i (y_i - (a_1 x_i + a_0)) \sigma_i^{-2} = 0 \\ \displaystyle\sum_{i=1}^{N} (y_i - (a_1 x_i + a_0)) \sigma_i^{-2} = 0 \end{array} \right. \Rightarrow \left\{ \begin{array}{l} [xy] - a_1 [xx] - a_0 [x] = 0 \\ [y] - a_1 [x] - a_0 = 0 \end{array} \right.$$

dont la solution est donnée par (4.2b). □

4.3 Colorimètre avec ARDUINO

Nous utilisons une photo-résistance (sec. 3.2), le branchement et les contrôles de la LED (sec. 2.1.5), et les principes de l'échantillonnage ADC rapide aux intervalles de temps fixes (sec. 2.1.7) pour construire un colorimètre. Ce dispositif sera capable de mesurer des concentrations avec les périodes de 300 μsec (plus prudemment 500 μsec, voir sec. 2.1.7) et donc au taux maximal de 2-3 kHz. Les formules (4.2b) seront implémentées dans le code de calibration de colorimètre.

Matériel : une carte ARDUINO UNO et un ordinateur pour lui communiquer en TTY, une LED rouge générique (de λ approximativement 630 ou 660 nm), des résistances de 220Ω et $R' = 10\text{k}\Omega$, une photo-résistance de $R_{\min} = 3.1\text{k}\Omega$ exposée par la LED rouge, voir la fig. 3, et $R_{\max} \approx 0.35\text{M}\Omega$ sans lumière, un potentiomètre de 10kΩ en option. Carton, colle, scotch, feutre noir ...

5. Dans le cas simplifié $S^{-1} \approx \sigma^2 N^{-1}$, et par conséquence, les incertitudes Δa_1 et Δa_0 décroissent comme $1/\sqrt{N}$. On retrouve ainsi un résultat fondamental.

Le principe : La LED et la photo-résistance sont montées sur le breadbord à une distance d'approx. 15mm. Pour gagner en stabilité, on utilise la source interne de 3.3V comme VCC (plus filtré et plus stable que les 5V fournis par l'ordi sur son port USB) pour alimenter la LED et surtout le diviseur de tension de la photo-résistance $R + R'$. Le potentiomètre peut servir à adapter la valeur de la tension de référence ($V_{\max}$ anticipée sur la R) inférieure à 3.3V. La LED envoie sa lumière vers la R, éventuellement, à travers une cuvette. On évite toutes lumières parasites (caches).

FIGURE 3 – Montage du colorimètre : (a) mesure de $R_{\min}$ de la photo-résistance ; (b)-(c) avec $CuSO_4$ dans une cuvette.

