

Microcontrôleurs pour les sciences physiques

David THERINCOURT
Lycée Roland Garros - Académie de la Réunion
6 juin 2025

Table des matières

Table des matières	i
1 Introduction	3
1.1 Qu'est-ce qu'un microcontrôleur?	3
1.2 Pourquoi des microcontrôleurs en sciences physiques ?	5
1.3 Les fonctions d'un microcontrôleur	5
1.4 Applications en sciences physiques	5
2 Cartes Arduino	7
2.1 Qu'est-ce qu'Arduino ?	7
2.2 La carte Arduino UNO (Rev 3)	8
2.3 Educaduino Lab (Eurosmart)	9
2.4 Plug'Uino® Uno (Sciencéthic)	11
3 Langage Arduino	13
3.1 Logiciel Arduino IDE	13
3.1.1 Arduino IDE (version 1.8)	13
3.1.2 Arduino IDE (version 2.x)	13
3.1.3 Etapes de la mise en oeuvre d'un projet Arduino	14
3.2 Premier programme : Blink	14
3.2.1 Edition	14
3.2.2 Compilation	16
3.2.3 Téléversement	17
3.2.4 Exécution	18
3.3 Spécificités du langage Arduino	19
3.3.1 Syntaxe	20
3.3.2 Constantes prédéfinies	20
3.3.3 Structure du programme	20
3.3.4 Documentation	20
4 Les bases d'Arduino	21
4.1 Allumer une LED (sorties numériques)	21
4.1.1 Principe	21
4.1.2 Montage	21
4.1.3 Programme en langage Arduino (C/C++)	22
4.1.4 Applications	22
4.2 Modifier l'intensité lumineuse d'une LED (sorties PWM)	23
4.2.1 Principe	23
4.2.2 Montage	24
4.2.3 Programme en langage Arduino (C/C++)	24
4.3 Afficher des messages (port série UART)	25
4.3.1 Principe	25
4.3.2 Moniteur série	25
4.3.3 Programme de test	25
4.3.4 Applications	26
4.4 Mesurer une tension (CAN)	26
4.4.1 Principe	26

4.4.2	Montage	27
4.4.3	Programme Arduino	28
4.4.4	Applications	28
5	Nouveaux programmes du lycée	31
5.1	Capteur résistif - CTN (seconde générale)	31
5.1.1	Principe	31
5.1.2	Montages pour la mesure d'une résistance	31
5.1.3	Cas d'un capteur résistif CTN	34
5.1.4	Programme 1 : mesure de la résistance de la CTN	36
5.1.5	Programme 2 : thermomètre numérique (application)	38
5.1.6	A retenir	40
5.2	Émission d'un son (seconde générale)	40
5.2.1	Principe	40
5.2.2	Méthode 1 : construire le signal carré	40
5.2.3	Méthode 2 : utiliser une fonction spéciale	41
5.2.4	Applications	41
5.3	Mesurer la célérité d'un son (première générale)	41
5.3.1	Présentation du module HC-SR04	42
5.3.2	Mesure de la célérité du son	45
5.3.3	Application : réalisation d'un télémètre	46
5.3.4	A retenir	48
5.4	Mesurer une pression - Loi de Mariotte (première générale)	48
5.4.1	Principe	48
5.4.2	Capteur de pression absolue MPX5700AP	48
5.4.3	Calcul de la pression	50
5.4.4	Programme Arduino	50
5.4.5	Résultats	51
5.5	Mesurer une pression - Loi de la statique des fluides (première générale)	53
5.5.1	Principe	53
5.5.2	Capteur Plug'Uino (XGZP6857A)	53
5.5.3	Montage	55
5.5.4	Programme Arduino	55
5.6	Géométrie d'un condensateur (terminale générale)	56
5.6.1	Principe	57
5.6.2	Montage	57
5.6.3	Programme	57
5.6.4	A retenir	57
5.7	Capteur capacitif (terminale générale)	57
5.7.1	Principe	57
5.7.2	Montage	59
5.7.3	Programme Arduino	59
5.7.4	Application : mesure d'une capacité	60
5.7.5	A retenir	61
6	Aller plus loin	63
6.1	Contrôler l'intensité d'une LED avec un potentiomètre	63
6.1.1	Montage	63
6.1.2	Programme	63
6.2	Arduino IDE - Saisir une valeur au clavier	64
6.2.1	Montage	64
6.2.2	Programme	64
6.3	Grove - Afficheur LCD 16x2 sur port I2C	65
6.3.1	Principe	65
6.3.2	Grove LCD RGB Backlight	66
6.3.3	Grove LCD 16x2 (rouge/jaune/bleu)	67
6.4	Eduarduino Lab - Afficheur LCD	67
6.4.1	Principe	67

6.4.2	Programme	68
6.5	Educaduino Lab - Loi de Mariotte	69
6.5.1	Principe	69
6.5.2	Capteur MPXHZ6400A	69
6.5.3	Calcul de la pression	70
6.5.4	Montage	71
6.5.5	Programme Arduino	71
6.5.6	Résultats	72
6.6	Educaduino Lab - Loi de la statique des fluides	72
6.6.1	Principe	72
6.6.2	Capteur MPX2010DP/GP	72
6.6.3	Capteur Educaduino Lab (MPX2010GP)	74
6.6.4	Montage	75
6.6.5	Programme Arduino	75
7	Cartes Micro :bit (langage Micropython)	77
7.1	Cartes Micro :bit	77
7.2	Langage BBC MicroPython	79
7.2.1	Edition	79
7.2.2	Librairie	80
7.2.3	Écrire sur une sortie digitale	80
7.2.4	Lire une entrée digitale	80
7.2.5	Générer une tension MLI (PWM)	81
7.2.6	Mesurer une tension (CAN)	81
7.2.7	Générer une tension analogique (CNA)	81
7.2.8	Faire une pause	81
7.2.9	Mesurer une durée	81
7.3	Les bases de la carte Micro :bit	82
7.4	Programmes du lycée	82
7.4.1	Capteur résistif - CTN (seconde générale)	82
7.4.2	Application : thermomètre numérique	83
7.4.3	Mesurer une pression - Loi de Mariotte (première générale)	84
8	Cartes Pyboard (langage Micropython)	87
8.1	Cartes Pyboard	87
8.1.1	La carte officielle Pyboard	87
8.1.2	La carte Plug'Uino PY de Scientechnic	88
8.1.3	La carte Feather STM32F405 Express d'Adafruit	90
8.2	Langage Micropython	91
8.2.1	Edition	91
8.2.2	Librairie	91
8.2.3	Écrire sur une sortie digitale	92
8.2.4	Lire une entrée digitale	92
8.2.5	Mesurer une tension (CAN)	92
8.2.6	Générer une tension analogique (CNA)	93
8.2.7	Générer une tension MLI (PWM)	93
8.2.8	Faire une pause	93
8.2.9	Mesurer une durée	93
8.3	Les bases	94
8.4	Programme du lycée	94
8.4.1	Capteur résistif - CTN (seconde générale)	94
8.4.2	Application : thermomètre numérique	95
8.4.3	Mesurer une pression - Loi de Mariotte (première générale)	96
9	Cartes Arduino (librairie Numpy de Python)	99
9.1	Langage Python avec Numpy	99
9.1.1	Qu'est-ce que Numpy ?	99
9.1.2	Principe de fonctionnement	99
9.1.3	Téléversement du firmware Numpy sur la carte Arduino	100

9.1.4	Installation la librairie Numpy sur l'ordinateur	101
9.1.5	Exemple : le programme Blink	102
9.2	Les bases de Numpy	102
9.2.1	Allumer une LED (sorties numériques)	102
9.2.2	Modifier l'intensité lumineuse d'une LED (sorties PWM)	103
9.2.3	Afficher des messages dans la console Python	103
9.2.4	Mesurer une tension (CAN)	104
9.3	Programmes du lycée	104
9.3.1	Capteur résistif - CTN (seconde générale)	105
9.3.2	Application : thermomètre numérique	106
9.3.3	Émission d'un son (seconde générale)	107
9.3.4	Mesurer la célérité d'un son (première générale)	108
9.3.5	Application : télémètre (première générale)	109
9.3.6	Mesurer une pression - Loi de Mariotte (première générale)	110

Les nouveaux programmes 2019 en seconde générale et technologique, en première générale et en terminale générale introduisent l'utilisation des **microcontrôleurs** et des **capteurs** en sciences physiques.

L'objectif de ce document est d'apporter aux professeurs de physique-chimie les notions techniques nécessaires sur les microcontrôleurs afin d'aborder au mieux les nouvelles **capacités numériques**.

Cette documentation est principalement construite autour de l'**Arduino UNO** programmé en **langage Arduino (C/C++)**.

A la fin du document, des solutions sont proposées pour programmer en **langage Python** des microcontrôleurs (Micro :bit, PyBoard et Arduino) avec le **langage Micropython** ou la **librairie Numpy**.

Chapitre 1

Introduction

1.1 Qu'est-ce qu'un microcontrôleur ?

Un microcontrôleur est un circuit intégré regroupant un micro-processeur, de la mémoire et des périphériques sur la même puce. Contrairement à un microprocesseur classique, un **microcontrôleur est surtout utilisé pour une application électronique spécifique**.

De nos jours, les microcontrôleurs sont présents un peu partout : dans les appareils domestiques, médicaux, de télécommunication, dans les voitures, les avions, l'industrie, ...

Apparus dans les années 70, les microcontrôleurs à architecture 8 bits ne sont pas près de disparaître. Très peu chère, on les retrouve dans des petites applications (ex. télécommande).

La célèbre carte **Arduino UNO R3** est construite autour d'un **microcontrôleur 8 bits** !

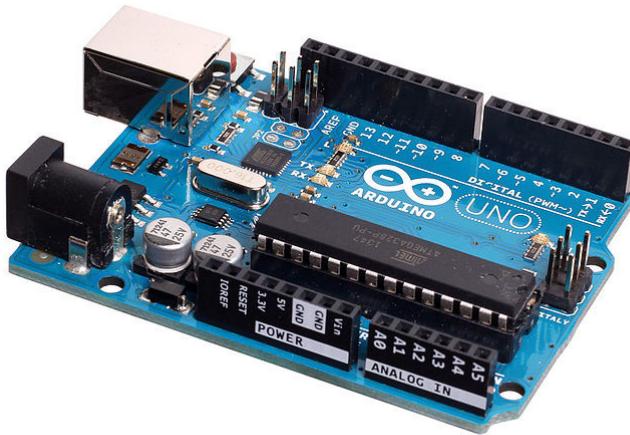


FIG. 1 – Carte Arduino UNO (microcontrôleur Atmel ATMEGA 328)

Actuellement, la tendance est aux **microcontrôleurs 32 bits** (ex. ARM Cortex-M, STM32, ...) qui sont plus adaptés aux applications plus évoluées. C'est ce type de microcontrôleur qui a permis le portage du langage Python (Micropython) au sein des microcontrôleurs. Les cartes Micro :bit, Pyboard ou encore à base d'ESP32 en sont les parfaits exemples !



FIG. 2 – Carte micro :bit



FIG. 3 – Carte PyBoard (microcontrôleur STM32)

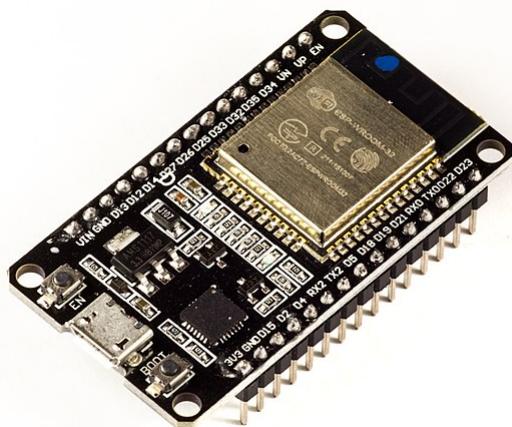


FIG. 4 – Carte ESP-WROOM-32 (microcontrôleur ESP32)

1.2 Pourquoi des microcontrôleurs en sciences physiques ?

Le monde actuel est fortement imprégné par le numérique. Par exemple, les téléphones portables et les objets connectés comportent une **multitude de capteurs** mesurant des grandeurs très variées comme la température, la fréquence cardiaque, la pression, l'accélération, les ondes sonores,

Il est donc important d'expliquer comment s'effectue la **mesure d'une grandeur physique analogique** dans ces appareils numériques du quotidien.

Il en est de même pour la **génération de signaux** (ex. son).

1.3 Les fonctions d'un microcontrôleur

Les microcontrôleurs permettent principalement de :

- **générer de signaux** (ex. son, impulsion de commande, ...).
- **mesurer des tensions** (ex. adaptation de capteurs analogiques, acquisition de signaux, ...).
- **mesurer des durées** (ex. période, fréquence, temps caractéristique, ...).

1.4 Applications en sciences physiques

De manière générale, les microcontrôleurs sont utilisés :

- pour réaliser des **petites applications** (ex. thermomètre, télémètre à ultrasons, ...) en rapport avec un cours ou un TP ;
- dans des **projets** (enseignement scientifique).

Avec des capteurs, il est en plus possible de :

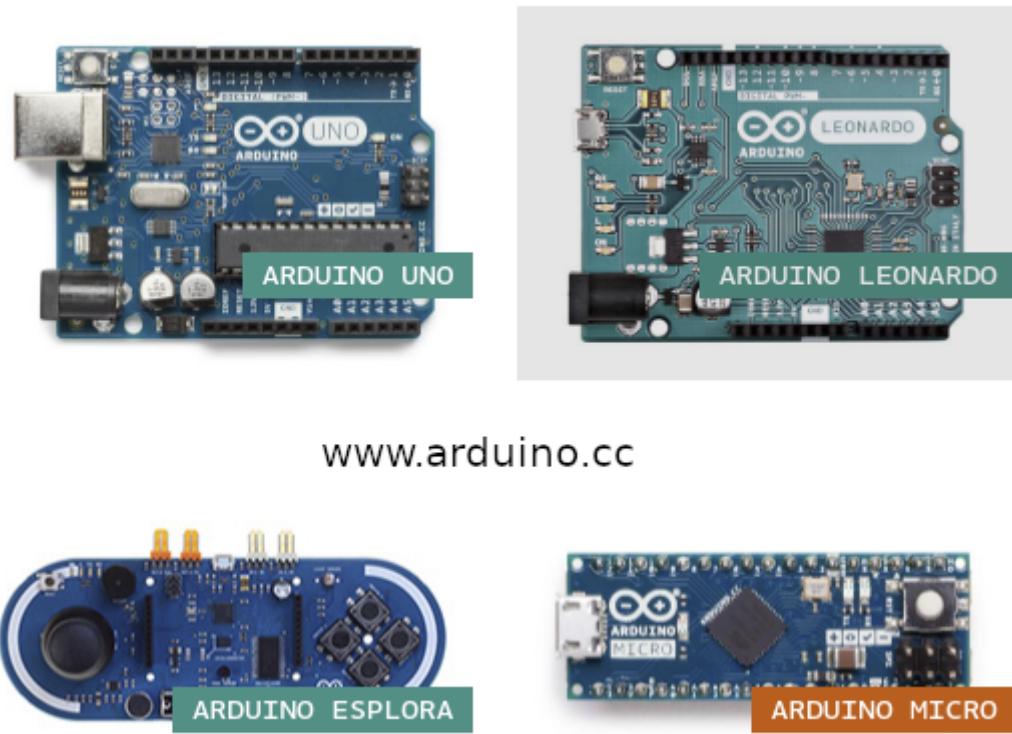
- réaliser des **mesures** (ex. température, célérité son, pression, ...);
- faire de l'**acquisition de données** en mode **autonome** (ex. mesure de pression sur un ballon sonde) ou mode **connecté** (branché à un ordinateur).

Chapitre 2

Cartes Arduino

2.1 Qu'est-ce qu'Arduino ?

Arduino est une **carte électronique** à base de microcontrôleur (ex. ATMEL AVR ATMEGA) développée par Arduino.cc sous licence libre. Tous les schémas des cartes sont disponibles librement sur le Web. A peu près une vingtaine de versions de cartes officielles ont été fabriquées dont la célèbre l'Arduino UNO.



www.arduino.cc

A l'origine conçue pour la **création artistique**, la carte Arduino trouve des applications dans des domaines d'applications aussi variés qu'insolites. Une carte Arduino s'utilise généralement comme :

- **dispositif autonome** dans des applications comme la domotique, la robotique, les systèmes embarqués, ...
- **interface** entre un ordinateur (logiciel tiers) et des capteurs ou des actionneurs ;

Arduino IDE est le **logiciel de développement** des cartes du même nom. Également sous licence libre, cet environnement de développement intégré (IDE) utilise C/C++ comme langage de programmation et le port USB pour le téléchargement du programme.

2.2 La carte Arduino UNO (Rev 3)

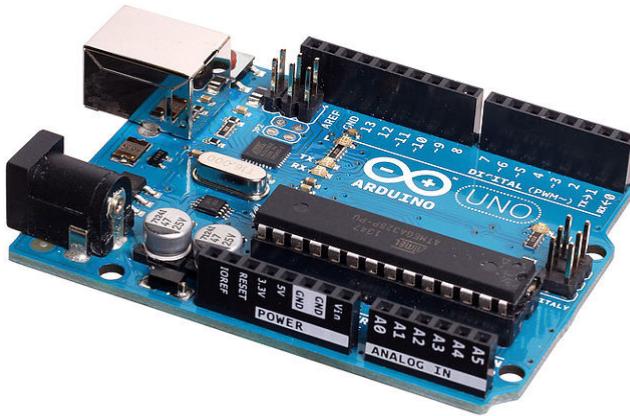


FIG. 2 – Arduino Uno R3

Arduino UNO est une des cartes officielles les plus récentes et économiques.

Caractéristiques principales :

- microcontrôleur 8 bits ATMEGA328P cadencé à 16 Mhz ;
- alimentation externe (7 à 12 V) ou USB (5 V) ;
- programmation et communication via port USB ;
- 14 broches d'entrées/sorties numériques dont 6 PWM ;
- 6 entrées analogiques sur 10 bits ;
- 1 port I2C (communication avec capteurs/actionneurs numériques) ;
- 1 port UART (communication série) ;
- 3 timers (comptage et mesure de temps) ;
- gestion des interruptions.

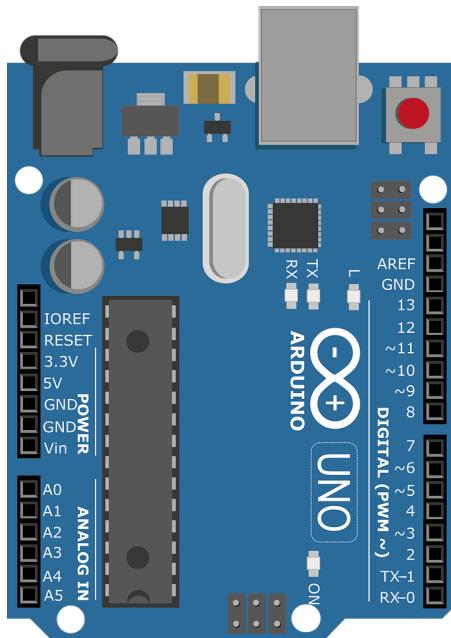


FIG. 3 – Brochage de l'Arduino Uno R3

⚠ Avertissement

Les niveaux de tension acceptables sur les broches d'entrées doivent être **comprises entre 0 V et 5 V** sous peine de détruire le microcontrôleur ou la carte !

i Note

La carte Arduino UNO ne possède pas de vraies sorties analogiques mais des sorties à **modulation de largeur d'impulsion** (MLI). Ce sont les six fameuses **sorties PWM** (Pulse Width Modulation).