Le code et l'interface : `~/Arduino/cmeter/cmeter.ino`

```
/*
  Simple precise fixed-time-interval ADC sampler used as colorimeter;
  can sample at periods larger than 300 usec (safer 500 usec, see code)
  and thus has maximal theoretical sampling rate of 3kHz (more like 2kHz)

  TP: examine the relationship between the absorbance and concentration
  of a copper (II) sulfate CuSO4 solution (Beer's law) and measure
  the concentration of unknown copper (II) sulfate solution samples.
  The molar absorptivity of CuSO4 at lambda_max=635 nm is 2.81/M/cm.
  see more in http://dvoirah.ovh/etudes/Arduino/BQE-Arduino.pdf

  minimum parts required (in addition to Arduino UNO and computer):
    generic red LED (around 630 or 660 nm), 220 and 10k Ohm resistors
    photoresistor (3.1k Ohm exposed by red LED, closed at 0.35M Ohm)
    the LED-photoresistor distance on the breadbord is about 15mm
    use Arduino's 3.3V supply as VCC (more stable) to power the photoresistor
    use generic colorimeter 12.55x12.65x44.55 plastic cuvettes (with caps)
    for CuSO4 standard solutions of n/10 M with n=0,1,2,3,4,5 and 2-3 controls

  On interfacing with linux serial port, normally at /dev/ttyACM0, see
  https://playground.arduino.cc/Interfacing/LinuxTTY
  stty -F /dev/ttyACM0 cs8 115200 ignbrk -brkint -icrnl -imaxbel -opost -onlcr -isig -icanon -iexten -echo -
      echoe -echok -echoctl -echoke noflsh -ixon -crtscts
  stty -F /dev/ttyACM0 115200 cs8 cread clocal
  screen /dev/ttyACM0 115200

  When doing analog readings, especially with a 'noisy' board like the Arduino,
  we suggest two tricks to improve results. One is to use the 3.3V voltage pin
```

```
  as an analog reference, and the other is to take several readings in a row
  and average them. The 3.3V goes through a secondary filter/regulator stage
  and is less noisy to use it, connect 3.3V to AREF (next to digital GND).
  Several measurements can be averaged in time-independent or slow systems.

*/
#define _DEBUG_                     /* debugging, undefine to save space */
#include <EEPROM.h>                 /* for nonvolatile memory access */

#define ADC_PIN A0                  /* pins A0..A5 for analog 10-bit ADC */
#define LED_PIN 10                  // digital pin for controlling LED source
// NB: use a PWM (~) pin to use analog fading effect, e.g. pin 8 is not suitable
// TODO: use PWM to control LED intensity
#define DIVISOR_R 10000             // 10K Ohm resistor in series with photo-R
#define STD_MAX 8                   // max number of calibration standards (>=2)

/* TODO: keep calibration coeffs and settings in nonvolatile memory */
double a1_coeff=.1;                 // linear regression coefficients
double a0_coeff= 0;
double linreg_stderr=0;             // error of linear regression approximation
unsigned int  npts = 0;             // number of samples to measure
uint8_t apts = 2;                   // number of samples for averaging each point
unsigned long READ_PERIOD = 4000;  // 4000 us gives 250 Hz sampling rate

unsigned long lastRead = 0;         // time in us of the durrent point sequence
uint8_t hex_format=0;               // default raw integer output format
uint8_t std_num=0;                  // num of calibration data points (standards)
unsigned int std_C[STD_MAX], std_R[STD_MAX];

double Rval(unsigned int adc) {     // ADC data -> resistance of photo resistor
 return( DIVISOR_R / (1024 / ((double) adc) - 1) ); /* 10K / (1023/ADC - 1) */
}

double Cval(unsigned int adc) {     // ADC data -> concentration
 return(a1_coeff * ((double) adc) + a0_coeff);
}

double get_stderr() {               /* compute std error of linear regression */
 unsigned int *x=std_R, *y=std_C;  /* for the current calibration data set   */
 int k; double s=0,f;
 if(std_num>1) {                    /* check if there's enough data */
   for(k=std_num; k--; s+=f*f) f = Cval(*x++) - *y++;
   s /= (double) std_num;           /* unshifted normalized standard error   */
 }
 return(sqrt(s));                   /* sqrt( (sum_i (f(x_i) - y_i)^2) / N ) */
}

double linreg() {                   /* linear regression of current std data */
 unsigned int *x=std_R, *y=std_C;
 int k; double xx,_y,_x,xy;
 if(std_num>1) {                    /* check if there's enough standard data */
   for(k=std_num, _x  = _y = xx = xy = 0; k--; _x += *x++, _y += *y++) {
     xx += ((double) *x) * *x;                  /* average of x^2 */
     xy += ((double) *x) * *y;                  /* average of x*y */
   }
   a1_coeff = (xy*std_num - _x*_y) / (xx*std_num - _x*_x); /* slope of y(x) */
   a0_coeff = _y - _x * a1_coeff;
   a0_coeff/= (double) std_num;                 /* shift of y(x) */
   linreg_stderr = get_stderr();                /* resulting std error */
  /* TODO: store calibration coeffs in nonvolatile memory */
 }
 return(linreg_stderr);
}

unsigned long min_val=0, max_val=1023;          /* 10-bit ADC value bracket */
void setup() {
  /* LED_BUILTIN is set to the correct LED digital pin for the board in use;
     it is 13 on the UNO, MEGA and ZERO, 6 on MKR1000. */
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);  // turn the onboard LED on = "busy"
  /* set up faster serial communication (instead of default 9600bps) */
  Serial.begin(115200); // 115200 bps = 14400 bytes/sec, 70 usec/byte
  /* determine actual bracket of ADC values with light source on/off */
  pinMode(LED_PIN, OUTPUT);          // LED light-source controlling pin
  digitalWrite(LED_PIN,  LOW);       // make sure the LED is off
  delay(50);                         // wait for a 50 msec and measure
```

```
  max_val = analogRead(ADC_PIN);    // Vmax for maximum resistance (dark)
  digitalWrite(LED_PIN, HIGH);      // turn the LED on
  delay(200);                       // wait for a 200 msec and measure
  min_val = analogRead(ADC_PIN);    // Vmin for minimum resistance (max light)
  while (!Serial) {
    ; /* wait for serial port to connect (for native USB port only) */
  }
  // connect AREF to 3.3V and use that as VCC, because it is less noisy!
  analogReference(EXTERNAL);
  /* TODO: restore calibration coeffs and settings from nonvolatile memory */
  Serial.print("# real time colorimeter ");
  Serial.print(min_val);
  Serial.print("..");
  Serial.print(max_val);
#ifdef _DEBUG_
  Serial.print(" (debug)");
#endif
  Serial.println();
  digitalWrite(LED_BUILTIN, LOW);   // onboard LED off (end of setup)
}

void loop() {
 static unsigned int ncnt=0;
 static uint8_t acnt=0;
 static char key='Q';
 static unsigned long val=0,sval=0;
 char r,cflag=0;
  while(Serial.available()) { // read settings
    r = Serial.read();
    if(isDigit(r)) {
      val*=10;
      val+= r-'0';
    }
    else {
      switch(key) {  // keys that preceed numerical values
        case 'T':    // set sampling period (usec)
          if(val) READ_PERIOD = val;
          break;
        case 'D':    // concentration value for calibration data (standards)
    std_num=0; // reset calibration data storage
  case 'd':    // prepare new calibration point
    if(std_num >= STD_MAX) std_num--; // prevent storage overflow
    std_C[std_num] = val;             // enter calibration data y(x) >= 0
    std_R[std_num] = 0;               // clear x data (raw ADC values)
    cflag = 1;                        // calibration flag on
          break;
        case 'A':    // set number of averaged points
          if(val) apts = val;
          break;
        case 'N':    // set number of samples
          if(val) npts = val;
          /* TODO: store READ_PERIOD and/or npts to nonvolatile memory */
      }
      val=0;
      key=r;
      switch(key) {  // switches
        case 'h':    // set hex format
    hex_format =0;
        case 'H':    // toggle hex format (legacy)
          hex_format^=1;
          break;
        case 'c':    // concentration format toggle
          hex_format^=2;
          break;
        case 'r':    // resistance format toggle
          hex_format^=4;
          break;
        case 'C':    // (re)calibrate using currently stored data
          if(std_num>1) {
      Serial.print("# N=");    // number of points (>=2)
      Serial.print(std_num);
      Serial.print(" sigma="); // standard error
      Serial.print((unsigned int) floor(linreg()+.5));
      Serial.print(" a0 = ");  // shift
      Serial.print(a0_coeff);
      Serial.print(" a1 = ");  // slope
```

```
      Serial.print(a1_coeff);
      Serial.println();
#ifdef _DEBUG_
      for(acnt=0; acnt < std_num; acnt++) {
        Serial.print("# (V,C)_");
        Serial.print(acnt+1);
        Serial.print(" = (");
        Serial.print(std_R[acnt]);
        Serial.print(",");
        Serial.print(std_C[acnt]);
        Serial.print(") -> ");
        Serial.print((int) round(Cval(std_R[acnt])-std_C[acnt]) );
        Serial.println();
      }
#endif
    }
    else
      Serial.println("# Insufficient data for calibration");
          break;
  case 'X':    // toggle the LED on/off by bringing the voltage level HIGH/LOW
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    delay(50);                      // wait for a 20 milliseconds
    break;
        case 'S':    // start sampling by rewinding npts
          ncnt=npts;
    acnt=apts;
    sval=0;
    if(!digitalRead(LED_PIN)) {     // make sure the LED is on
      digitalWrite(LED_PIN, HIGH);  // turn the LED on
      delay(50);                    // wait for a 50 milliseconds
    }
        default:
          Serial.print("# T=");
          Serial.print(READ_PERIOD);
          Serial.print(" usec, samples=");
          Serial.print(npts);
          Serial.print("/");
          Serial.print(apts);
          Serial.print(" @ ");
          Serial.print(1000000.0 / READ_PERIOD);
          Serial.print(" Hz, err=");
    Serial.print((unsigned int) floor(linreg_stderr+.5));
          Serial.println();
        case 'D':
        case 'd':
        case 'A':
        case 'N':
        case 'T':
        case 'Q':
          break;
      }
    }
  }
  if(ncnt) {
    /* acquire and display npts samples. ATTN: on 16MHz boards, the time
       resolution is 4 usec, and ncnt is overrun in approx 70 min */
    digitalWrite(LED_BUILTIN,HIGH);// onboard LED on = "busy"
    for(sval=ncnt=0, acnt=apts, lastRead=micros()+8; ncnt<npts; ncnt++){
      while(micros()<lastRead);     // wait for the next read
      lastRead += READ_PERIOD;
      val = analogRead(ADC_PIN);    // 1024 ADC takes about 100 usec (10kHz)
      acnt--; sval+=val;            // accumulate data
      if(!acnt) {                   // display every apts values only
  val = (unsigned long) (sval / apts); // average of apts measurements
  sval=0; acnt=apts;
  if(cflag) {
    std_R[std_num]=val;       // store raw value of x for calibration
    std_num++;              // add new point
  }
  switch(hex_format) {        // modify the raw value if necessary
    case 2:                   // concentration from linear regression
    case 3:                   // (may be negative when close to 0)
      val = (unsigned long) abs(round(Cval(val)));
      break;
    case 4:
    case 5:                   // resistance in Ohm
```

```
      val = (unsigned long) Rval(val);
      break;
  }
  if(hex_format&1)
    Serial.println(val,HEX); // at least 200 usec for 3 bytes
  else
    Serial.println(val);
  if(cflag) {
    cflag=0;
    ncnt=npts-1;              // disable any subsequent sampling
#ifdef _DEBUG_
    Serial.print("# C[");
    Serial.print(std_num);
    Serial.print("] = ");
    Serial.print(std_C[std_num-1]);
    Serial.println();
#endif
  }
      }
    }
    ncnt -= npts;
    digitalWrite(LED_BUILTIN, LOW);// onboard LED off = "idle"
  }
}
```