2.3 Educaduino Lab (Eurosmart)

<https://educaduino-lab.com/>

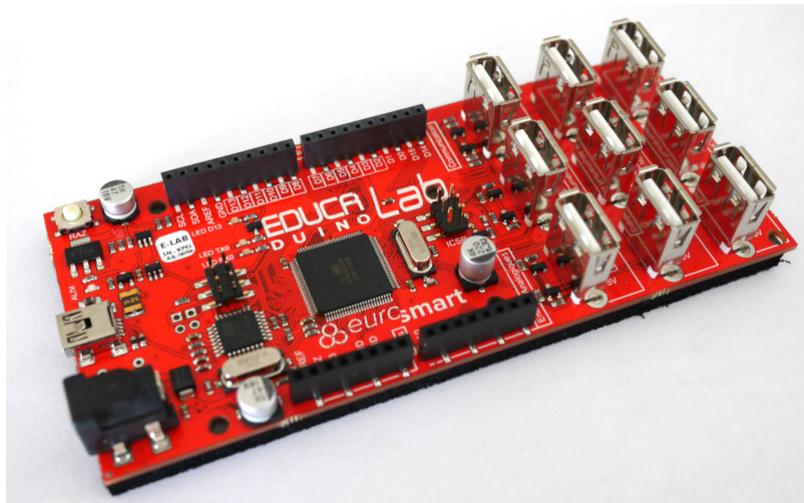


FIG. 4 – La carte Educaduino-Lab (E-LAB)

La carte **Educaduino Lab** a été conçue sur la base d'une carte Arduino MEGA 2560. Cette dernière est équivalente à une carte arduino UNO mais avec plus de mémoire et surtout **plus de ports d'entrée/sortie**. Ce qui a permis à Eurosmart d'y placer des **connecteurs USB pour ses propres capteurs** tout en gardant la connectique classique de l'Arduino UNO.

Caractéristiques principales :

- microcontrôleur ATMEGA 2560 (comme l'Arduino MEGA 2560) ;
- protection des ports d'entrée/sortie ;
- brochage compatible Arduino Uno Rev 3 (pin 0.8mm, shield Grove, ...) ;
- ports supplémentaires en USB pour capteurs Educaduino-Lab ;

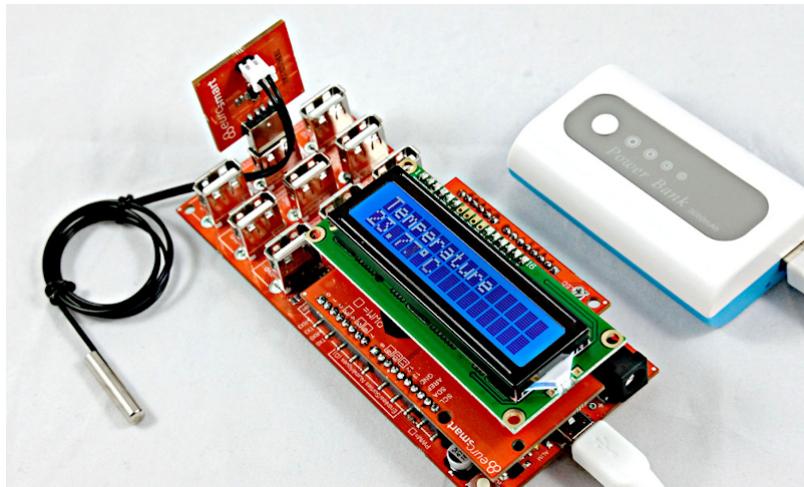


FIG. 5 – Mesure d'une température (image : Eurosmart)

Une malette avec un afficheur LCD et plusieurs capteurs adaptés au programme du lycée est également proposée.



FIG. 6 – Kit sciences-physiques 2nde/1ère (image : Eurosmart)

2.4 Plug'Uino® Uno (Sciencéthic)

<https://www.scencethic.com/>

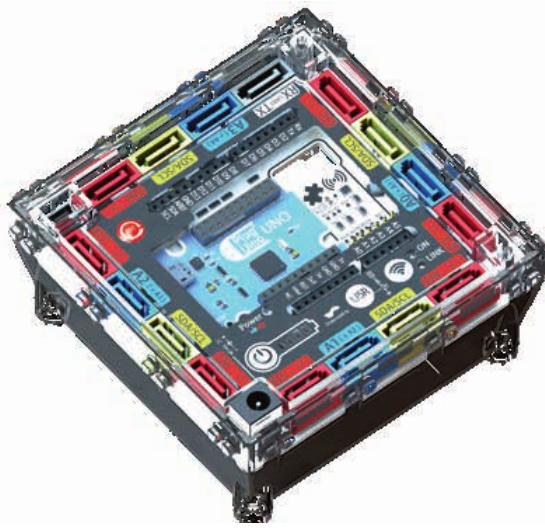


FIG. 7 – La carte Plug'Uino ® Uno (image : Sciencéthic)

Sciencéthic propose également une carte **Plug'Uino Uno** protégée contre les mauvaises manipulations et 100% compatible Arduino UNO Rev 3.

Caractéristiques principales :

- microcontrôleur ATMEGA 328P (comme l'Arduino Uno) ;

- protection des ports d'entrée/sortie ;
- brochage compatible Arduino Uno Rev 3 (pin 0.8mm, shield Grove, ...) ;
- connecteurs SATA pour les capteurs Plug'uno ;

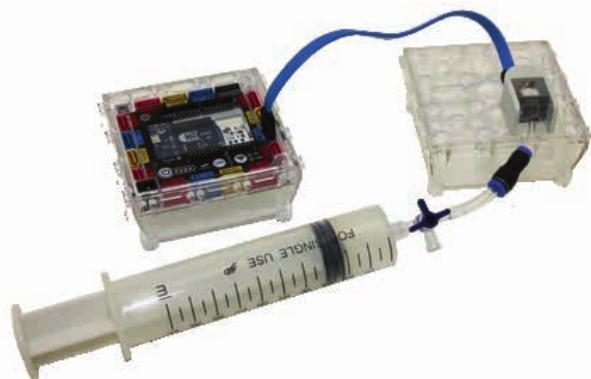


FIG. 8 – Capteur de pression et loi de Mariotte (image : Sciencéthic)

Chapitre 3

Langage Arduino

3.1 Logiciel Arduino IDE

3.1.1 Arduino IDE (version 1.8)

Le logiciel **Arduino IDE 1.1** est un environnement intégré de développement (IDE) multiplateforme. Il est téléchargeable sur le site officiel <http://www.arduino.cc/en/>

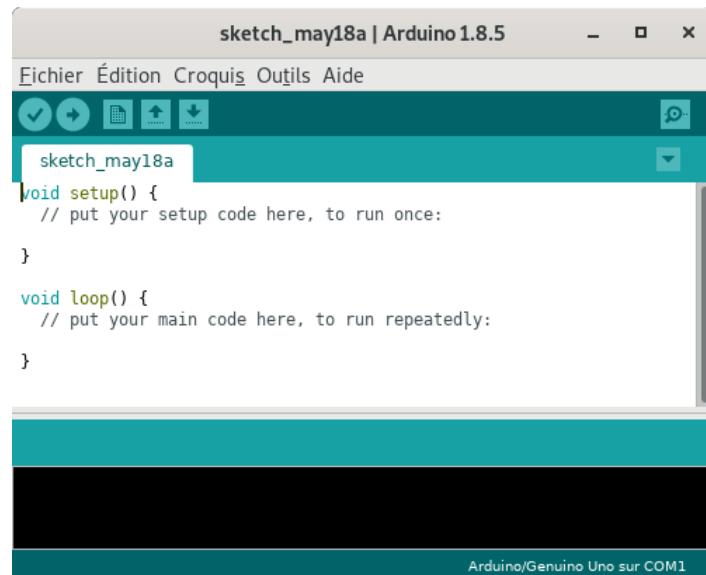


FIG. 1 – Logiciel Arduino IDE 1.8

3.1.2 Arduino IDE (version 2.x)

Le logiciel Arduino IDE 2 est la nouvelle version de l'environnement intégré de développement (IDE) d'Arduino.



FIG. 2 – Logiciel Arduino IDE 2

3.1.3 Etapes de la mise en oeuvre d'un projet Arduino

La mise en œuvre d'un projet Arduino s'effectue dans l'ordre suivant :

1. **Édition** du code source du programme dans l'éditeur de l'interface ;
2. **Compilation** du code source du programme (vérification du code et génération du programme en langage machine) ;
3. **Téléversement** du programme sur la carte Arduino ;
4. **Exécution** du programme sur le carte Arduino (de façon autonome sans ordinateur).

3.2 Premier programme : Blink

Le programme **Blink** a pour objectif de faire clignoter la LED intégrée à la carte de développement. Cette LED est connectée en interne à la broche 13.

3.2.1 Edition

Le programme **Blink** est disponible dans les exemples du logiciel **Arduino IDE**.

Dans le menu **Fichier > Exemples > Basics > Blink**.

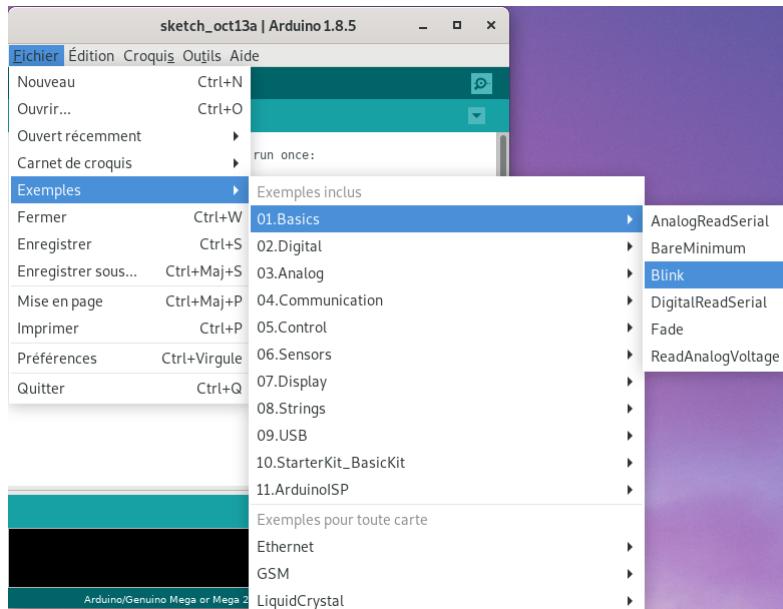


FIG. 3 – Ouvrir le code source du programme Blink

```

Fichier Édition Croquis Outils Aide
Blink §

/*
  Blink
  ...
*/

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

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}

3 Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) sur /dev/ttyACM0

```

The screenshot shows the Arduino IDE with the title bar "sketch_oct13a | Arduino 1.8.5". The code editor window contains the "Blink" sketch. The code is as follows:

```

/*
  Blink
  ...
*/

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

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}

```

The status bar at the bottom indicates "3" and "Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) sur /dev/ttyACM0".

FIG. 4 – Édition du code source du programme Blink

i Note

- Un programme Arduino écrit en **langage C/C++** est composé d'**une suite d'instructions**.
- Ces instructions sont exécutées dans l'**ordre des lignes de code**.
- Les **commentaires** en gris sont délimités par les caractères /* et */ sur plusieurs lignes ou commencent pas les caractères // sur une même ligne.



The screenshot shows the Arduino IDE interface with the following code in the editor:

```
Éichier Édition Croquis Outils Aide
Fichier Édition Croquis Outils Aide
blink2 §
/*
Blink
*/
// Déclarations
int LED = 13;

// Configuration
void setup() {
    // initialize digital pin LED_BUILTIN as an output.
    pinMode(LED, OUTPUT);
}

// Boucle sans fin
void loop() {
    digitalWrite(LED, HIGH);    // turn the LED on (HIGH is the voltage level)
    delay(1000);               // wait for a second
    digitalWrite(LED, LOW);     // turn the LED off by making the voltage LOW
    delay(1000);               // wait for a second
}
```

The status bar at the bottom indicates: 14, Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) sur /dev/ttyACM0.

FIG. 5 – Une version modifiée du code source du programme Blink

Avertissement

Un programme Arduino respecte toujours une **structure spécifique** composée en trois parties :

- Les déclarations : **définitions** des constantes et des variables ;
- La fonction **setup()** : **configuration** de la carte (entrées, sorties, port série, ...);
- La fonction **loop()** : **instructions du programme exécutées dans une boucle infinie** (sans fin).

3.2.2 Compilation

Avertissement

Avant de lancer la compilation, il est important de **choisir le modèle de carte Arduino utilisé**. Le programme généré est dépendant du type de microcontrôleur présent sur la carte.

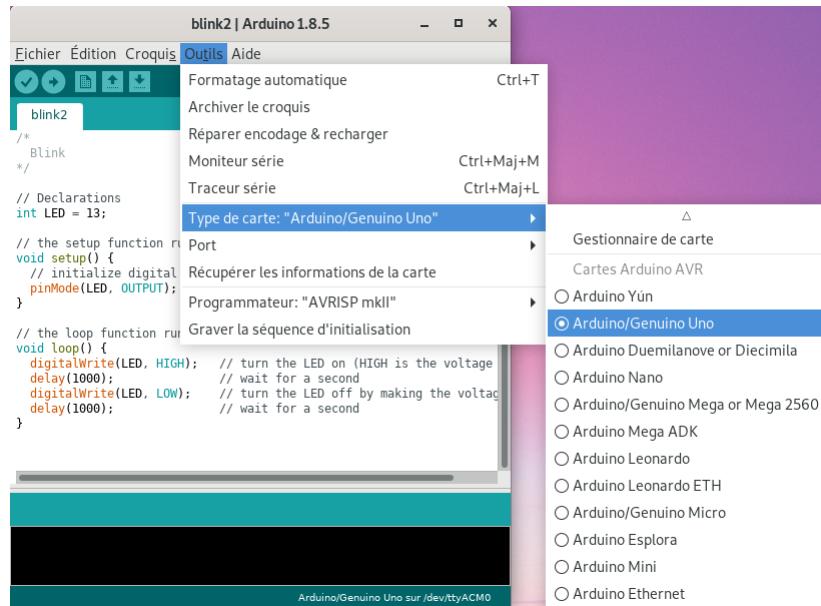


FIG. 6 – Choix du type de carte

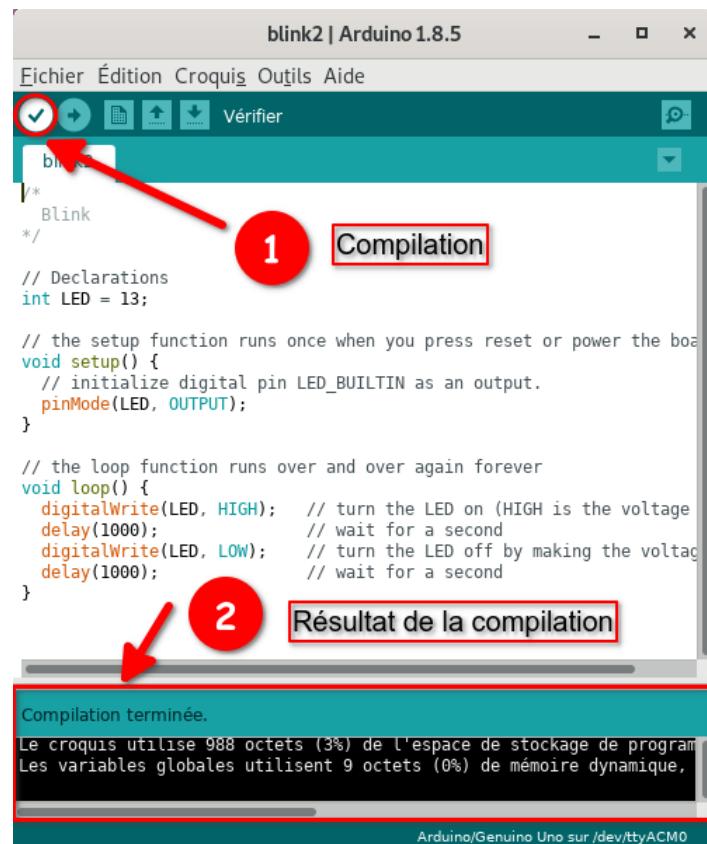


FIG. 7 – Puis la compilation peut s'effectuée !

3.2.3 Téléversement

Avertissement

Pour téléverser le programme obtenu, il faut **sélectionner le port de communication série** sur lequel est connectée la carte Arduino.

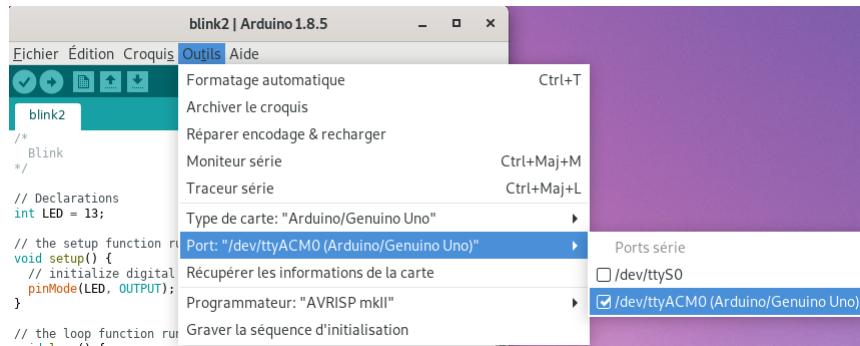


FIG. 8 – Choix du port de communication

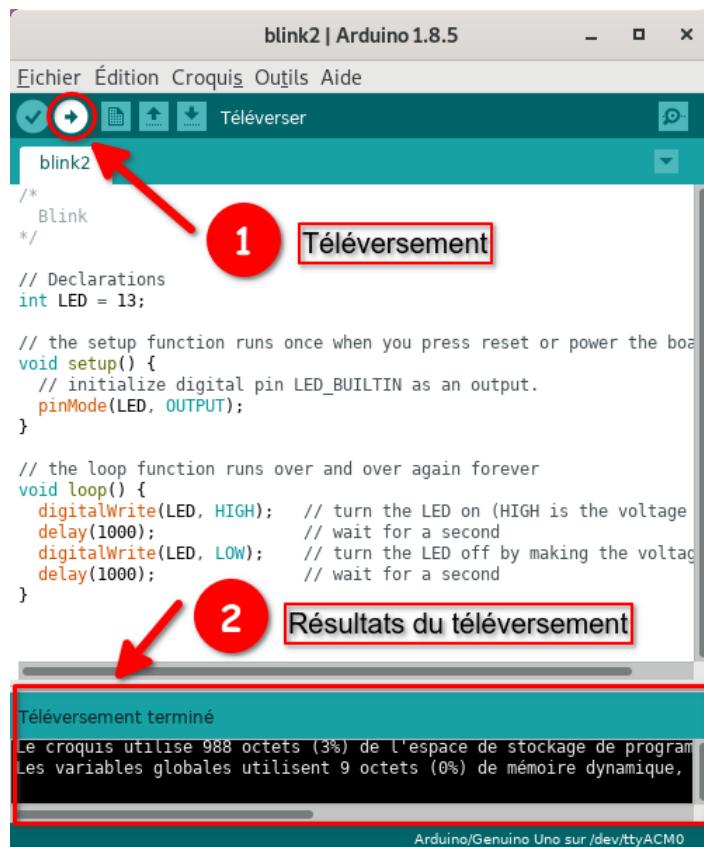


FIG. 9 – Téléversement du programme

3.2.4 Exécution

Le programme s'exécute sur la carte Arduino de façon autonome (sans ordinateur).

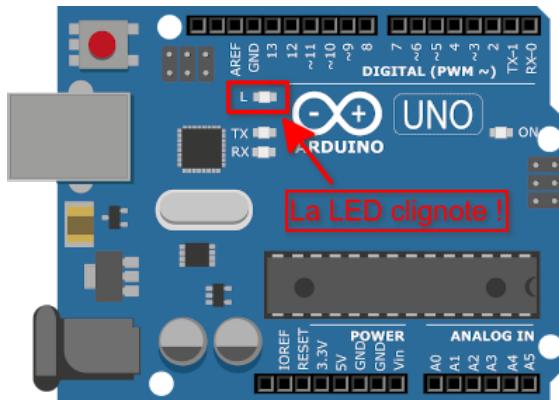


FIG. 10 – Exécution du programme Blink sur la carte Arduino Uno R3

3.3 Spécificités du langage Arduino

Le langage de programmation C/C++ est utilisé par le logiciel Arduino IDE pour programmer les microcontrôleurs Arduino.