Une fois téléchargé dans le micro-contrôleur, ce programme permet d'opérer et de communiquer avec le colorimètre (voir également la sec. 2.1.7 et le code). Ainsi `A5N10` préconise d'afficher deux mesures de 5 échantillons moyennés chaque (lisez le code !). Le taux de l'échantillonnage est réglé par la période `T` en μsec. Les mesures sont démarrées par `S`. On allume/éteint la LED avec `X`. Les points de calibration successifs sont introduits par `d`, et on peut réinitier l'ensemble de ces points avec `D`. Pour effectuer la calibration et déterminer les valeurs de a_0 et a_1 on utilise `C`. Par exemple, la commande `d500S` ajoute le point dont la valeur de concentration est 500. A noter, qu'on cible l'intervalle d'affichage à 4 chiffres. Pour cela on implémente des facteurs dans l'échelle de C. Ainsi 0.5M devient 500 mM ou même 5000 10^{-4}M. Par défaut, les valeurs $0 \ldots 1023$ de l'ADC sont affichées. Pour les convertir en concentration C, utilisez `c`. Par ailleurs, la commande `r` permet d'afficher la valeur de la résistance de R.

4.4 TP Mesures de concentration

Objectif : examiner la relation entre l'absorbance et la concentration du sulfate de cuivre (II) $CuSO_4$ dans une solution aqueuse (la loi de *Beer*) et déterminer la concentration dans des échantillons de concentration inconnue.

Sur le spectre d'absorption d'une solution de 0,5M de $CuSO_4$ dans l'eau à température ambiante, on peut voir que l'absorption est maximale à la longueur d'onde $\lambda_{max} = 780$ nm, avec un coefficient d'absorption molaire ε de ≈ 12.5 mol^{-1}cm^{-1} ; à 635 nm il est de 2.8 mol^{-1}cm^{-1}.

Matériel : des cuvettes plastiques génériques pour les colorimètres et spectrophotomètres (4ml, $12.55 \times 12.65 \times 44.55$ mm) avec couvercles hermétiques pour les solutions standardisés de $CuSO_4$ de la concentration $n\,10^{-1}$ M avec $n = 0, 1, 2, 3, 4, 5$ et 2-3 solutions contrôle

Mode opératoire : Calibrez votre appareil avec les solutions standards. Représentez les données de calibration et la courbe sur un graph (tableur ou papier millimétrée) et confirmez la linéarité et le fonctionnement du code `linreg()`. Mesurez la solution inconnue et en déduire sa concentration.