The screenshot shows the Arduino IDE interface with the following details:

- Menu Bar:** Fichier, Édition, Croquis, Outils, Aide.
- Title Bar:** blink2 §
- Code Area:**

```
/*
  Blink
*/

// Declarations
int LED = 13;

// Configuration
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED, OUTPUT);
}

// Boucle sans fin
void loop() {
  digitalWrite(LED, HIGH);      // turn the LED on (HIGH is the voltage level)
  delay(1000);                 // wait for a second
  digitalWrite(LED, LOW);       // turn the LED off by making the voltage LOW
  delay(1000);                 // wait for a second
}
```
- Bottom Status Bar:** 14, Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) sur /dev/ttyACM0

FIG. 11 – Edition d'un code source d'un programme Arduino

3.3.1 Syntaxe

- Toutes les instructions se terminent par un point virgule ; sauf pour les directives `#include` et `#define`.
- Les **blocs d'instructions** sont délimités par des accolades `{...}`.
- Les **commentaires** en gris sont délimités par les caractères `/*` et `*/` sur plusieurs lignes ou commencent pas les caractères `//` sur une même ligne.

3.3.1.1 Typage des variables

Le type d'une variable doit être renseigné à sa déclaration.

Quelques types disponibles :

Type	Description	Valeurs
<code>int</code>	entier sur 16 bits	-32768 à 32767
<code>long</code>	entier sur 32 bits	-2147483648 à 2147483647
<code>float</code>	flottant sur 32 bits	-3.4028235E+38 à -3.4028235E+38 ;
<code>char</code>	caractère sur 8 bits	Table ASCII

Exemples :

```
int a = 5;
float pi = 3.14;
char c = 'A';
```

3.3.2 Constantes prédéfinies

Afin d'améliorer la lecture du code, des constantes sont définies.

Constante	Valeur
<code>LOW</code>	0 (niveau logique)
<code>HIGH</code>	1 (niveau logique)
<code>OUTPUT</code>	broche en sortie
<code>INPUT</code>	broche en entrée
<code>LED_BUILTIN</code>	13 (numéro de broche de la LED intégrée)

3.3.3 Structure du programme

Un programme Arduino respecte toujours une **structure spécifique** composée en trois parties :

- Les déclarations : **définitions** des constantes et des variables ;
- La fonction `setup()` : **configuration** de la carte (entrées, sorties, port série, ...);
- La fonction `loop()` : **instructions du programme exécutées** dans une **boucle infinie** (sans fin).

3.3.4 Documentation

Une référence complète du langage Arduino est disponible sur le site officiel [ici](#).

Chapitre 4

Les bases d'Arduino

4.1 Allumer une LED (sorties numériques)

4.1.1 Principe

Les entrées/sorties numériques de la carte Arduino UNO sont accessibles sur les broches 0 à 13.

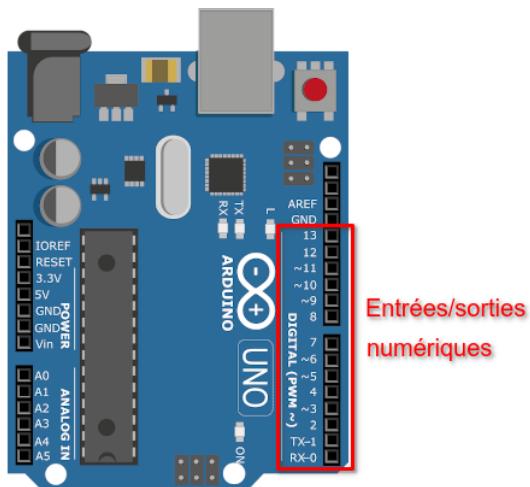


FIG. 1 – Entrées/sorties numériques

Avertissement

Une sortie numérique ne peut délivrer que 40 mA au maximum (200 mA pour l'ensemble des sorties). Au delà de cette valeur, la carte peut-être détériorée !

4.1.2 Montage

Une LED en série avec une résistance est connectée entre la broche 11 et la masse (GND).

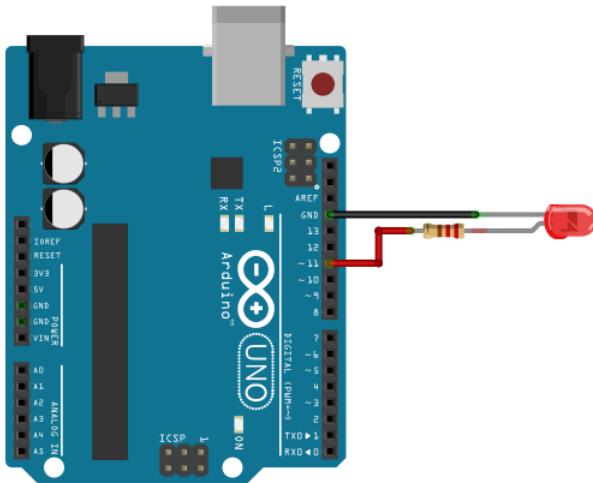


FIG. 2 – Branchement d'une LED sur la broche 11

La formule suivante donne le calcul de la résistance en fonction du courant nominal et de la couleur de la LED :

$$R = \frac{V_{CC} - V_S}{I}$$

- V_{CC} est la tension d'alimentation (5 V) ;
- $V_S \approx 2V$ est la tension de seuil de la LED (dépend de la couleur) ;
- I est l'intensité du courant généralement de l'ordre de 20 mA.

Une valeur de résistance de 220 Ω donne un bon compromis.

4.1.3 Programme en langage Arduino (C/C++)

```
int pinLED = 11; // LED connectée sur broche 11

void setup() {
    pinMode(pinLED, OUTPUT); // Broche LED paramétrée en sortie
}

void loop() {
    digitalWrite(pinLED, 0); // LED éteinte
    delay(1000); // Attendre 1000 ms = 1s
    digitalWrite(pinLED, 1); // LED allumée
    delay(1000); // Attendre 1000 ms = 1s
}
```

Dans la fonction `setup()` :

- `pinMode(LED,OUTPUT)` paramètre la broche LED en sortie (OUTPUT).

Dans la fonction `loop()` :

- `digitalWrite(LED,LOW)` fixe un niveau logique 0 (LOW) sur la broche LED.
- `digitalWrite(LED,HIGH)` fixe un niveau logique 1 (HIGH) sur la broche LED.

4.1.4 Applications

- Commande d'un actionneur (LED, relais, ...) en tout ou rien.
- Communication numérique.

4.2 Modifier l'intensité lumineuse d'une LED (sorties PWM)

4.2.1 Principe

La carte Arduino ne possède pas de vraies sorties analogiques. Mais il est possible de faire varier la valeur moyenne de la tension d'une sortie digitale (donc de faire varier l'intensité lumineuse d'une LED) en modifiant son rapport cyclique (durée de l'état haut par rapport à la période). C'est le principe de la Modulation à Largeur d'Impulsion (MLI) ou Pulse Width Modulation (PWM) en anglais.

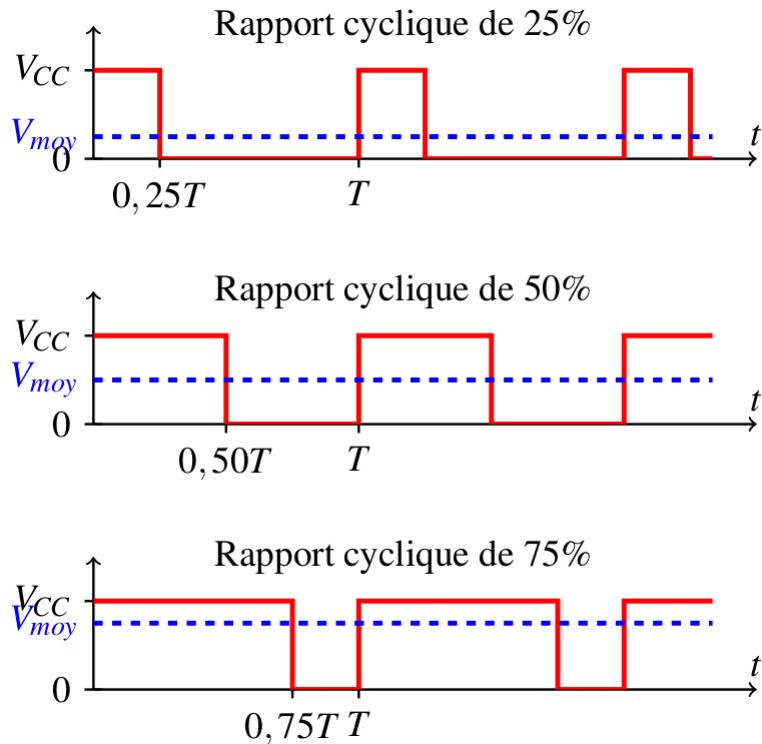


FIG. 3 – Principe de la MLI ou PWM

La carte Arduino UNO est dotée de **6 sorties PWM** sur les broches 3, 5, 6, 9, 10, 11.

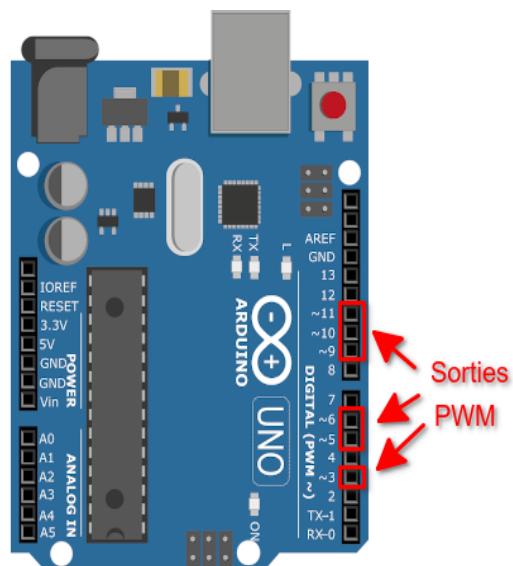


FIG. 4 – Sorties PWM de l'Arduino Uno R3

Avertissement

La fréquence d'un signal PWM est fixée à 490 Hz pour un arduino Uno R3 !

4.2.2 Montage

Une LED en série avec une résistance de $220\ \Omega$ est branchée sur la broche 11.

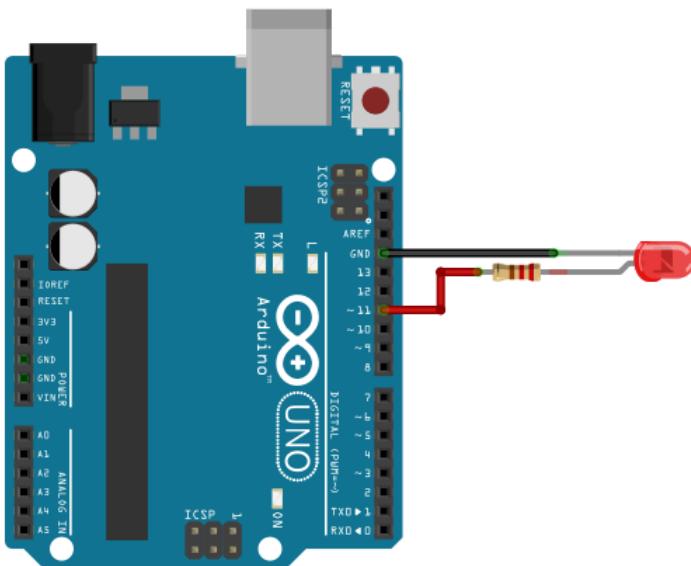


FIG. 5 – Branchement d'une LED sur la broche 11

4.2.3 Programme en langage Arduino (C/C++)

```
#define LED 11           // LED connectée à la broche 11

void setup() {
    pinMode(LED,OUTPUT); // Configuration de la broche LED en sortie
}

void loop() {
    analogWrite(LED,0);   // PWM à 0%
    delay(1000);          // Attendre 1 s
    analogWrite(LED,100); // PWM à 100/255 = 39%
    delay(1000);          // Attendre 1 s
    analogWrite(LED,200); // PWM à 200/255 = 78%
    delay(1000);          // Attendre 1 s
}
```

La fonction `analogWrite(LED,duty)` génère une modulation à largeur d'impulsion sur la broche 11 où `duty` est un nombre entier entre 0 et 255 respectivement pour un rapport cyclique entre 0% et 100%.

4.2.3.1 Applications

- Variation de l'intensité lumineuse d'une LED.
- Variation de la vitesse d'un moteur à courant continu.

- Obtention d'une tension constante par filtrage passe-bas (limitée en fréquence).

4.3 Afficher des messages (port série UART)

4.3.1 Principe

De base, les cartes Arduino ne possède pas d'écran pour afficher des messages. L'interface série (UART) reste le moyen le plus simple pour afficher des informations sur un ordinateur en provenance d'une carte Arduino.

4.3.2 Moniteur série

Le logiciel Arduino IDE intègre un **moniteur série** (dans le menu Outils > Moniteur série) pour lire des données au format texte (ASCII) envoyées par le microcontrôleur.

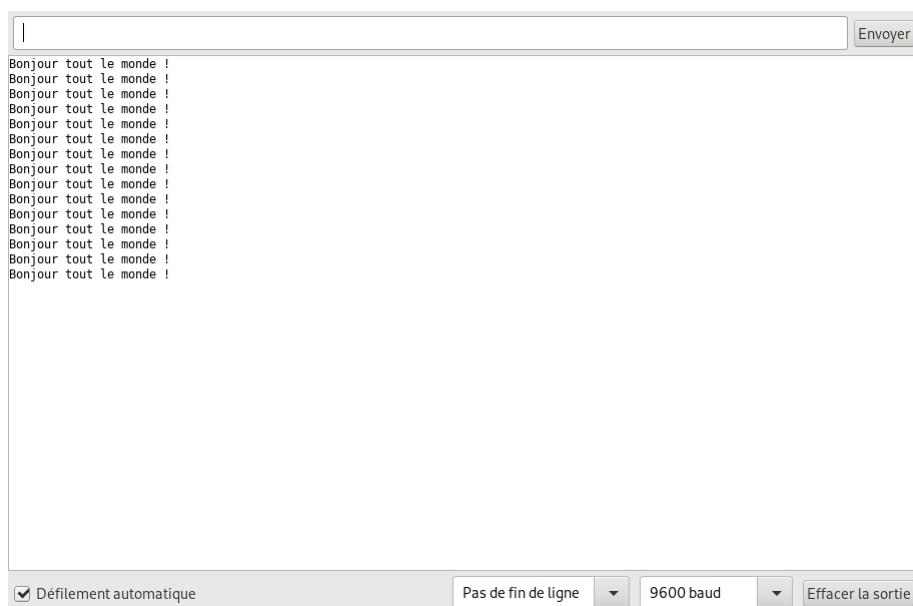


FIG. 6 – Moniteur série du d'Arduino IDE 1.8

Le moniteur série offre aussi la possibilité de transmettre des données vers le microcontrôleur à l'aide du clavier !

Note

Il est possible d'utiliser d'autres logiciels de communication série comme [Putty](#) ou encore [Termite](#).

4.3.3 Programme de test

Le code source ci-dessous transmet un entier N dans le moniteur série d'Arduino IDE. Cet entier, initialisé à 0, est incrémenté de 1 toutes les secondes.

```
int n = 0;

void setup() {
    Serial.begin(9600);      // Paramétrage du port série
```

(suite sur la page suivante)

```

}
void loop() {
    Serial.print("N = ");      // Affichage
    Serial.println(n);        // Affichage du contenu de la variable n
    n = n + 1;                // Incrémentation de la variable n
    delay(1000);              // Temporisation de 1s
}

```

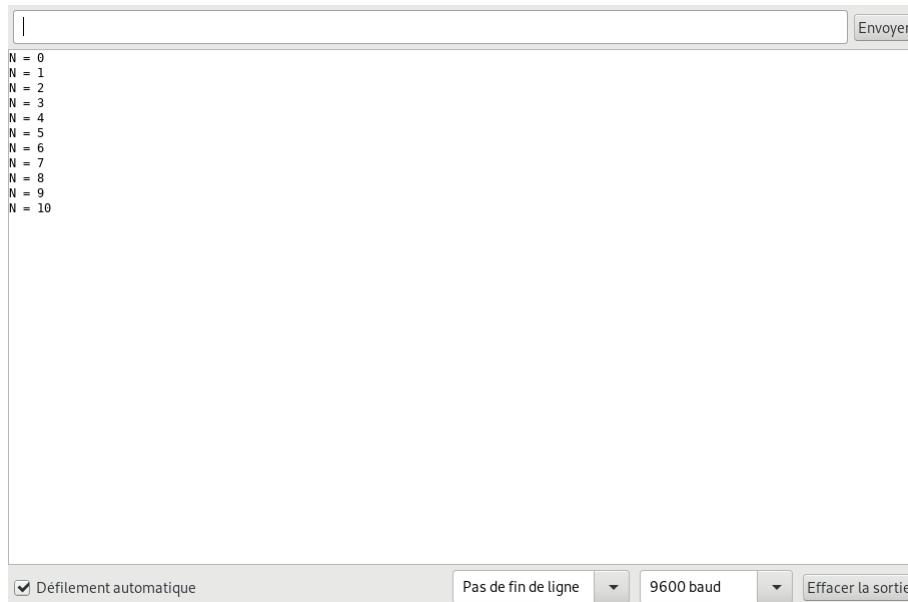


FIG. 7 – Affichage dans le moniteur série

- L'instruction `Serial.begin(9600)` paramètre le port série à 9600 baud (rapidité).
- `Serial.print("N = ")` affiche la chaîne de caractères `N =` dans le moniteur série.
- `Serial.println(n)` affiche le contenu de la variable `n` suivie d'un saut de ligne.

4.3.4 Applications

- Affichage d'une ou plusieurs mesures sur l'écran d'un ordinateur.
- Affichage des données d'une acquisition au format CSV pour exploitation par un tableur, des logiciels spécialisés (Regressi, Latis, ...) ou des calculatrices Web (ex. Desmos).

4.4 Mesurer une tension (CAN)

4.4.1 Principe

La mesure d'une tension par un microcontrôleur est prise en charge en interne par un **Convertisseur Analogique-Numérique (CNA)**.

Ce type de conversion est importante en sciences physique. Par exemple, elle permet d'obtenir la mesure d'une grandeur physique provenant d'un **capteur** sous la forme d'une tension électrique.

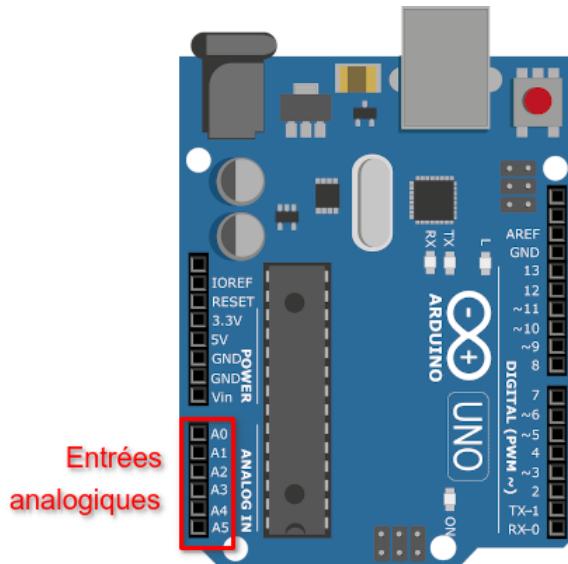


FIG. 8 – Entrées analogiques de l'Arduino Uno R3

Le microcontrôleur de l'Arduino UNO intègre **6 entrées analogiques** disponibles sur les broches A0 à A5. Par défaut, la tension d'entrée doit-être comprise entre 0 V et 5 V (Vref).

Le conversion renvoie un **nombre entier naturel sur 10 bits** compris entre 0 et 1023.

La résolution analogique (**quantum**) est donc :

$$q = \frac{5}{1023} \approx 4,89 \text{ mV}$$

Avertissement

Le tension appliquée sur les entrées analogiques doivent être **strictement comprise entre 0 V et 5 V** sous peine de détruire le microcontrôleur.

4.4.2 Montage

Un potentiomètre (pont diviseur de tension) est branché entre la masse (GND) et la tension d'alimentation (5V). Ce potentiomètre délivre donc une **tension réglable** en 0 V et 5 V sur l'entrée A0.

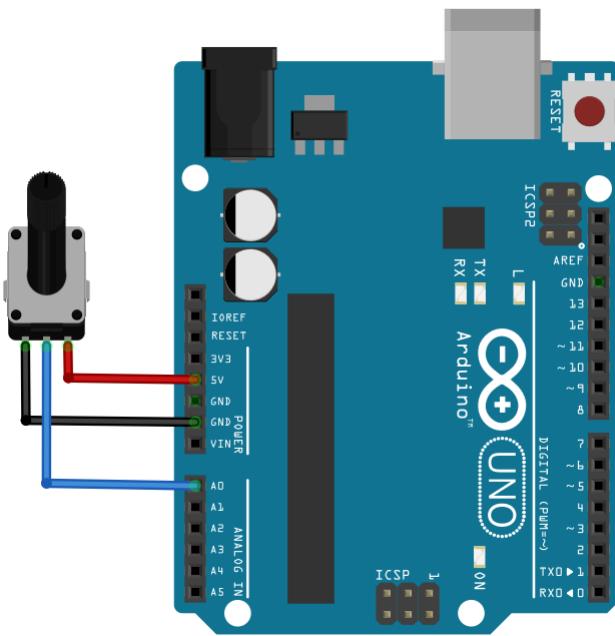


FIG. 9 – Montage potentiométrique sur l'entrée analogique A0

4.4.3 Programme Arduino

Le programme suivant lit la tension sur l'entrée A0 et affiche sa valeur (en volt) dans le moniteur série du logiciel Arduino IDE.

```
/*
 * Lecture de l'entrée analogique A0 sur le port série.
 */

int N; // Entier lu sur A0 compris entre 0 et 1023 (10bits)
float tension; // La tension calculée à partir de N

void setup() {
    Serial.begin(9600); // Paramétrage port série
}

void loop() {
    N = analogRead(A0); // Lecture valeur sur A0
    tension = N*5.0/1023; // Calcul de la tension
    Serial.print("Valeur lire sur A0 = ");
    Serial.println(N); // Affichage valeur sur A0
    Serial.print("Tension = ");
    Serial.println(tension); // Affichage de la tension
    delay(1000); // temporisation de 1 s
}
```

- La fonction `analogRead(A0)` retourne un entier sur 10 bits compris entre 0 (pour 0V) et 1023 (pour 5V).
- L'expression `N*5.0/1023` calcule la valeur de la tension en volt.

4.4.4 Applications

- Interface avec un circuit comportant un capteur.

- Un potentiomètre est un capteur de position.

Chapitre 5

Nouveaux programmes du lycée

5.1 Capteur résistif - CTN (seconde générale)

Programme de seconde générale 2019 - Enseignement commun

Mesurer une grandeur physique à l'aide d'un capteur électrique résistif. **Produire et utiliser une courbe d'étalonnage** reliant la résistance d'un système avec une grandeur d'intérêt (température, pression, intensité lumineuse, etc.). Utiliser un dispositif avec microcontrôleur et capteur.

5.1.1 Principe

La résistance électrique d'un capteur résistif varie en fonction de la grandeur physique mesurée (ex. température, pression, intensité lumineuse, ...).



FIG. 1 – Capteur de température CTN / Capteur de lumière LDR

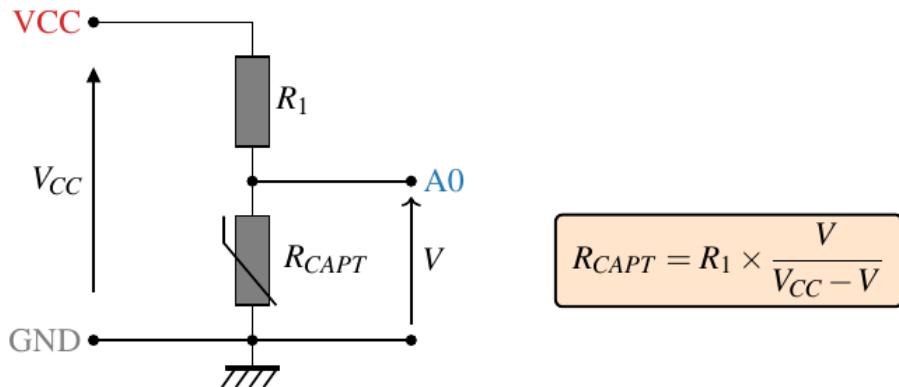
Pour obtenir une valeur de cette grandeur physique, il est nécessaire de **mesurer la résistance du capteur**.

Malheureusement un microcontrôleur ne mesure que des tensions sur ses entrées analogique (ex. A0, A1, ...) pour Arduino). Il faut donc **adapter le capteur dans un montage électrique** (ex. pont diviseur de tension, pont wheatstone).

5.1.2 Montages pour la mesure d'une résistance

La plupart des modules avec capteur résistif utilise un **pont diviseur de tension** pour la mesure de la résistance du capteur. Par rapport au pont Wheatstone, cette méthode présente l'avantage d'être simple à mettre en oeuvre.

5.1.2.1 Montage 1 : capteur connecté à la masse



- VCC est la tension d'alimentation du microcontrôleur.
- L'entrée analogique A0 mesure la tension V aux bornes du capteur résistif.

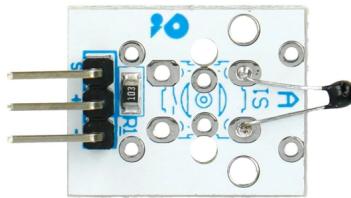
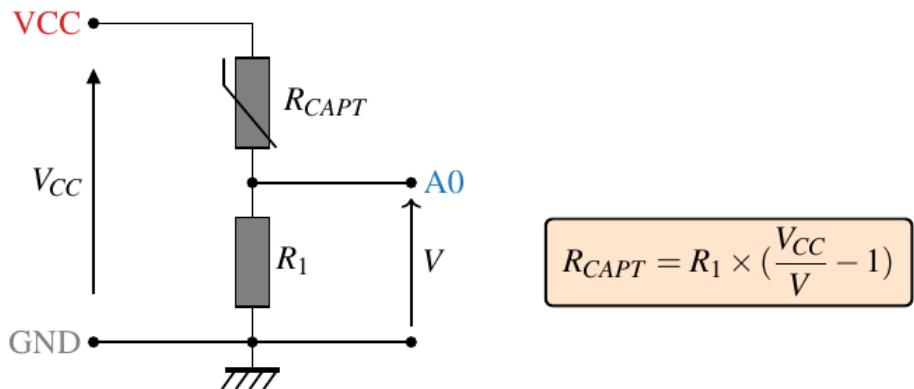
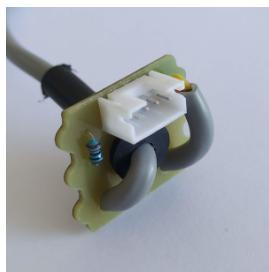


FIG. 2 – Module Velleman VMA320 ($R_1 = 10 \text{ k}\Omega$)

5.1.2.2 Montage 2 : capteur connecté à Vcc

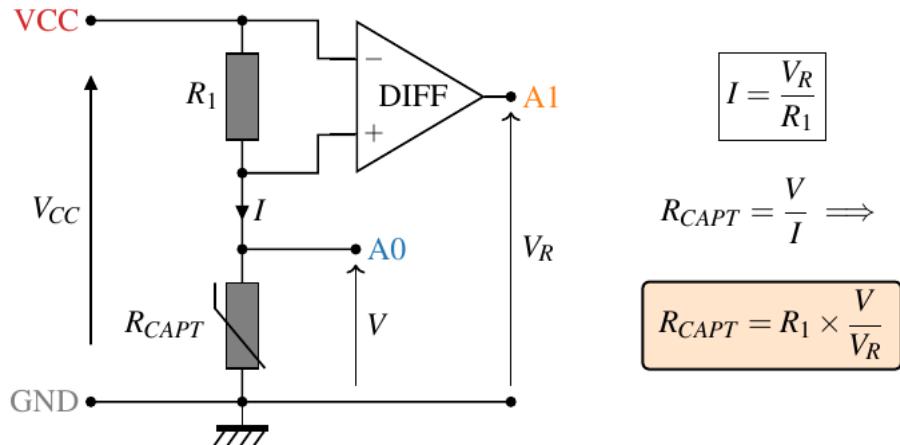


- VCC est la tension d'alimentation du microcontrôleur.
- L'entrée analogique A0 mesure la tension V aux bornes de la résistance R.

FIG. 3 – Module Grove - Temperature Sensor V1.2 ($R_1 = 100 \text{ k}\Omega$)FIG. 4 – Module CTN Plug'uino ($R_1 = 10 \text{ k}\Omega$)

5.1.2.3 Montage 3 : mesure de la tension et du courant (ex. capteurs Educaduino)

En plus de la **mesure de la tension** du capteur, une **mesure du courant** est aussi réalisée à partir de la tension aux bornes de la résistance R par l'intermédiaire d'un amplificateur différentiel. La résistance du capteur est alors calculée avec la **loi d'Ohm**.



- L'entrée analogique A0 mesure la tension V aux bornes du capteur résistif.
- L'entrée analogique A1 mesure la tension V_R aux bornes de la résistance R .

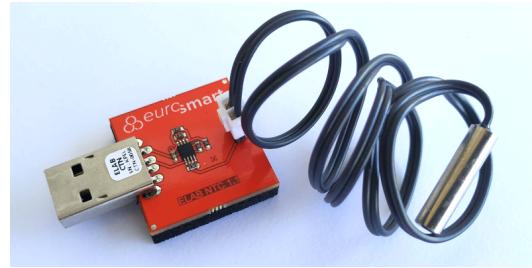


FIG. 5 – Module CTN 10K Educaduino LAB ($R_1 = 10 \text{ k}\Omega$)

5.1.3 Cas d'un capteur résistif CTN

Une CTN est un **capteur résistif à coefficient de température négatif** ...

5.1.3.1 Courbe d'étalonnage

Les mesures sont effectuées avec un thermomètre et un **ohmmètre**.

T (°C)	R (Ohm)
2.4	25455
5.1	22714
10.0	18622
15.1	15201
20.0	12607
25.0	10475
30.0	8740
35.1	7333
40.0	6194
45.0	5217
50.0	4358
55.1	3689
60.0	3120
65.1	2647
70.1	2264
75.1	1926
80.0	1658

Téléchargement : [data_ctn_1.txt](#) (mesures au format CSV).

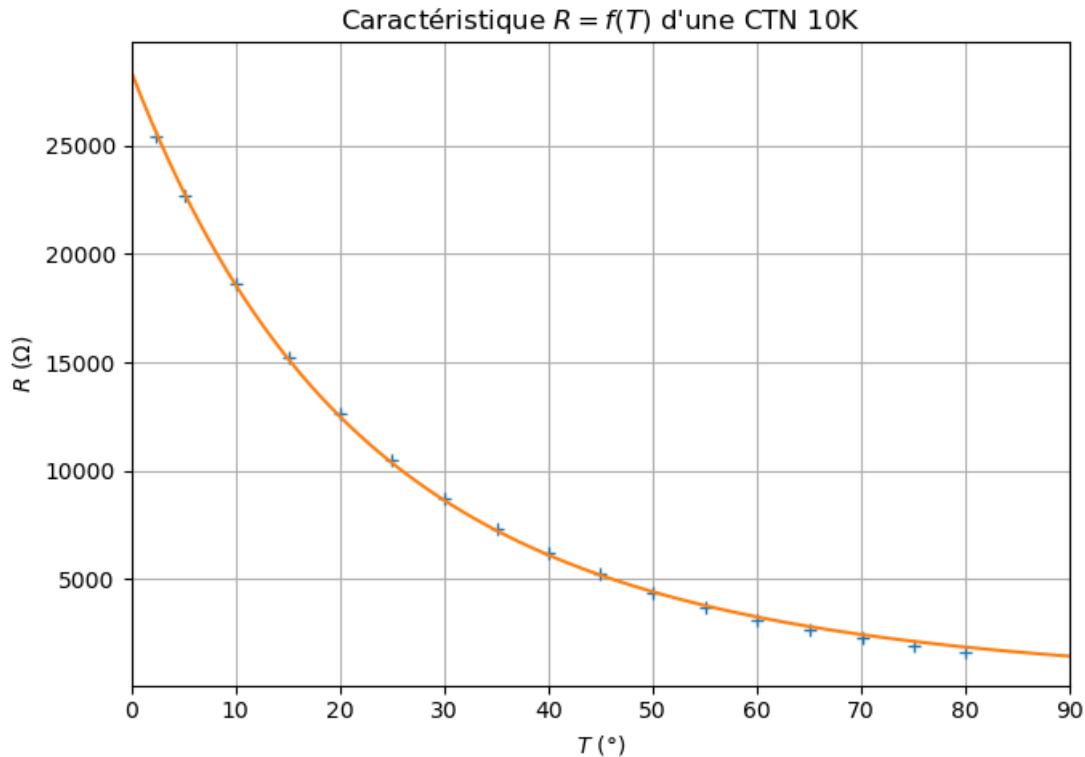


FIG. 6 – Courbe d'étalonnage d'une CTN 10k

Note

Dans cet exemple, la résistance mesurée prend la valeur particulière de 10 kΩ pour 25°C !

5.1.3.2 Relation de Steinhart-Hart

Sur une grande plage de variation, la relation entre la température (en K) et la résistance de la CTN est :

$$\frac{1}{T} = A + B \times \ln(R) + C \times (\ln(R))^3$$

A, B et C sont les coefficients de Steinhart-Hart. Ils sont donnés par le constructeur ou peuvent se déterminer expérimentalement à l'aide du programme Python `steinhart-hart.py` à partir de trois points de la courbe d'étalonnage.

Résultats obtenus à partir du programme Python :

$$A = 1,144 \cdot 10^{-3} K^{-1} \quad B = 2,078 \cdot 10^{-3} K^{-1} \quad C = 3,610 \cdot 10^{-7} K^{-1}$$

5.1.3.3 Relation simplifiée de Steinhart-Hart

Sur une plage de variation plus réduite de la température, la relation de Steinhart-Hart permet d'écrire :

$$R \approx R_0 \times e^{\beta(\frac{1}{T} - \frac{1}{T_0})}$$

- R_0 est la valeur de la résistance pour la température T_0 .
- β est le coefficient de température (en K).

Ces coefficients sont généralement donnés par le constructeur (datasheet).

Module	Référence CTN	R_0 (kΩ)	β (K)
Grove - Temperature Sensor V1.2	NCP18WF104F03RC	100	4255 (25/80°C)
Velleman VMA320	NTC-MF52 3950	10	3950

Les coefficients R_0 et β peuvent également être déterminés par une modélisation de la caractéristique sur la plage de température d'utilisation de la CTN.

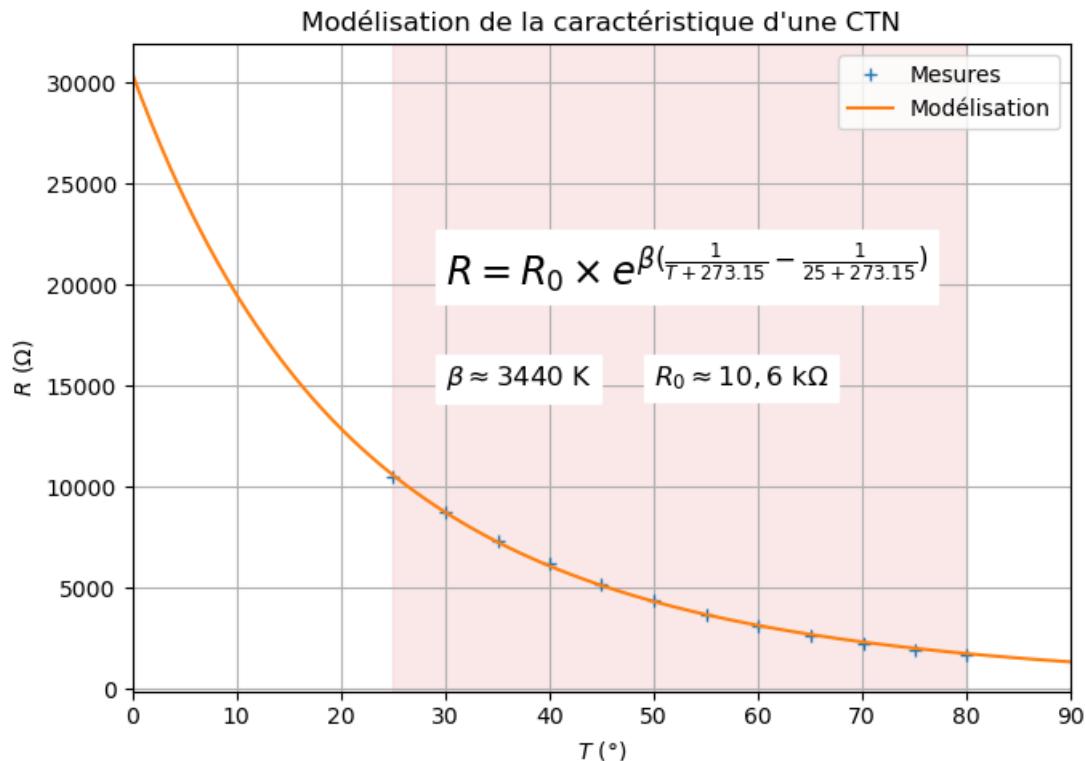


FIG. 7 – Modélisation de la caractéristique de la CTN de 25°C à 80°C

Résultat de la modélisation de $T_0 = 25^\circ\text{C}$ à 80°C :

$$R_0 \approx 10,6 \text{ k}\Omega \quad \text{et} \quad \beta \approx 3440 \text{ K}$$

Inversement, le calcul de la température (en K) s'obtient à partir de la relation suivante :

$$\boxed{\frac{1}{T} = \frac{1}{\beta} \times \ln\left(\frac{R}{R_0}\right) + \frac{1}{T_0}}$$

5.1.4 Programme 1 : mesure de la résistance de la CTN

La CTN est connectée à la masse (montage 1). L'entrée analogique A0 mesure la tension du capteur.