4.5 Développement

On peut envisager des projets d'amélioration de certains éléments de notre petit appareil, notamment

EEPROM Sauvegarder les résultats de calibration (et certains autres réglages) dans la mémoire non-volatile de ARDUINO pour éviter leur perte. Au présent, l'appareil doit être recalibré chaque fois que il est rebranché (mis sous tension) !

Ecran Ajouter un écran (et quelques boutons de contrôle) pour afficher la concentration instantanée en toute autonomie.

GUI Développer une application sur PC pour communiquer avec l'appareil en utilisant Java ou Tcl/Tk.

Bluetooth Installer une carte («module») bluetooth et développer la même application pour ANDROID.

Température Ajouter une thermistance (sec. 3.1) pour contrôler la température des échantillons.

Spectre Étudier les possibilités d'utiliser des LED multicolores RGB et obtenir ainsi un «spectrophotomètre» à trois λ.

5 Projets UO ARDUINO 2021

Lors de la dernière séance TP Arduino, à la fin de la séance d'initiation, on se met en groupes de travail (de 2, 3, 4) et détermine un projet. On peut sélectionner un projet scientifique parmi ceux qui vous sont proposés, mais tout autre projet utilisant Arduino est possible. Il est donc indispensable d'étudier cette section, consulter de nombreux sites internet et toutes autres sources, échanger entre vous, avant la séance d'initiation (la première rencontre).

Vous devrez étudier votre projet d'un point de vue théorique et en discuter avec l'enseignant «mentor» pour commander éventuellement les éléments qui vous seront utiles pour son développement, idéalement à la fin de la 1ère séance.

Vous devrez réaliser le projet ou tout au moins aller le plus loin possible, en concertation avec l'enseignant responsable et présenter vos travaux à l'issue des séances de projets. Vous aurez la possibilité d'emporter vos montages pour travailler chez vous, en dehors de l'université. Nous vous rappelons que vous trouverez sur le réseau la majorité des informations qui vous seront nécessaires à la compréhension de votre projet. Vous aurez ensuite dans la majorité des cas à adapter un programme à votre cas particulier.

Parmi les projets que nous avons recherché (et donc nous pensons qu'ils seront à la fois faisables et intéressants dans le cadre de ce module) nous pouvons envisager les projets suivants.

5.1 Capteur de vitesse de vent par sonde Pitot (DS)

On propose de fabriquer un capteur autonome de vitesse relative (résistance) du vent sur la base d'Arduino et d'une sonde (tube) Pitot (Henri Pitot en 1732, voir le cours de hydrodynamique). Ce capteur peut être déployé dans une voiture (les voitures de F1 possèdent une sonde de ce type), drone, etc. L'objectif minimum sera d'avoir un prototype capable de mesurer la vitesse avec une précision et une stabilité raisonnables et les renvoyer par la connexion série/usb et/ou sur un écran lcd.

En option, on peut envisager une connexion via bluetooth vers des smartphones et une application pour piloter le capteur.

La sonde peut être trouvée d'occasion ou fabriquée au laboratoire. Pour les mesures, il s'agit de piloter plusieurs capteurs de pression (statique, dynamique) et de température, utiliser leur données pour calculer la vitesse, puis tester et calibrer l'appareil.

Pour les plus avancés, une correction pour la vitesse réelle mesurée par un GPS peut y être ajoutée.

5.2 Capteurs environnementaux

Dans différentes applications d'évaluation de la pollution, d d'installations chimiques (fuites), de combustion ou autres, on cherche à mesurer les concentrations des gaz (NO_2, CO, CO_2, O_3, SO_2 ...) ainsi que la pression, la température et les particules fines. On propose de développer un appareil portatif capable d'effectuer et d'enregistrer ces mesures (mode traceur) en temps réel et en autonomie pendant une certaine période du temps. Ce genre d'appareil peut être monté sur un drone ou un robot pour travailler dans les zones dangereuses, voir inaccessibles aux hommes. Ce projet est naturellement d'importance pour les sites de Calais et Dunkerque.