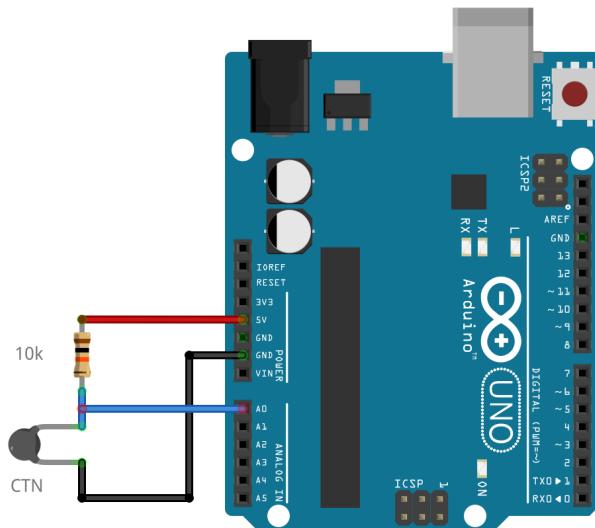


FIG. 8 – Mesure de la résistance d'une CTN sur l'entrée analogique A0

```
/*
 * Mesure de la résistance d'une CTN
 */

// Déclaration des variables pour la mesure de la résistance
float Vcc = 5.0;      // Tension d'alimentation
float R1 = 10000;     // Résistance du pont diviseur de tension
float U;              // Tension de la CTN en volt
float R_mes;          // Résistance mesurée de le CTN en Ohm

void setup() {
    Serial.begin(9600); // Paramétrage du port série
}

void loop() {
    U = analogRead(A0)*Vcc/1023; // Lecture tension en V
    R_mes = R1*U/(Vcc-U);      // Calcul de la résistance
    Serial.print("R = ");       // Affichage
    Serial.print(R_mes);        // Affichage
    Serial.println(" Ohm");     // Affichage + sout de ligne
    delay(1000);               // Temporisation de 1s
}
```



```

r = 8552.01 Ohm
R = 8566.24 Ohm
R = 8566.24 Ohm
R = 8600.00 Ohm
R = 8600.00 Ohm
R = 8633.88 Ohm
R = 8633.88 Ohm
R = 8633.88 Ohm
R = 8633.88 Ohm
R = 8667.88 Ohm
R = 8667.88 Ohm
R = 8702.01 Ohm
R = 8702.01 Ohm
R = 8736.26 Ohm
R = 8736.26 Ohm

```

Défilement automatique Afficher l'horodatage Pas de fin de l... 9600 baud Effacer la sortie

FIG. 9 – Résultats dans le moniteur série d'Arduino IDE

5.1.5 Programme 2 : thermomètre numérique (application)

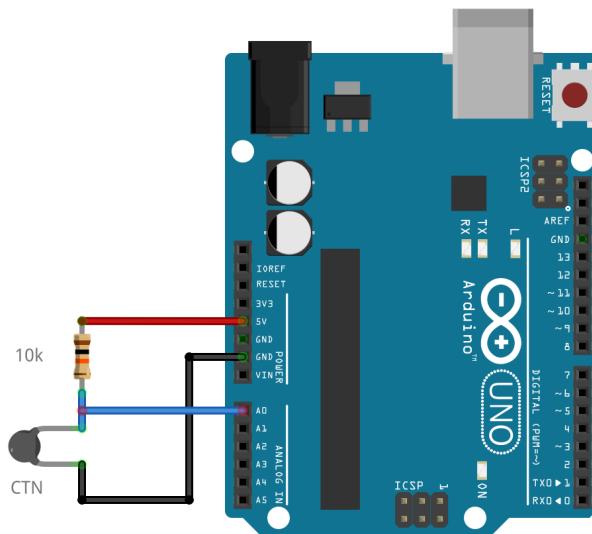


FIG. 10 – Mesure de la température donnée par une CTN sur l'entrée analogique A0

Rappel de relation simplifiée de Steinhart-Hart :

$$\frac{1}{T} = \frac{1}{\beta} \times \ln\left(\frac{R}{R_0}\right) + \frac{1}{T_0}$$

avec pour notre CTN :

$$R_0 \approx 10,5 \text{ k}\Omega \quad \text{et} \quad \beta \approx 3290 \text{ K}$$

```
/*
 * Mesure de la résistance d'une CTN et
 * Calcul de la température à partir de la relation simplifiée de Steinhart-Hart
 */
```

(suite sur la page suivante)

(suite de la page précédente)

```

// Déclaration des variables pour la mesure de la résistance
float Vcc = 5.0;      // Tension d'alimentation
float R1 = 10000;     // Résistance du pont diviseur de tension
float U;              // Tension CTN en volt
float R_mes;          // Résistance CTN en ohm

// Déclaration des variables pour le calcul de la température
float T0 = 25;        // Température de référence (25 °C)
float R0 = 10600;      // Resistance R0 à T0 = 25 °C
float beta = 3440;    // Coefficient de température en K
float inverse;        // Inverse de la température en K-1
float T_K;            // Temperature en K
float T;               // Temperature en °C

void setup() {
    Serial.begin(9600); // Paramétrage du port série
}

void loop() {
    U = analogRead(A0)*5.0/1023;           // Lecture de la tension en V
    R_mes = R1*U/(Vcc-U);                // Calcul de la résistance mesurée
    Serial.print("R = ");                 // Affichage
    Serial.print(R_mes);                  // Affichage
    Serial.println("Ohm");                // Affichage + retour à la ligne

    inverse = 1/beta * log(R_mes/R0) + 1/(T0+273.15); // Calcul de l'inverse de la
    ↪température
    T_K = 1/inverse;                     // Calcul de la température en
    ↪Kelvin
    T = T_K - 273.15;                   // Calcul de la température en °C
    Serial.print("T = ");                 // Affichage
    Serial.print(T);                     // Affichage
    Serial.println("°C");                // Affichage + retour à la ligne
    delay(1000);                        // Pause
}

```



FIG. 11 – Résultats dans le moniteur série d'Arduino IDE

5.1.6 A retenir

Placer un **capteur résistif** (température, pression, lumière, ...) dans un **pont diviseur de tension** reste une **solution simple pour mesurer sa résistance** à l'aide d'un microcontrôleur.

5.2 Émission d'un son (seconde générale)

Programme de seconde générale 2019 - Enseignement commun

Utiliser un dispositif comportant un microcontrôleur pour produire un signal sonore.

5.2.1 Principe

Certains microcontrôleurs (Arduino et Micro :bit) ne possédant pas de sortie analogique (CNA) pour générer des tensions sinusoïdales, une méthode simple pour produire un son est de générer une tension carrée (entre 0 et Vcc) de fréquence f à l'entrée d'un haut-parleur.

Le son obtenu par cette technique n'est pas pur car il comporte des harmoniques aux fréquences $3f$, $5f$, $7f$, ...

Avertissement

Pour les faibles fréquences, le son devient « métallique » avec la présence importante d'harmoniques !

5.2.2 Méthode 1 : construire le signal carré

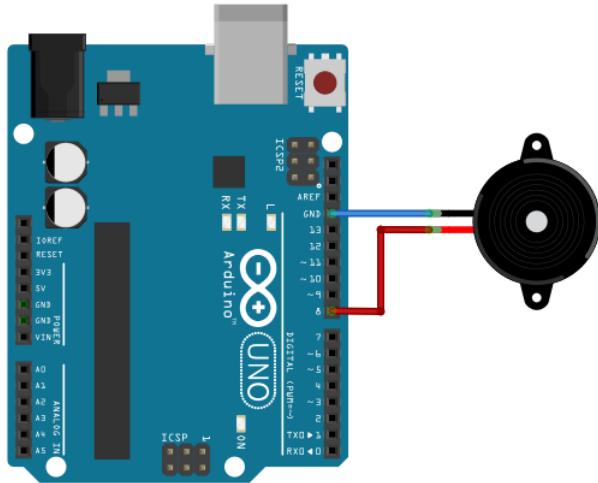


FIG. 12 – Branchement d'un haut-parleur sur la broche 8

```
#define brocheHP 8

float frequence=440;
float periode=1/frequence;
```

(suite sur la page suivante)

(suite de la page précédente)

```

void setup(){
    pinMode(brocheHP, OUTPUT);
}

void loop(){
    digitalWrite(brocheHP,HIGH);
    delayMicroseconds(1000000*periode/2.0);
    digitalWrite(brocheHP,LOW);
    delayMicroseconds(1000000*periode/2.0);
}

```

5.2.3 Méthode 2 : utiliser une fonction spéciale

```

/*
Exemple Arduino
*/
#include "pitches.h"

// notes in the melody:
int melody[] = {NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, NOTE_B3, NOTE_C4};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {4, 8, 8, 4, 4, 4, 4};

void setup() {
for (int thisNote = 0; thisNote < 8; thisNote++) {
    int noteDuration = 1000/noteDurations[thisNote];
    tone(8, melody[thisNote],noteDuration);
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    noTone(8);
}
}

void loop() {
// Pas de boucle ici !
}

```

5.2.4 Applications

- Hauteur d'un son (relation entre fréquences et notes).
- Générer une mélodie à partir de plusieurs notes.

5.3 Mesurer la célérité d'un son (première générale)

Programme de première générale 2019 - Enseignement de spécialité.

Exploiter la relation entre la durée de propagation, la distance parcourue par une perturbation et la célérité, notamment pour localiser une source d'onde. Déterminer, par exemple à l'aide d'un microcontrôleur ou d'un smartphone,

une distance ou la célérité d'une onde.

5.3.1 Présentation du module HC-SR04

Les modules du type HC-SR04 sont des émetteurs-récepteurs ultrasonores fonctionnant par réflexion. Ils sont utilisés généralement dans des applications comme télémètre (< 5 m).



FIG. 13 – Module HC-SR04

5.3.1.1 Fonctionnement

- Le module est alimenté entre GND et Vcc (généralement 5 V ou 3,3 V sur certains modules).
- Le déclenchement d'une mesure (émission d'une salve) se fait par une brève impulsion ($> 10 \mu\text{s}$) sur l'entrée trig.
- La durée que prend l'onde pour aller de l'émetteur au récepteur est celle de l'impulsion renvoyée sur la sortie echo.

5.3.1.2 Modification du module

Tel qu'il est vendu, ce module n'a peu d'intérêt en sciences physiques car les signaux électriques sur l'émetteur et le récepteur ne sont pas accessibles. Il est possible de résoudre ce problème en y soudant des **connecteurs supplémentaires** (voir photo ci-dessous) et de **visualiser les signaux correspondants à l'oscilloscope ou avec une interface d'acquisition**.

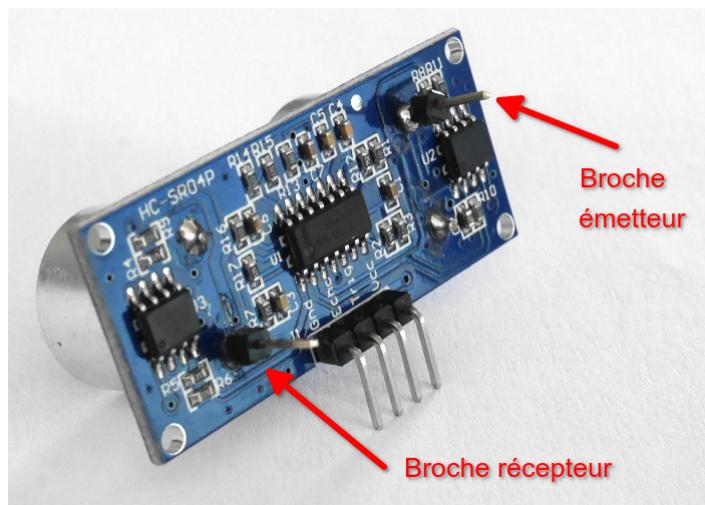


FIG. 14 – Détail du module HC-SR04

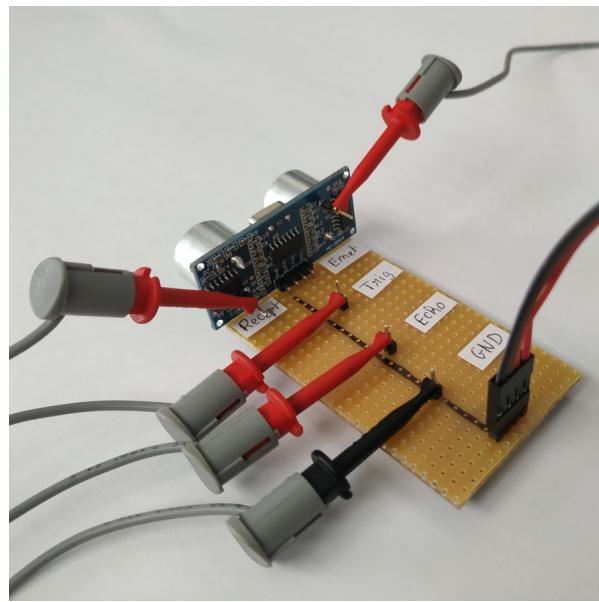


FIG. 15 – Branchement du module HC-SR04 modifié

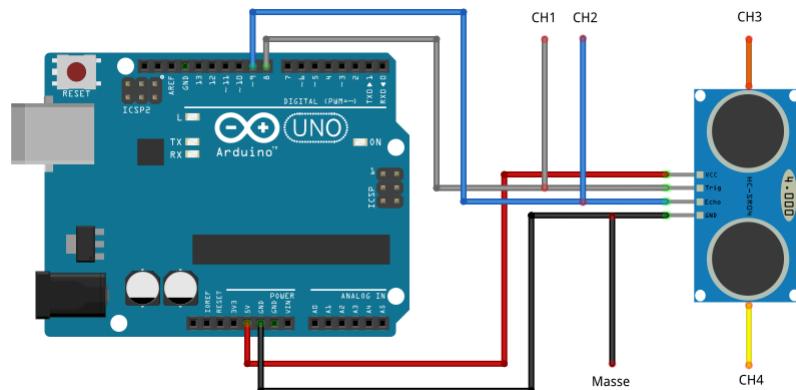


FIG. 16 – Montage avec oscilloscope ou interface d'acquisition

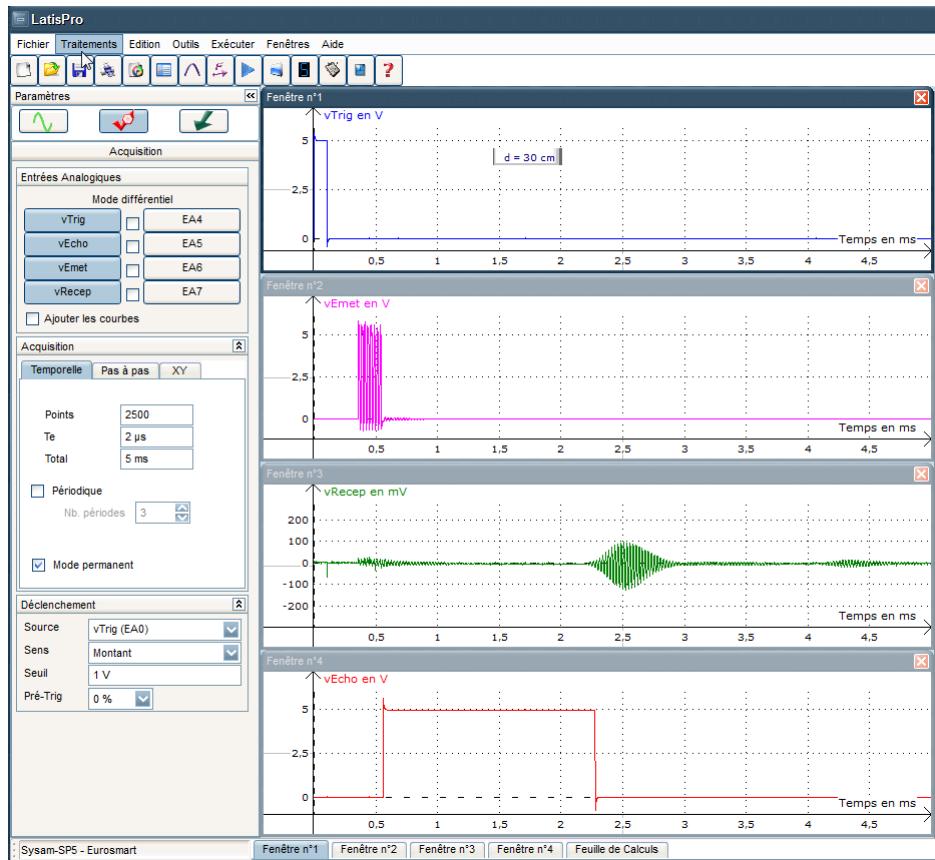


FIG. 17 – Mesures obtenues dans Latis avec Sysam SP5

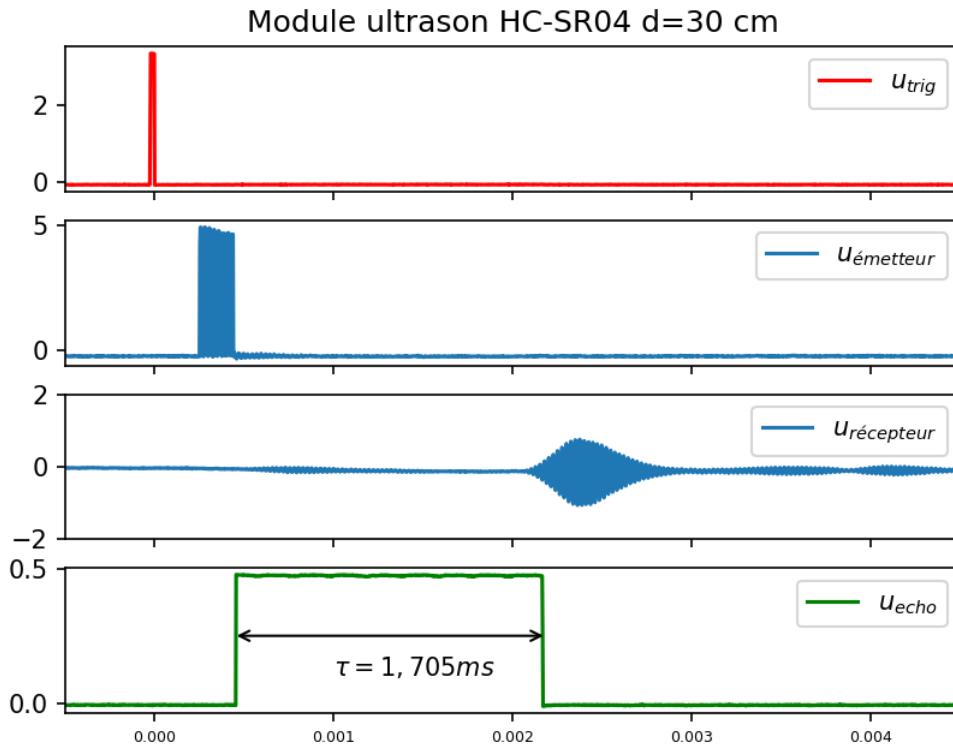


FIG. 18 – Mesures à l'oscilloscope pour une distance de 30 cm

5.3.2 Mesure de la célérité du son

La manipulation consiste à relever la durée de l'écho sonore à l'aide du microcontrôleur pour différentes distances et déduire la célérité du son.

$$c = \frac{2 \times d}{\Delta t}$$

Algorithme :

```

TRIG <- 0

REPETER :
    TRIG <- Vcc                      # Début impulsions sur Trig
    Attendre 10 µs
    TRIG <- 0                          # Fin impulsions sur Trig
    Dt   <- Durée impulsions sur Echo # Mesure
    Afficher Dt
    Attendre 1 s
  
```

Pour plus de précision, il est possible de modifier le programme afin de **réaliser plusieurs mesures** de la durée de l'écho et d'en déduire sa **moyenne**.

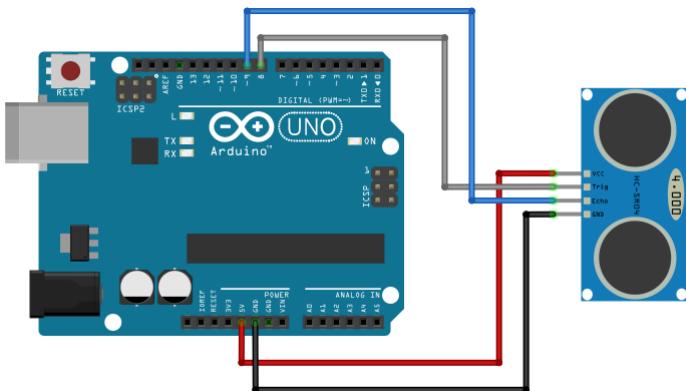


FIG. 19 – Montage célérité son

```
// Mesure de la durée de l'écho sonore

#define pinTrig 8          // Trig sur broche 8
#define pinEcho 9          // Echo sur broche 9

long dureeEcho;           // Durée de l'Echo

void setup() {
    pinMode(pinTrig,OUTPUT);      // Broche Trig en sortie
    digitalWrite(pinEcho,LOW);     // Sortie Trig à l'état bas
    pinMode(pinEcho,INPUT);       // Broche Echo en entrée
    Serial.begin(9600);          // Paramétrage du port série
}

void loop() {
    digitalWrite(pinTrig,HIGH);   // Début impulsion de déclenchement
    delayMicroseconds(10);        // Attendre 10 microseconde
    digitalWrite(pinTrig,LOW);    // Fin impulsion (Etat bas)
    dureeEcho = pulseIn(pinEcho,HIGH); // Mesure de la durée de l'impulsion sur Echo
    Serial.print("Durée (µs) = ");
    Serial.println(dureeEcho);    // Affichage sur port série
    delay(1000);                // Attendre 1s
}
```

5.3.3 Application : réalisation d'un télémètre

Connaissant la célérité du son, la distance par rapport à un obstacle est calculée par le microcontrôleur à l'aide de la relation suivante :

$$d = \frac{c \times \Delta t}{2}$$

Il suffit d'ajouter le calcul de la distance juste après la mesure de la durée.

```
// Mesure de la durée d'une distance

#define pinTrig 8          // Trig sur broche 8
#define pinEcho 9          // Echo sur broche 9

long dureeEcho;           // Durée de l'Echo
```

(suite sur la page suivante)

(suite de la page précédente)

```

float distance;           // Distance en module et réflecteur
float vitesse = 345 ;    // Vitesse obtenue

void setup() {
pinMode(pinTrig,OUTPUT);      // Broche Trig en sortie
digitalWrite(pinEcho,LOW);    // Sortie Trig à l état bas
pinMode(pinEcho,INPUT);       // Broche Echo en entrée
Serial.begin(9600);          // Paramétrage du port série
}

void loop() {
digitalWrite(pinTrig,HIGH);    // Début impulsion de déclenchement
delayMicroseconds(10);         // Attendre 10 microseconde
digitalWrite(pinTrig,LOW);     // Fin impulsion (Etat bas)
dureeEcho = pulseIn(pinEcho,HIGH); // Mesure de la durée de l'impulsion sur Echo
distance = (vitesse*dureeEcho*1E-6)/2; // Calcul de la distance
Serial.print("Duree (us) = ");   // Affichage sur port série
Serial.println(dureeEcho);
Serial.print("Distance (m) = "); // Affichage sur port série
Serial.println(distance);
delay(1000);                  // Attendre 1s
}

```

En autonome avec un écran LCD :

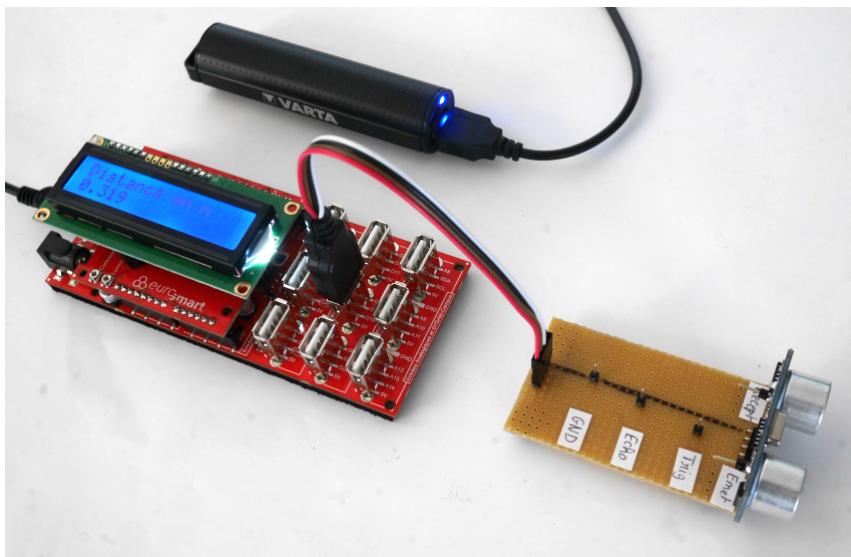


FIG. 20 – Télémètre sur Educaduino-Lab LCD

```

//Application : télémètre sur écran LCD 2x16

#include <LiquidCrystal.h>           // Importation de la librairie LiquidCrystal
#define pinTrig 8                 // Trig sur broche 8
#define pinEcho 9                 // Echo sur broche 9

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Brochage de l'afficheur

float distance;           // Distance en module et réflecteur
long dureeEcho;           // Durée mesurée

```

(suite sur la page suivante)

```

float vitesse = 345 ; // Vitesse obtenue

void setup() {
    pinMode(pinTrig,OUTPUT); // Broche Trig en sortie
    digitalWrite(pinEcho,LOW); // Sortie Trig à l état bas
    pinMode(pinEcho,INPUT); // Broche Echo en entrée
    lcd.begin(16, 2); // fixe le nombre de colonnes et de lignes de l'afficheur
}

void loop() {
    digitalWrite(pinTrig,HIGH); // Début impulsion de déclenchement
    delayMicroseconds(10); // Attendre 10 microseconde
    digitalWrite(pinTrig,LOW); // Fin impulsion (Etat bas)
    dureeEcho = pulseIn(pinEcho,HIGH); // Mesure de la durée de l'impulsion sur l'Echo
    distance = (vitesse*dureeEcho*1E-6)/2; // Calcul de la distance
    lcd.setCursor(0,0); // place le curseur au début de la ligne 0
    lcd.print("Distance en m"); // Affiche la légende
    lcd.setCursor(0,1); // place le curseur au début de la ligne 1
    lcd.print(distance); // Affiche la valeur de la distance
    delay(1000); // Attendre 1s
}

```

5.3.4 A retenir

- Le module HC-SR04 fournit un **signal Echo pour la mesure automatique de la durée** de propagation du son.
- La fonction **pulseIn()** **mesure cette durée**.

5.4 Mesurer une pression - Loi de Mariotte (première générale)

Programme de première générale 2019 - Enseignement de spécialité.

Tester la loi de Mariotte, par exemple en utilisant un dispositif comportant un microcontrôleur.

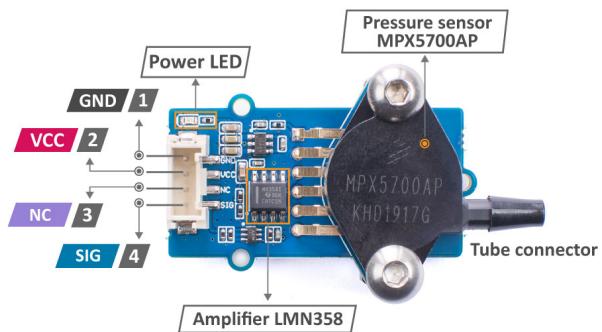
5.4.1 Principe

La manipulation consiste à vérifier la loi de Mariotte (à température et à quantité de matière constantes) :

$$P \times V = \text{Constante}$$

5.4.2 Capteur de pression absolue MPX5700AP

Le MPX5700AP est un capteur analogique de pression absolue (15 kPa à 700 kPa - 5V).



- 1** : Connected to the system GND
- 2** : Power supply from grove 5V/3.3V
- 3** : Not connected
- 4** : Analog signal output

FIG. 21 – Capteur MPX5700AP Grove (15 kPa à 700 kPa) (image : seeedstudio)

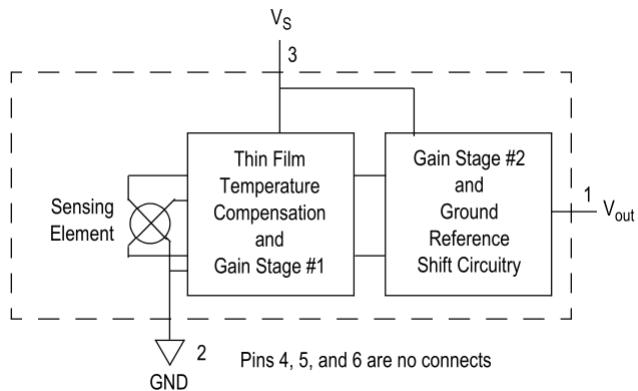


FIG. 22 – Schéma électrique (source : Freescale Semiconductor, Inc.)

Table 1. Operating Characteristics ($V_S = 5.0$ Vdc, $T_A = 25^\circ\text{C}$ unless otherwise noted, $P1 > P2$. Decoupling circuit shown in required to meet electrical specifications.)

Characteristic	Symbol	Min	Typ	Max	Unit
Pressure Range ⁽¹⁾ Gauge, Differential: MPX5700D Absolute: MPX5700A	P_{OP}	0 15	—	700 700	kPa
Supply Voltage ⁽²⁾	V_S	4.75	5.0	5.25	Vdc
Supply Current	I_O	—	7.0	10	mAdc
Zero Pressure Offset ⁽³⁾ Gauge, Differential (0 to 85°C) Absolute (0 to 85°C)	V_{off}	0.088 0.184	0.2 —	0.313 0.409	Vdc
Full Scale Output ⁽⁴⁾ (0 to 85°C)	V_{FSO}	4.587	4.7	4.813	Vdc
Full Scale Span ⁽⁵⁾ (0 to 85°C)	V_{FSS}	—	4.5	—	Vdc
Accuracy ⁽⁶⁾ (0 to 85°C)	—	—	—	± 2.5	% V_{FSS}
Sensitivity	V/P	—	6.4	—	mV/kPa
Response Time ⁽⁷⁾	t_R	—	1.0	—	ms
Output Source Current at Full Scale Output	I_{O+}	—	0.1	—	mAdc
Warm-Up Time ⁽⁸⁾	—	—	20	—	ms

FIG. 23 – Caractéristiques (source : Freescale Semiconductor, Inc.)

5.4.3 Calcul de la pression

La mesure de pression (en kPa) est donnée par la relation pour une alimentation de 5 V :

$$P = \frac{P_{max} - P_{min}}{V_{max} - V_{min}} \times (v_{out} - V_{min}) + P_{min}$$

Soit :

$$P = \frac{700 - 15}{4,7 - 0,2} \times (v_{out} - 0,2) + 15 \quad \Rightarrow \quad P \approx 152 \times (v_{out} - 0,2) + 15$$

5.4.4 Programme Arduino

```
/*
 * Mesure d'une pression absolue
 * Capteur Grove MPX5700AP (15 kPa à 700 kPa) branché sur la broche A0
 * David THERINCOURT - 2025
 */

float tension ; // Tension mesurée (V)
float pression ; // Pression mesurée (kPa)

void setup() {
    Serial.begin(9600); // Initialisation du port série
}

void loop() {
    tension = analogRead(A0)*5.0/1023 ; // Lecture de la tension sur A0 en volt
    pression = 152*(tension-0.2) + 15 ; // Calcul de la pression en kPa
}
```

(suite sur la page suivante)

(suite de la page précédente)

```

Serial.print("U = ");
Serial.print(tension);
Serial.print(" V => ");
Serial.print("P = ");
Serial.print(pression, 0);
Serial.println(" kPa");
delay(1000);
}

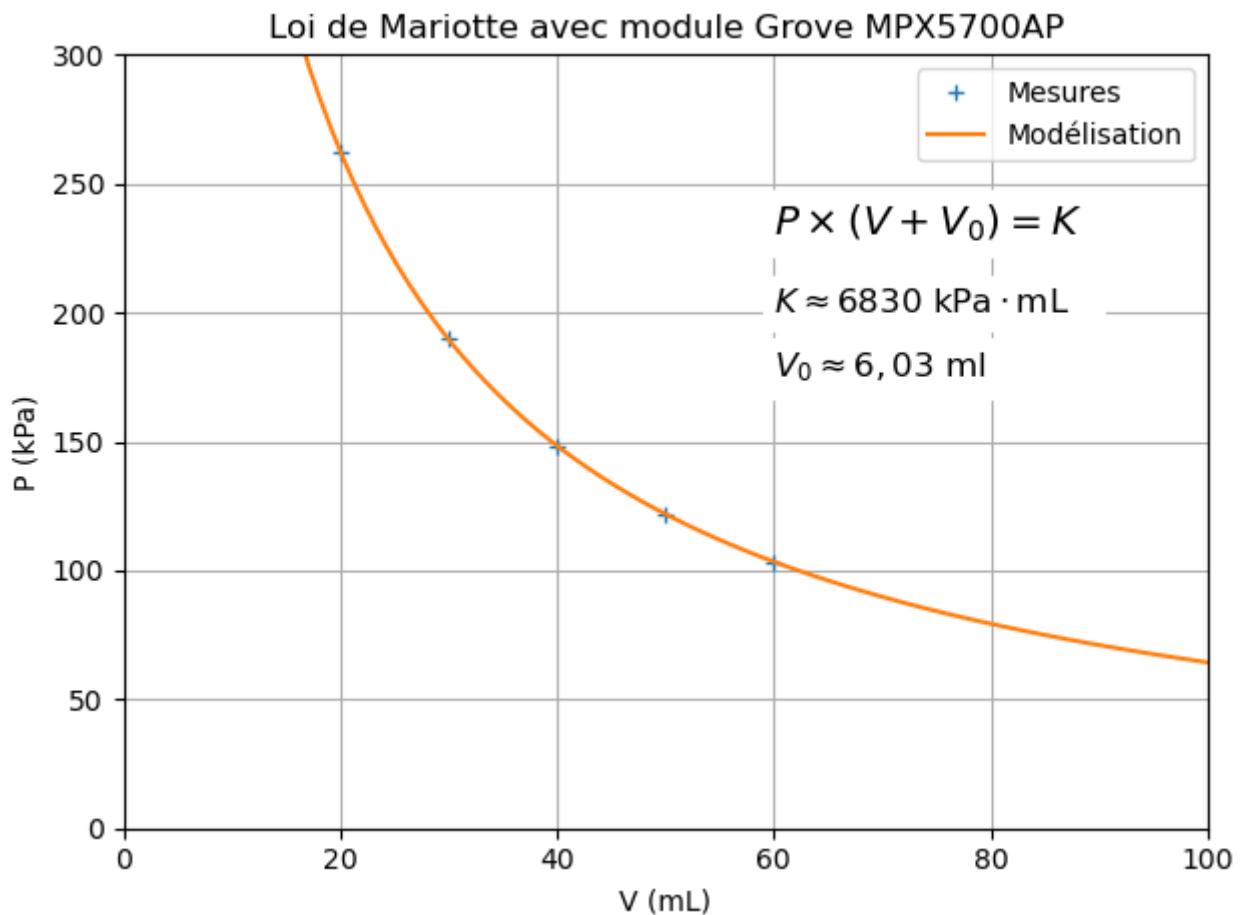
```



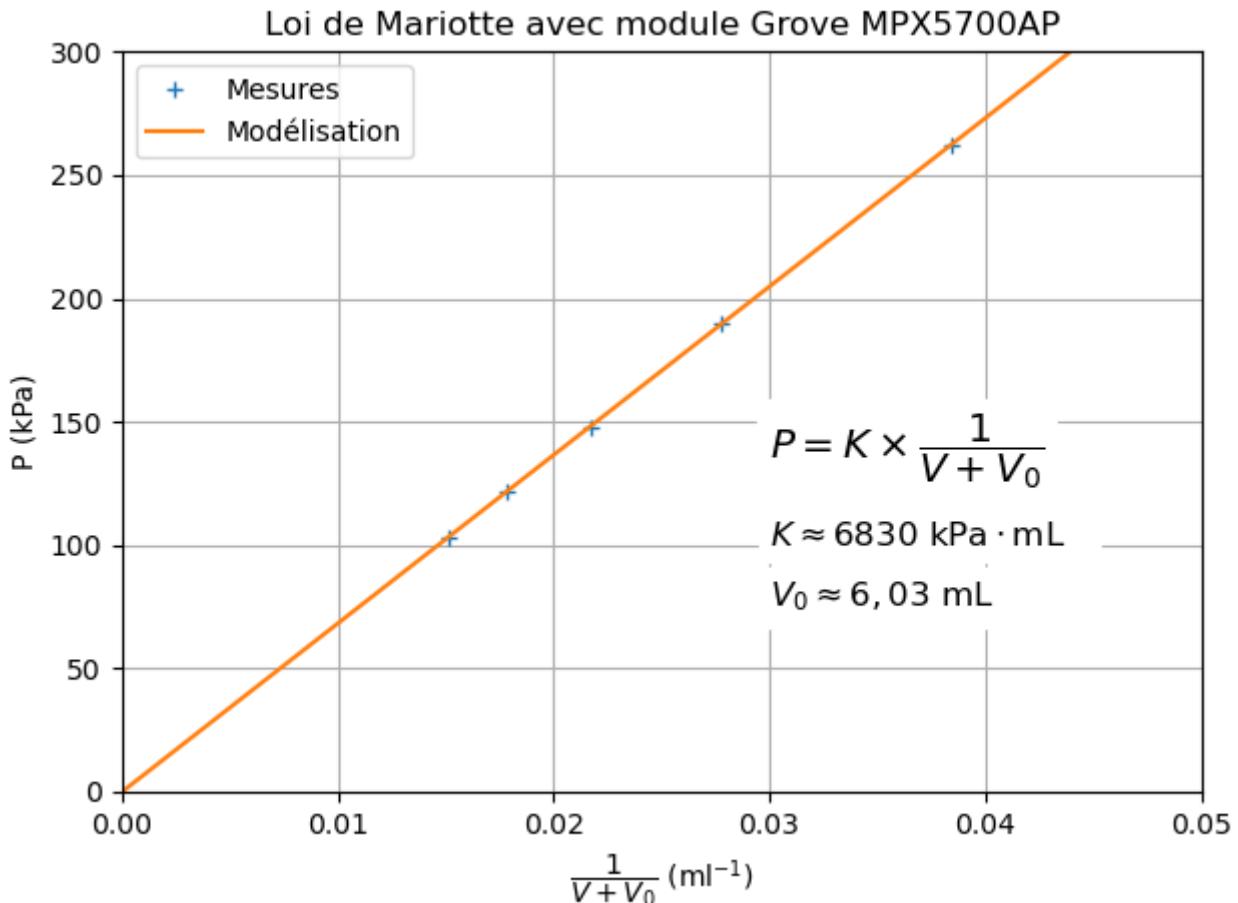
FIG. 24 – Résultats obtenus dans le moniteur série d’Arduino IDE

5.4.5 Résultats

V (mL)	60	50	40	30	20
P (kPa)	103	122	148	190	262



V_0 est le volume additionnel contenu dans le tube et le capteur.



5.5 Mesurer une pression - Loi de la statique des fluides (première générale)

Programme de première générale 2019 - Enseignement de spécialité.

Tester la loi fondamentale de la statique des fluides.

5.5.1 Principe

Vérifier expérimentalement la loi de la statique des fluides à l'aide d'un capteur de pression différentielle.

$$P_B - P_A = \rho \times g \times (z_A - z_B)$$

5.5.2 Capteur Plug'Uino (XGZP6857A)

Le capteur Plug'Uino (pression pour statistique des fluides) comporte un capteur de pression différentielle (manomètre) de référence XGZP6857A005KPG.



FIG. 25 – Capteur de pression différentielle XGZP6857A (manomètre)

Ce capteur délivre une tension analogique de 0,5 V à 4,5 V pour pression différentielle comprise entre 0 et 5 kPa.



FIG. 26 – Kit Plug'Uino compatible Grove avec capteur XGZP6857A (image : Sciencéthic)

Item	Data	Unit
Power Supply	5(or 3.3 or 3)	V
Max. Excitation current	3	mA
Output Signal	0.5-4.5(or 0.2-2.7 or customized)	V
Accuracy($\geq 20\text{kPa}$)	$\pm 1.0(\text{or } \pm 0.5)$	%Span
TCO(Temp. Coefficient of Offset)	± 0.03	%FS/ $^{\circ}\text{C}$
TCS(Temp. Coefficient of Span)	± 0.03	%FS/ $^{\circ}\text{C}$
Long Term Stability(1year)	± 0.5	%Span
Over Pressure	2X	Rated
Compensation Temp.	$0 \sim 60/32 \sim 140$	$^{\circ}\text{C}/^{\circ}\text{F}$
Ambient Temp.	$-10 \sim 85/14 \sim 176$	$^{\circ}\text{C}/^{\circ}\text{F}$
Storage Temp.	$-40 \sim 125/-40 \sim 257$	$^{\circ}\text{C}/^{\circ}\text{F}$

FIG. 27 – Extrait de la documentation technique du capteur XGZP6857A

Le calcul de la pression obtient alors par l'expression suivante :

$$P = \frac{P_{max} - P_{min}}{V_{max} - V_{min}} \times (v_{out} - V_{min}) + P_{min}$$

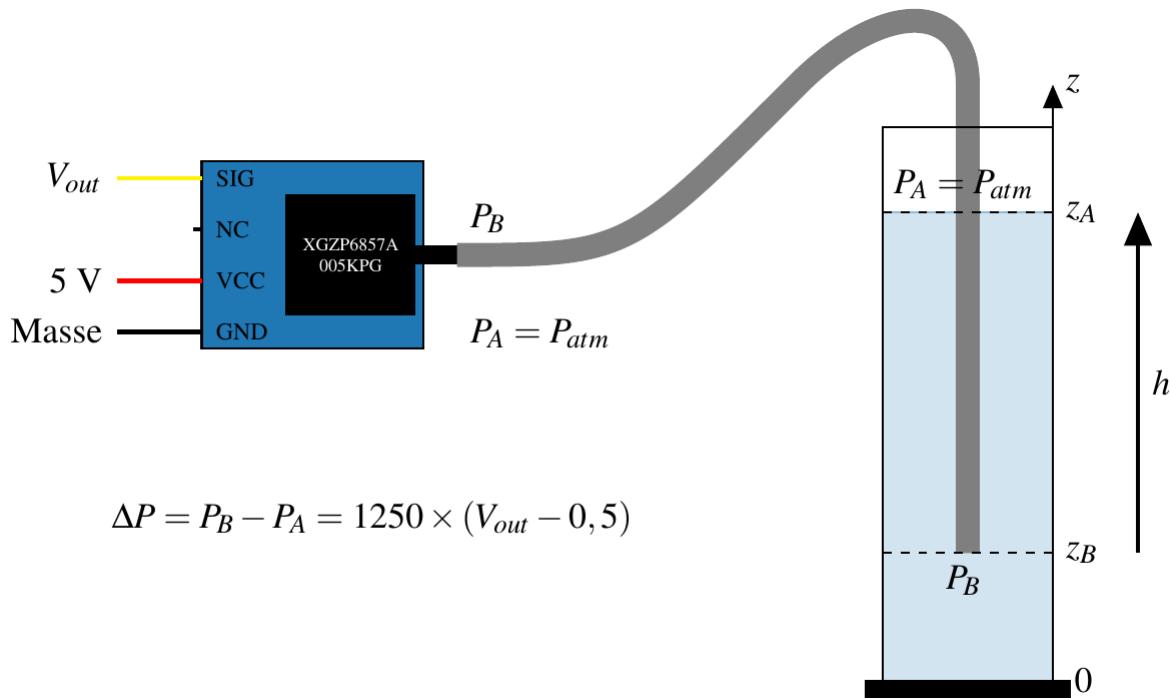
Soit :

$$P = \frac{5000 - 0}{4,5 - 0,5} \times (v_{out} - 0,5) + 0 \quad \Rightarrow \quad P \approx 1250 \times (v_{out} - 0,5)$$