Ce projet ainsi que le projet Pitot fait partie du projet initial de BQE 2017. On envisage de piloter plusieurs capteurs avec un enregistrement des données sur une carte CF en temps réel, par exemple chaque seconde. En option, on peut envisager une communication wifi ou RF, ou même GSM.

Si la partie électronique n'est pas trop lourde, une calibration des capteurs peut être envisagée (en collaboration avec les chimistes ? sur Dk ? chambre des essays). On peut avoir plusieurs binômes faisant différents capteurs et autres parties de l'appareil, notamment un système d'enregistrement universel (avec aussi données GPS) à déployer ici et dans le capteur Pitot.

5.3 Viscosimètre avec un capteur inductif de déplacement/vitesse (DS)

Pour mesurer la viscosité d'un liquide (glycerol) on utilise une petite bille en acier du rayon r descendant le long de l'axe d'un cylindre en plexiglas de rayon $\rho \gg r$ et de longueur $l = 50$ cm rempli du liquide et placé verticalement. Ce viscosimètre fait parti des TP en L1 et L3. Le régime stationnaire (vitesse constante) s'installe après 10 premiers centimètres de descente. On détecte le passage de la bille à 10, 20, 30, et 40 cm ; le temps de passage entre les points du repère donne la vitesse stationnaire qui dépend de la viscosité du liquide (loi de Stokes). A noter que le même dispositif peut être testé (sans liquide) pour mesurer la constante gravitationnelle g.

La détection utilise le principe d'un détecteur de métal. On entoure le cylindre par un câble fin isolé en faisant 8 tours à chaque point de repère (donc $N = 32$ en tout) et on mesure le changement d'inductance L de l'ensemble provoqué par le passage de la bille. Pour $N = 32$ tours de câble et $\rho = 25$ mm (5m de câble) on obtient

$$L \approx 5\, N^2\, \rho = 128\, \mu\,\text{H}.$$

Pour mesurer L en temps réel on passe des impulsions de courte durée (quelques μsec). On peut utiliser Arduino lui même pour donner les impulsions, voir aussi d'autres projets, et les capteurs industriels. Dans le circuit, le câble fait partie d'un filtre LR, où on doit avoir au moins $R = 220\Omega$ pour protéger le circuit d'Arduino de surcharge. La résistance du câble étant de quelques Ω, donc bien inférieure à R. On peut ainsi anticiper le temps caractéristique du filtre $\tau = L/R = 0.58\mu$sec qui limite la durée utile des impulsions. On note que la fréquence du contrôleur ATmega326 est de 16MHz et son cycle de 0.06μsec est seulement 10 fois inférieur à τ. Ainsi pour former les impulsions de quelques μsec il est préférable d'utiliser l'accès direct aux registres au lieu de la procédure `digitalWrite` de la bibliothèque, trop gourmande en opérations. Le délai de lecture de son ADC avec `analogRead` est de 100μsec peut aussi être diminué. Autrement, pour le même $L = 100\mu$H il existe aussi un circuit intégré CS209A avec son propre oscillateur dont le fonctionnement reste à étudier.

5.4 Expérience du Rüchardt

Une bouteille de volume V_0 est munie d'un tube de petit diamètre interne d. Une balle en acier de la masse m et de même diamètre glisse facilement dans le tube en faisant des petits déplacements Δy tout en gardant la bouteille enfermée hermétiquement. Le volume V_0 est tel que ces déplacements ne l'affectent pratiquement pas. La pression p dans la bouteille dévie légèrement de la pression atmosphérique p_{atm}. Ce dispositif permet de mesurer la valeur de la constante adiabatique γ (coefficient de Laplace) du gaz dans la bouteille. Une bobine placée en haut du tube et alimentée par une source de la tension $U(t)$ sert à ajuster la position de la balle ou à forcer ses oscillations. Dans le but de développer un TP moderne pour les L3 et Master, on propose d'utiliser un microcontrôleur Arduino pour échantillonner en temps réel (sec. 2.1.7) les mesures simultanées de déplacement $\Delta y(t)$, de pression $p(t)$, de température du gaz $T(t)$, et de tension $U(t)$. En particulier, les déplacements sont mesurés avec un capteur laser 850nm *VL6180X/VL53L0X* (Time of Flight Micro-LIDAR Distance Sensor) capable d'un taux d'échantillonnage maximal de $\approx 200Hz$ (voir sec. 2.7.1 de sa notice). Pour une pression atmosphérique ($p_{\mathrm{atm}} \approx 100\,\mathrm{kPa}$), une masse $m = 10$ de 20 g, $V_0 = 10\,\mathrm{L}$, $d = 10$ à 20 mm, $\gamma = 7/5$ (gaz diatomique) à la température 25°C, les fréquences propres sont de l'ordre de 1 à 10 Hz.