La précision du capteur est de :

$$\pm 0,5\% \times 5000 = \pm 25 \text{ Pa}$$

5.5.3 Montage



5.5.4 Programme Arduino

```
/*
 * Capteur de pression différentielle Plug 'Uino
 * 0 - 5 kPa -> 0,5 - 4,5 V (capteur XGZP6857A005KPG)
 * David THERINCOURT - 2025
 */

void setup() {
    Serial.begin(9600);           // Initialisation du port série
}

void loop() {
    int N = analogRead(A0);      // Lecture sur A0
}
```

(suite sur la page suivante)

(suite de la page précédente)

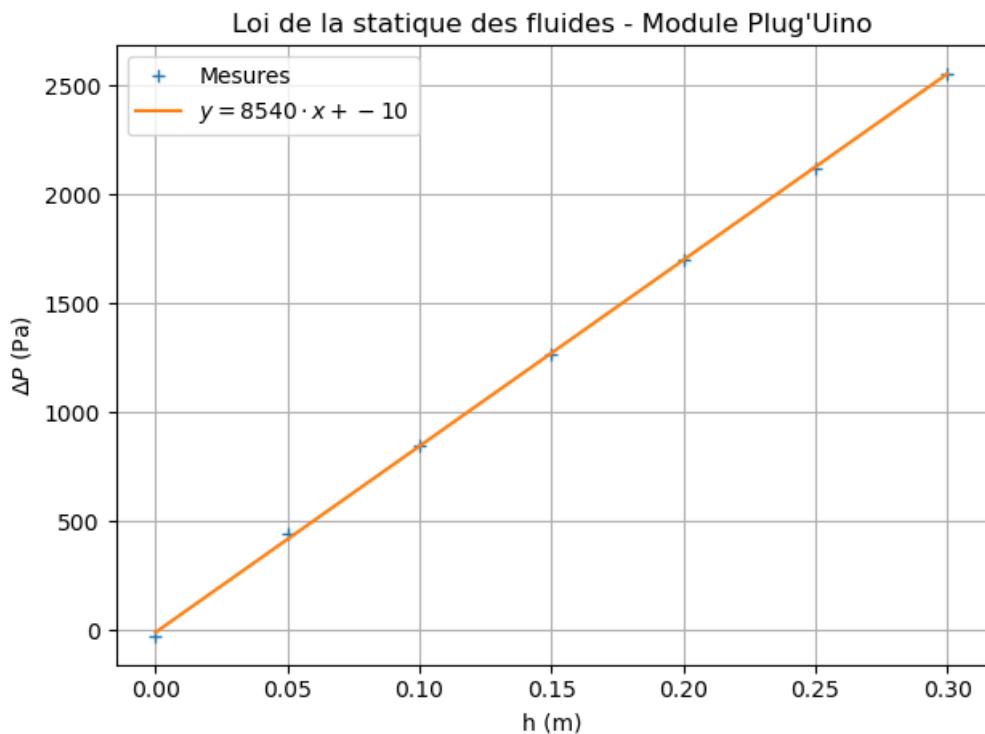
```

float u = N * (5.0 / 1023.0); // Calcul de la tension en volt
float P = 1250*(u-0.5);      // Calcul de la pression en Pa
Serial.print("U = ");        // Affichage
Serial.print(u);              // ...
Serial.print(" V \t");        // ... \t pour un tabulation
Serial.print(P);              // ...
Serial.println(" Pa");        // ...
delay(1000);                 // Temporisation
}

```



FIG. 28 – Résultats dans le moniteur série d'Arduino IDE



5.6 Géométrie d'un condensateur (terminale générale)

Programme de terminale générale 2019 - Enseignement de spécialité.

Identifier et tester le comportement capacitif d'un dipôle. Illustrer qualitativement, par exemple à l'aide d'un microcontrôleur, d'un multimètre ou d'une carte d'acquisition, l'**effet de la géométrie d'un condensateur sur la valeur de sa capacité**.

5.6.1 Principe**5.6.2 Montage****5.6.3 Programme****5.6.4 A retenir****5.7 Capteur capacitif (terminale générale)****Programme de terminale générale 2019 - Enseignement de spécialité.**

Expliquer le principe de fonctionnement de quelques capteurs capacitifs. Étudier la réponse d'un dispositif modélisé par un dipôle RC. Déterminer le temps caractéristique d'un dipôle RC à l'aide d'un microcontrôleur, d'une carte d'acquisition ou d'un oscilloscope.

5.7.1 Principe

Soit le circuit RC série suivant :

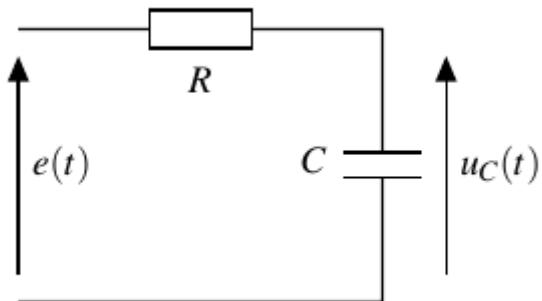


FIG. 29 – Schéma électrique

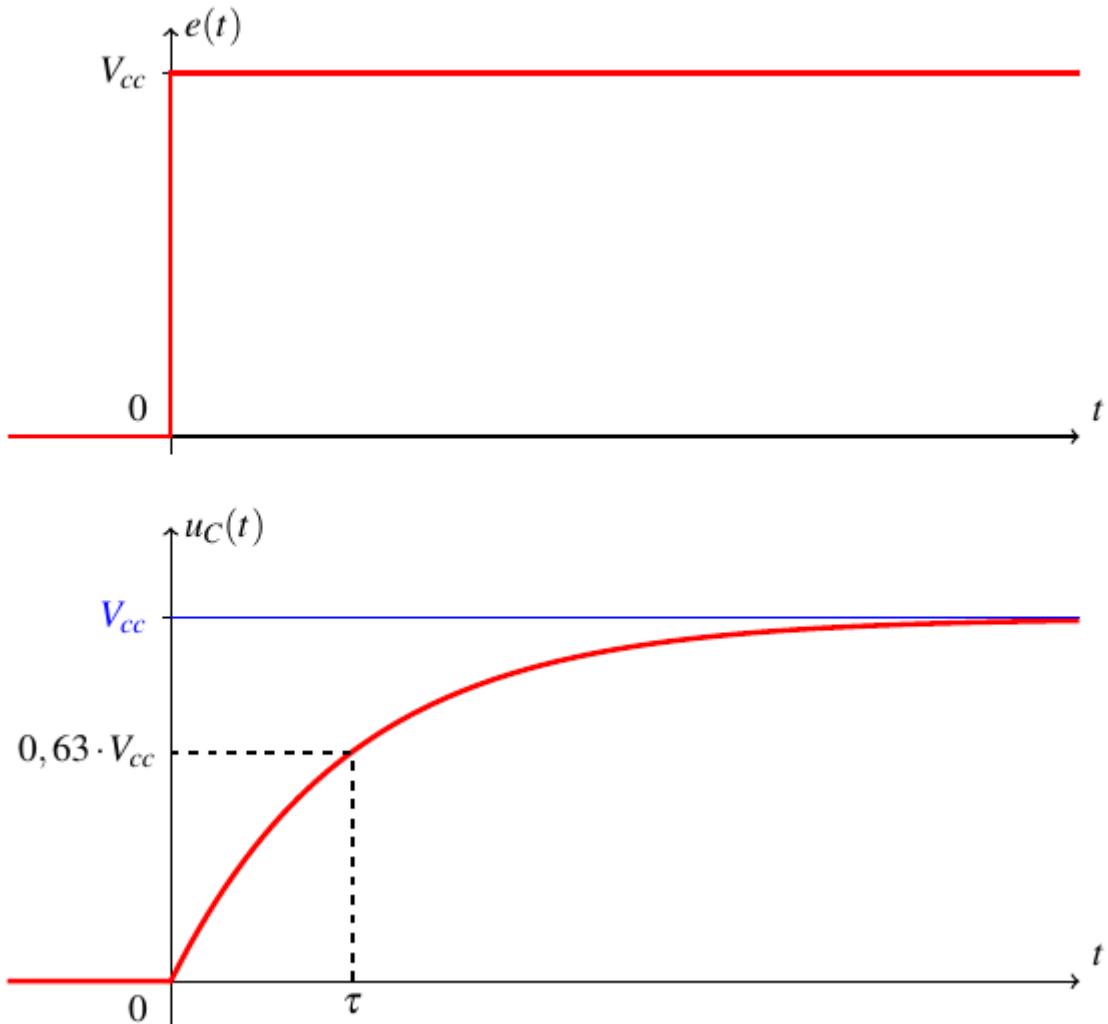


FIG. 30 – Évolution de la tension du condensateur en fonction du temps

Lors de la charge du condensateur C à travers la résistance R sous tension constante V_{cc} , le **temps caractéristique** (ou constante de temps) τ est la durée que prend la tension u_C pour atteindre 63% de sa valeur finale V_{cc} .

Avec un microcontrôleur, il est assez facile de mesurer ce temps caractéristique par une **mesure de durée** jusqu'à la **détection du seuil de 63%** de la valeur finale de la tension du condensateur.

5.7.2 Montage

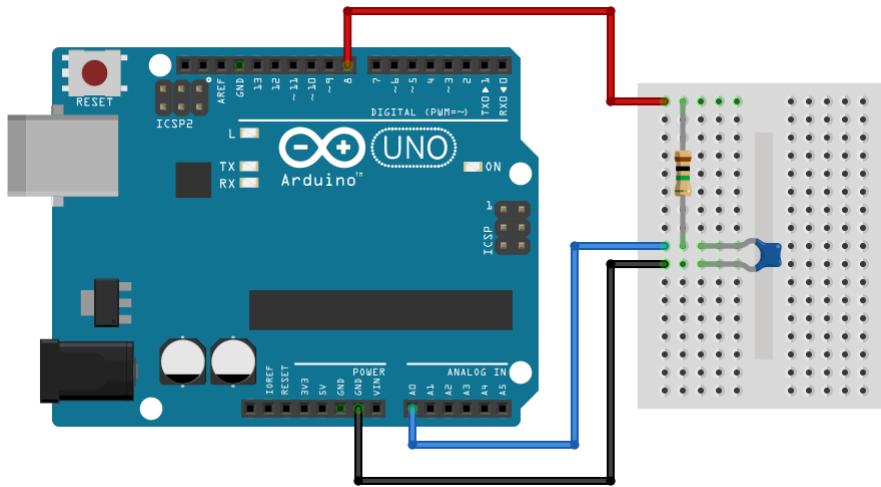


FIG. 31 – Charge d'un condensateur sous tension constante avec un Arduino UNO R3

- La broche digitale 8 charge ou décharge le condensateur à travers la résistance.
- A0 mesure la tension aux bornes du condensateur.

Données :

$$R = 1 \text{ M}\Omega \quad C = 22 \text{ nF}$$

5.7.3 Programme Arduino

```
/*
 * Mesure de constante de temps d'un circuit RC
 */

#define pinE 8

int N = 0;
unsigned long t0;
unsigned long t1;
unsigned long tau;
float C;

void setup() {
    Serial.begin(9600);
    Serial.println("Start");
    pinMode(pinE,OUTPUT); // Broche digitale en sortie

    digitalWrite(pinE,LOW); // Décharge condensateur avant mesure
    delay(1000); // pendant 1 s

    digitalWrite(pinE,HIGH); // Début charge condensateur
    t0 = micros(); // Mesure instant initial

    while (N<646) { // Boucle tant que tension inférieure à seuil (0,
        -632*1023=646)

```

(suite sur la page suivante)

(suite de la page précédente)

```

N=analogRead(A0);           // Lecture tension condensateur
}

t1 = micros();              // Mesure instant où seuil atteint
digitalWrite(pinE,LOW);     // Début décharge condensateur
tau = t1 - t0;              // Calcul de tau

Serial.print(tau);          // Début affichage
Serial.println(" µs");
}

void loop() {
    // Boucle sans fin pas utilisée ici !
}

```

5.7.4 Application : mesure d'une capacité

Sachant que le temps caractéristique est défini par la relation :

$$\tau = R \cdot C$$

Le calcul de la capacité C du condensateur est :

$$C = \frac{\tau}{R}$$

Il suffit donc d'ajouter cette relation dans le code précédent !

```

/*
 * Mesure de la capacité d'un circuit RC
 */

#define pinE 8

float R = 1000;           // Resistance en kOhm
int N = 0;
unsigned long t0;
unsigned long t1;
unsigned long tau;
float C;

void setup() {
    Serial.begin(9600);
    Serial.println("Start");
    pinMode(pinE,OUTPUT);      // Broche digitale en sortie

    digitalWrite(pinE,LOW);    // Décharge condensateur avant mesure
    delay(1000);              // pendant 1 s

    digitalWrite(pinE,HIGH);   // Début charge condensateur
    t0 = micros();             // Mesure instant initial

    while (N<646) {           // Boucle tant que tension inférieure à seuil (0,
        N=analogRead(A0);      // Lecture tension condensateur
    }
}

```

(suite sur la page suivante)

(suite de la page précédente)

```
}

t1 = micros();           // Mesure instant où seuil atteint
digitalWrite(pinE,LOW); // Début décharge condensateur

tau = t1 - t0;          // Calcul de tau
C = tau/R;              // Calcul de C en nF

Serial.print(C);         // Début affichage
Serial.println(" nF");   // Fin affichage
}

void loop() {
    // Boucle sans fin pas utilisée ici !
}
```

5.7.5 A retenir

- La fonction `micros()` renvoie la durée en μs (< 70 min) depuis que la carte Arduino a été mise sous tension. La précision est de $4 \mu\text{s}$!
- La boucle `while` (tant que) associée à la fonction `analogRead()` détecte le seuil de la tension du condensateur.

Chapitre 6

Aller plus loin

6.1 Contrôler l'intensité d'une LED avec un potentiomètre

6.1.1 Montage

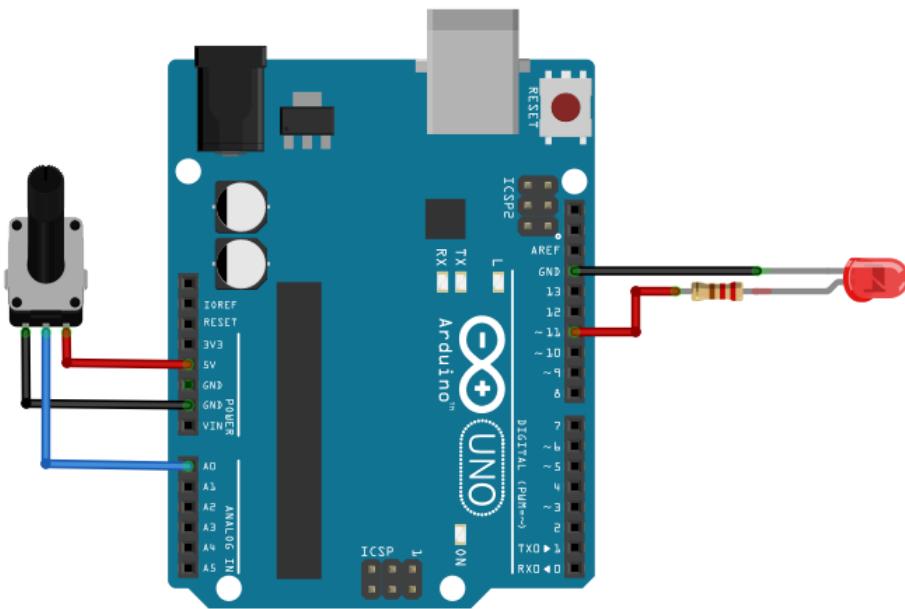


FIG. 1 – Commande de l'intensité d'une LED avec un potentiomètre

6.1.2 Programme

```
#define pinLED 11

int N;          // Valeur lue sur A0 de 0 à 1023
int duty;       // Rapport cyclique de 0 à 255

void setup() {
}

void loop() {
    N = analogRead(A0);           // Conversion analogique-numérique sur A0
```

(suite sur la page suivante)

(suite de la page précédente)

```
duty = N/4;           // Calcul du rapport cyclique
analogWrite(pinLED, duty); // Génération de la tension PWM
delay(30);           // Attendre 30 ms
}
```

Note

Pour convertir un entier sur 10 bits en un entier sur 8 bits, il suffit de la **division entière par 4 !**

6.2 Arduino IDE - Saisir une valeur au clavier

6.2.1 Montage

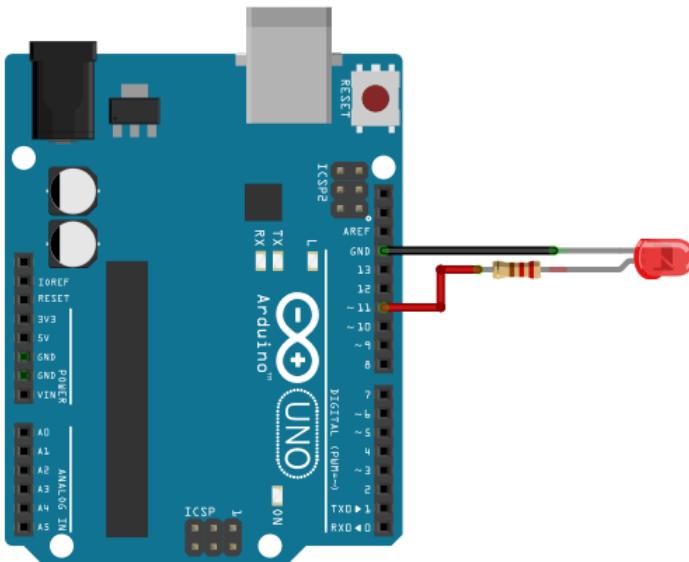


FIG. 2 – Branchement d'une LED sur la broche 11

Une LED en série avec une résistance de $220\ \Omega$ est branchée sur la broche 11.

6.2.2 Programme

Le programme ci-dessous contrôle l'intensité de la LED interne d'un Arduino avec le clavier.

- Le rapport cyclique (entier entre 0 et 255) est saisie au clavier dans le moniteur série avec la fonction `parseInt()` de la librairie `Serial`.
- Un signal PWM est généré sur la broche 11 (LED interne) avec la fonction `analogWrite()`.

```
// PWM avec saisie du rapport cyclique (entier de 0 à 255) au clavier dans le moniteur série.
// ATTENTION : Sélectionner "Pas de fin de ligne" dans le monitor série !!!
// David THERINCOURT 2025

#define LED 11          // LED connectée à la broche 11
```

(suite sur la page suivante)

(suite de la page précédente)

```

void setup()
{
    Serial.begin(9600);      // Initialisation du port série
    pinMode(LED, OUTPUT);   // Configuration de la broche LED en sortie
}

void loop()
{
    Serial.print("Rapport cyclique entre 0 et 255 : "); // Indication à l'utilisateur
    while (Serial.available()==0){}                      // Attente d'un message (Cocher
    ↵"Pas de fin de ligne")
    int N = Serial.parseInt();                          // Extraction de la valeur ↵
    ↵numérique (entier)
    Serial.println(N);                                // Affichage de N
    analogWrite(LED, N);                            // Ecriture sur la sortie PWM
}

```

6.3 Grove - Afficheur LCD 16x2 sur port I2C

6.3.1 Principe

Il s'agit d'un afficheur de 32 caractères disposés sur 2 lignes (16 caractères par ligne).

La communication avec le microcontrôleur s'effectue via un port série de données (I2C) nécessitant moins de câbles (4 en tout).