5.5 Magnétomètre, pendule, et scanner 3D

On propose d'utiliser Triple Axis Magnetometer Breakout - HMC5883L (le circuit intégré HMC5883L de Honeywell + interface I2C) pour mesurer le champ magnétique en temps réel. Par la suite, on peut utiliser une imprimante 3D pour scanner le champ dans un volume, ou avoir un système de moteurs. Applications : TP physique, entre autres. Une autre application fortement intéressante est de construire un pendule afin de observer et mesurer la vitesse de rotation de la Terre [6].

5.6 Accéléromètre dans un indicateur de freinage style F1

On se propose d'utiliser un accéléromètre + gyroscope (avec, en option, un GPS) pour détecter, enregistrer et réagir à toutes accélérations du véhicule. Ainsi un clignotant rouge feux arrière peut être activé quand le véhicule décélère au frein moteur. Une autre application est un indicateur automatique des virages dans une moto ou le pilote n'a pas le temps d'activer/désactiver le clignotant.

5.7 PCR et qPCR thermal cycler (DS)

De nombreux tests modernes de diagnostic moléculaire ciblant les acides nucléiques sont généralement limités aux pays développés ou aux laboratoires de référence nationaux des pays en développement. La capacité de rendre les technologies de diagnostic rapide des maladies infectieuses dans un format portable et peu coûteux constituerait une avancée révolutionnaire dans le domaine de la santé mondiale. De nombreux tests moléculaires moléculaires sont également élaborés sur la base de réactions en chaîne par polymérase (PCR), qui nécessitent des thermocycleurs relativement lourds (de l'ordre de 10 à 20 kg) et nécessitant une une alimentation électrique continue. La vitesse de montée en température de la plupart des thermocycleurs les plus économiques est relativement lente (2 à 3 °C/s), de sorte qu'une réaction en chaîne par polymérase peut prendre 1 à 2 heures. Par-dessus tout, ces thermocycleurs sont encore trop chers (de 2 000 à 4 000 dollars) pour être utilisés dans des environnements à faibles ressources.

Les projets OpenPCR et PS-PCR sont déjà très développés et testés, il y a une doc très détaillée pour les reproduire et tout le logiciel nécessaire (y comprit côté PC où on les branche via port usb). Comme cerise sur le gâteau, on pourrait peut être y ajouter un pilote par iPhone... évolution vers la détection en temps réel avec fluorescence voir Open qPCR alternative robotique à l'élément Peltier, voir rapid and low-cost PCR thermal cycler for low resource settings

En bref : il s'agit de réplication de gènes (DNA). À la base, c'est un bon élément Peltier piloté par Arduino. On mesure/contrôle aussi la température et (en option) la fluorescence. Il y a une grande partie biologique et un peu de cinétique chimique. Les tests sont à faire en collaboration avec les biologistes (Jean-Christophe Devedjan)

5.8 Capteur déplacement/vitesse (DS)

On récupère une vieille souris pour connecter ces capteurs au microcontrôleur Arduino et en faire un capteur de déplacement linéaire (1D et/ou 2D), angulaire (rotation), vitesse, et accélération. Il y a de nombreuses applications dans les labos de TP, notamment pendule de Pohl (TP L1 de mécanique), goniomètre (optique), mesures en 2D, robotique. Dans le cas du pendule de Pohl, l'idée est de reproduire le «picket fence» photogate système déjà utilisé pour les déplacements linéaires et les oscillations (GTI+RÉGRESSI) pour mesurer les oscillations du pendule de Pohl avec *deux* enregistrements simultanés : l'un pour les mouvements du pendule et l'autre pour les mouvements de la poussée produite par le moteur (force externe). Voir le prototype et la théorie.

5.9 Spectrophotomètre (DS)

Connecter Arduino à l'ancien spectrophotomètre pour faire les mesures automatisées.

6. D. B. Plewes, Magnetic monitoring of a small Foucault pendulum, *Rev. Sci. Instr.*, **89**(6):065112 (2018)

5.10 Anémomètre et girouette à effet Doppler (R. Bocquet)

Le but du projet est de réaliser un anémomètre à effet Doppler. Il s'agit d'utiliser cet effet bien connu, dans la gamme des ultrasons, pour mesurer la vitesse du vent ainsi que sa direction. Cette gamme de sons se situe à des fréquences au-dessus des 20 kHz, limite audible de tout être humain normalement constitué. Vous aurez à votre disposition des émetteurs et des récepteurs accordés certainement à 30 kHz mais qu'il faudra vérifier. L'idée de la mesure est assez simple : l'onde sonore est une onde mécanique portée par l'air, la vitesse de propagation dépendra donc de la vitesse de l'air. Vous devrez :

- comprendre le principe physique de l'effet Doppler dans l'air
- mettre en évidence cet effet dans une expérience
- proposer un montage permettant de donner également la direction
- proposer un montage (électronique + ARDUINO) pour mesurer des vents allant de 0,1 noeud à 50 noeuds [7]
- réaliser le montage si vous avez le temps et le tester.

Ne négligez pas le travail préparatoire dans ce projet.

5.11 Cinémomètre à GPS (R. Bocquet)

Le but du projet est de réaliser un instrument de la taille d'une grosse montre pour mesurer la vitesse et la direction de déplacement d'un bateau à voile ou d'une bicyclette. Pour cela on vous propose d'utiliser un composant (u-blox 6 GLONASS GPS) qui n'est autre qu'un GPS et un tout petit écran de visualisation de 4 caractères en bus I2C. On utilisera le format de données NMEA, très utilisé dans l'industrie du nautisme et disponible sur le GPS que vous avez à votre disposition. Vous devrez :

- réaliser le montage GPS et visualisation avec carte ARDUINO
- réaliser le montage GPS et visualisation sur écran
- réaliser le montage avec un micro-contrôleur Atiny 45 ou 85
  * développer une carte électronique autonome (pile 3,3 V bouton) et son boitier

5.12 Optimisation de la consommation d'un micro-contrôleur ATtiny 85 (R. Bocquet)

Les microcontrôleurs que vous utilisez ont la possibilité de fonctionner avec de très faibles consommations et sous des tensions de 3,3 V. Il s'agit dans ce projet de mettre en place une programmation d'un micro-contrôleur ATtiny 85 qui est un composant électronique enfichable sur la plaquette d'essais et de faire un montage permettant de mesurer la consommation du composant. Ce projet demande des notions d'électronique et de très bonnes connaissances de la langue anglaise. En effet vous devrez lire la notice du micro-contrôleur et programmer directement les ports du micro-contrôleur.

5.13 Centrale météorologique – 2 groupes de travail (R. Bocquet)

Il s'agit de mettre en place une centrale de mesure météorologique avec un boîtier extérieur embarquant les capteurs et un boîtier intérieur pour réception, traitement et stockage des données. Les deux boîtiers seront reliés par une transmission radiofréquence à 433 MHz. Dans le cas où un seul groupe est constitué, chaque partie peut -être traitée séparément.

Groupe 1 : capteurs et émetteur. réaliser et mettre au point un montage permettant à minima de mesurer la température, la pression, l'humidité relative et la luminosité. Vous définirez un protocole de données et mettrez en place une liaison RF pour transmettre les données. Vous étudierez les possibilités de transfert sur carte de votre montage en utilisant un micro-contrôleur ATtiny 45 ou 85.

Groupe 2 : récepteur. réaliser et mettre au point un récepteur des données météo du groupe 1 avec un moyen de sauvegarde. Vous programmerez une carte ARDUINO pour visualiser sur un écran LCD graphique les données et développerez un code pour réaliser une prévision météo, fonction des données enregistrées.

Groupes 1 et 2 : liaison RF. établir la liaison RF entre les 2 boîtiers et tester les distances possibles de transmission, ainsi que les difficultés qui pourraient nuire à la qualité de la transmission.

7. 1 noeud = 1 mile nautique par heure