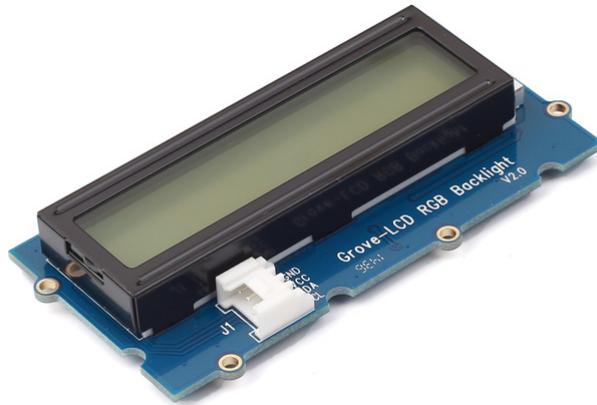


FIG. 3 – Module Grove - LCD RGB Backlight (image : <http://wiki.seeedstudio.com>)

Chaque afficheur utilise sa propre librairie (ex. `rgb_lcd.h` pour le Grove LCD RGB Backlight) en plus de la librairie `wire.h` qui est obligatoire pour la gestion du port I2C.

6.3.2 Grove LCD RGB Backlight



FIG. 4 – Modules Grove (image : <http://wiki.seeedstudio.com>)

Télécharger [ici](#) le fichier `Grove_LCD_RGB_Backlight-master.zip` pour l'installation de la librairie `rgb_lcd` avant la compilation du programme.

```
/*
 * Exemple affichage sur LCD 2x16 RGB I2C Grove
 */

#include <Wire.h>           // Importation librairie gestion port I2C
#include "rgb_lcd.h"         // Importation librairie gestion afficheur LCD I2C Grove

rgb_lcd lcd;                // Déclaration de l'afficheur

const int colorR = 255; // Couleur fond écran
const int colorG = 0;
const int colorB = 0;

void setup()
{
    lcd.begin(16, 2);          // Fixe 2 colonnes et 16 caractères/ligne
    lcd.setRGB(colorR, colorG, colorB); // Fixe couleur de fond
    lcd.print("hello, world!"); // Affiche texte
    delay(1000);              // Temporisation 1s
}

void loop()
{
    lcd.setCursor(0, 1);        // Déplace le curseur
    lcd.print(millis()/1000); // Affiche le temps écoulé en s (timer
    delay(100);                // Temporisation 1s
}
```

6.3.3 Grove LCD 16x2 (rouge/jaune/bleu)

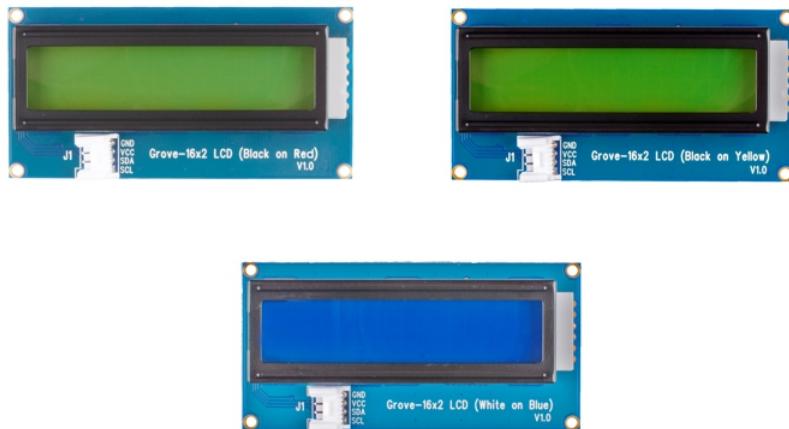


FIG. 5 – Grove LCD 16x12 (image : <http://wiki.seeedstudio.com>)

Toujours avec la librairie précédente `rgb_lcd`.

```
/*
 * Exemple affichage sur LCD 2x16 I2C Grove sans RGB
 */

#include <Wire.h>
#include "rgb_lcd.h"

rgb_lcd lcd;           // Déclaration de l'afficheur LCD branché sur un port I2C

void setup()
{
    lcd.begin(16, 2); // Initialisation de l'afficheur LCD sur 2 lignes à 16 caractères
}

void loop()
{
    lcd.setCursor(0, 0);      // Placement du curseur
    lcd.print("Bonjour !");  // Affichage de la valeur de la tension
    delay(1000);             // Pause de 1000 ms
}
```

6.4 Educaduino Lab - Afficheur LCD

6.4.1 Principe

Le pilotage d'un afficheur LCD 16x2 nécessite de 6 broches numériques. Le brochage dépend de la référence de l'afficheur.

Pour éviter un câblage trop complexe, le plus simple est de fixer sur la carte de développement un « shield » afficheur comme celui proposé par Educaduino LAB.

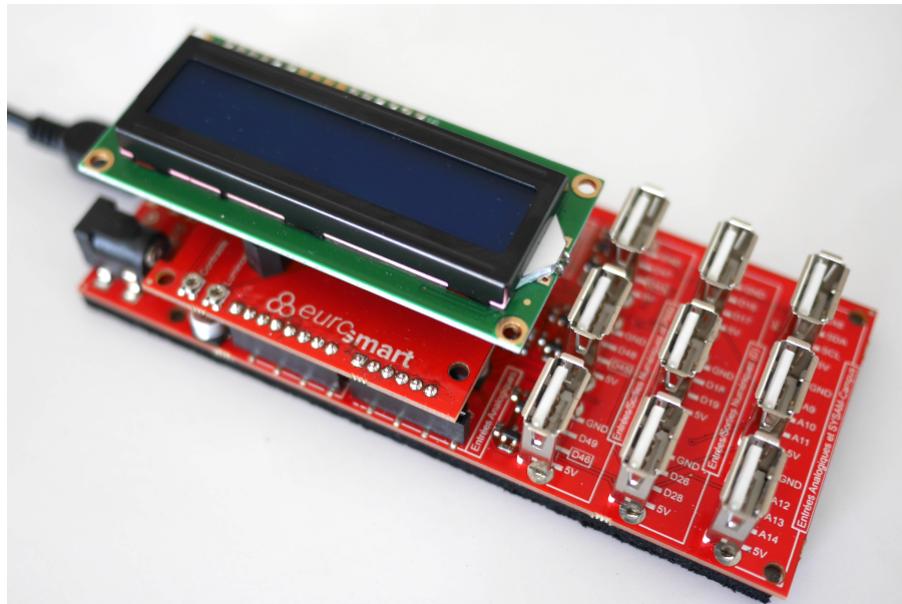


FIG. 6 – Afficheur LCD 16x2 d’Educaduino-Lab

6.4.2 Programme

La librairie `LiquidCrystal` est installée par défaut dans Arduino IDE.

```
/*
 * Exemple d'utilisation d'un écran LCD 16x2 parallèle
 */

#include <LiquidCrystal.h>           // Importation de la librairie LiquidCrystal

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Brochage de l'afficheur

void setup() {
    lcd.begin(16, 2);                // fixe le nombre de colonnes et de lignes de l'afficheur
}

void loop() {
    lcd.setCursor(5,0);             // place le curseur à la colonne 5 et à la ligne 0
    lcd.print("Bonjour");           // Affiche un texte
    lcd.setCursor(0,1);             // place le curseur à la colonne 0 et à la ligne 1
    lcd.print("tout le monde !");   // Affiche un autre texte
}
```

Note

Il sera peut-être nécessaire d'installer la librairie `LiquidCrystal` dans le logiciel Arduino.

6.4.2.1 En résumé

Instruction	Description
#include <LiquidCrystal.h>	Importe la librairie de gestion de l'afficheur LCD
LiquidCrystal lcd(12, 11, 5, 4, 3, 2)	Déclare l'afficheur en précisant les numéros de broches
lcd.begin(16, 2)	Fixe le nombre de colonnes et de lignes de l'afficheur
lcd.setCursor(col,line)	Positionne le curseur
lcd.print(variable)	Affiche le contenu d'une variable à la position du curseur

6.5 Educaduino Lab - Loi de Mariotte

6.5.1 Principe

La manipulation consiste à vérifier la loi de Mariotte (à température et à quantité de matière constantes) :

$$P \times V = \text{Constante}$$

6.5.2 Capteur MPXHZ6400A

Le MPXHZ6400A est un capteur analogique de pression absolue (20 kPa à 400 kPa - 5V).



FIG. 7 – Capteur MPX6400A Educaduino LAB

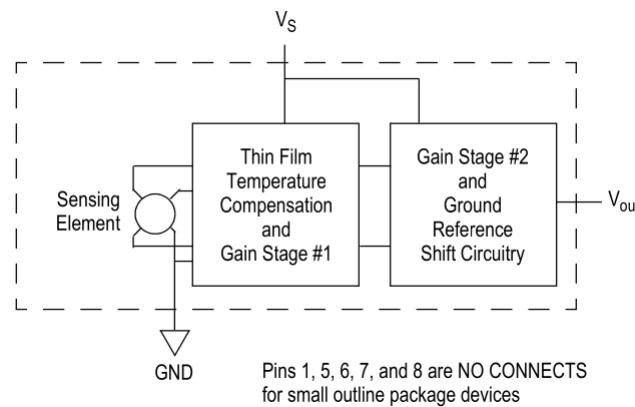


FIG. 8 – Schéma électrique (source : Freescale Semiconductor, Inc.)

Table 2. Operating Characteristics ($V_S = 5.0$ Vdc, $T_A = 25^\circ\text{C}$.)

Characteristic	Symbol	Min	Typ	Max	Unit
Pressure Range	P_{OP}	20	—	400	kPa
Supply Voltage ⁽¹⁾	V_S	4.64	5.0	5.36	Vdc
Supply Current	I_o	—	6.0	10	mAdc
Minimum Pressure Offset @ $V_S = 5.0$ Volts ⁽²⁾	V_{off}	0.133	0.2	0.267	Vdc
Full Scale Output @ $V_S = 5.0$ Volts ⁽³⁾	V_{FSO}	4.733	4.8	4.866	Vdc
Full Scale Span @ $V_S = 5.0$ Volts ⁽⁴⁾	V_{FSS}	4.467	4.6	4.733	Vdc
Accuracy ⁽⁵⁾ (0 to 85°C)	—	—	—	± 1.5	% V_{FSS}
Sensitivity	V/P	—	12.1	—	mV/kPa
Response Time ⁽⁶⁾	t_R	—	1.0	—	ms
Warm-Up Time ⁽⁷⁾	—	—	20	—	ms
Offset Stability ⁽⁸⁾	—	—	± 0.25	—	% V_{FSS}

FIG. 9 – Caractéristiques (source : Freescale Semiconductor, Inc.)

6.5.3 Calcul de la pression

La mesure de pression (en kPa) est donnée par la relation pour une alimentation de 5 V :

$$P = \frac{P_{max} - P_{min}}{V_{max} - V_{min}} \times (v_{out} - V_{min}) + P_{min}$$

Soit :

$$P = \frac{400 - 20}{4,8 - 0,2} \times (v_{out} - 0,2) + 20 \implies P \approx 82,6 \times (v_{out} - 0,2) + 20$$

Note

Dans la bibliothèque Educaduino, l'expression de la pression n'est pas en accord avec la documentation technique !

$$P = \frac{400 - 20}{5 - 0} \times v_{out} + 20 \implies P \approx 76 \times v_{out} + 20 \quad (\text{kPa})$$

6.5.4 Montage

Le montage est composé d'une carte Educaduino Lab, d'un capteur de pression Educaduino (MPXHZ6400A / 20 kPa à 400 kPa) et d'une seringue.

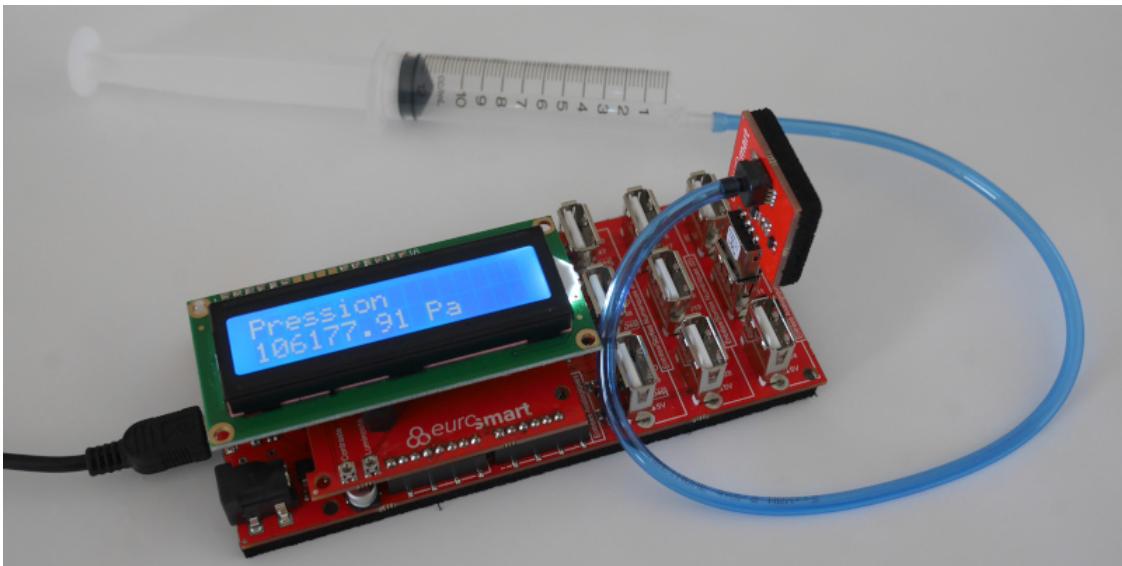


FIG. 10 – Mesure d'une pression avec Educaduino Lab

6.5.5 Programme Arduino

```

1  /*
2   * Mesure d'une pression absolue
3   * Capteur Educaduino 20 kPa à 400 kPa
4   * branché sur la broche A9
5   */
6
7 #define brocheCapteur A9      // Numéro de broche connectée au capteur
8 #include <LiquidCrystal.h>    // Librairie de gestion de l'écran LCD
9
10 LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Déclaration de l'écran LCD
11
12
13 float tension ;           // Tension mesurée
14 float pression ;          // Pression mesurée
15
16 void setup() {
17     lcd.begin(16, 2);       // Paramétrage de l'écran LCD
18 }
19
20

```

(suite sur la page suivante)

(suite de la page précédente)

```

21 void loop() {
22     tension = analogRead(brocheCapteur)*5.0/1023 ;      // Lecture de la tension
23     pression = 82.6*(tension-0.2) + 20 ;                // Calcul de la pression en kPa
24     lcd.clear();                                         // Début affichage
25     lcd.setCursor(0,0);
26     lcd.print("Pression en kPa");
27     lcd.setCursor(0,1);
28     lcd.print(pression);                                // Fin affichage
29     delay(1000);
30 }
```

6.5.6 Résultats

V (mL)	12	11	10	9	8	7	6	5	4
P (kPa)	96,5	105	114	124	136	152	171	194	227

Ces mesures ne tiennent pas compte du volume additionnel dans le tube et le capteur !

6.6 Educaduino Lab - Loi de la statique des fluides

6.6.1 Principe

Vérifier expérimentalement la loi de la statique des fluides à l'aide d'un capteur de pression différentielle.

$$P_B - P_A = \rho \times g \times (z_A - z_B)$$

6.6.2 Capteur MPX2010DP/GP

La mesure de pression s'effectue avec un capteur de pression différentiel du type MPX2010 (0 à 100 hPa) compensé en température.

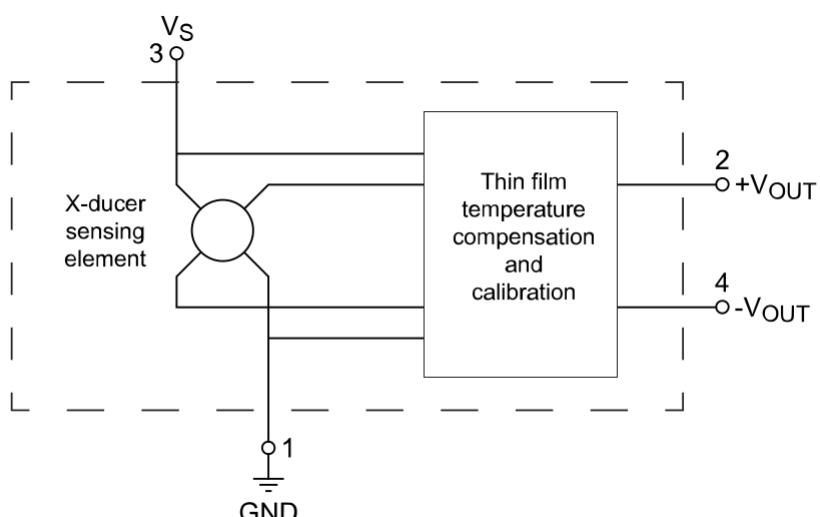


FIG. 11 – Schéma électrique du MPX2010 (image : NXP Semiconductors)

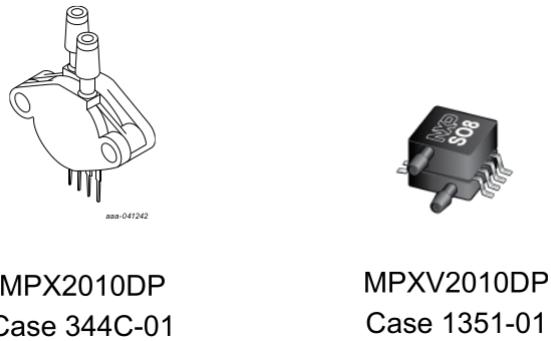


FIG. 12 – (image : NXP Semiconductors)

La **version DP** (pression différentielle) mesure la différence de pression entre deux points d'entrée.

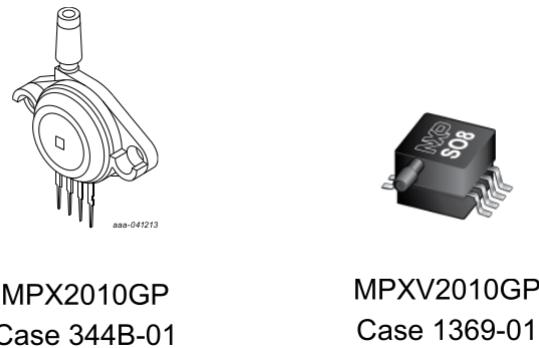


FIG. 13 – (image : NXP Semiconductors)

La **version GP** (pression relative - manomètre) mesure la pression d'un point d'entrée par rapport à une pression de référence (pression atmosphérique).

Table 9. Operating characteristics ($V_S = 10$ Vdc, $T_A = 25$ °C unless otherwise noted, $P1 > P2$)

Characteristic	Symbol	Min	Typ	Max	Units
Operating Pressure Range [1]	P_{OP}	0	—	10	kPa
Supply Voltage [2]	V_S	—	10	16	Vdc
Supply Current	I_o	—	6.0	—	mAdc
Full Scale Span [3]	V_{FSS}	24	25	26	mV
Offset [4]	V_{off}	-1.0	—	1.0	mV
Sensitivity	$\Delta V/\Delta P$	—	2.5	—	mV/kPa
Linearity [5]	—	-1.0	—	1.0	% V_{FSS}
Pressure Hysteresis (0 kPa to 10 kPa) [5]	—	—	±0.1	—	% V_{FSS}
Temperature Hysteresis (-40 °C to +125 °C) [5]	—	—	±0.5	—	% V_{FSS}
Temperature Coefficient of Full Scale Span [5]	TCV_{FSS}	-1.0	—	1.0	% V_{FSS}
Temperature Coefficient of Offset [5]	TCV_{off}	-1.0	—	1.0	mV
Input Impedance	Z_{in}	1300	—	2550	Ω
Output Impedance	Z_{out}	1400	—	3000	Ω
Response Time (10% to 90%) [6]	t_R	—	1.0	—	ms
Warm-Up Time [7]	—	—	20	—	ms
Offset Stability [8]	—	—	±0.5	—	% V_{FSS}

FIG. 14 – Extrait datasheet (source : NXP Semiconductors)

Pour ce capteur, la tension de sortie est proportionnelle à la différence de pression mesurée telle que :

$$U_{out} = S \times \Delta P \quad \text{avec} \quad S = 2,5 \text{ mV/kPa}$$

6.6.3 Capteur Educaduino Lab (MPX2010GP)

Le module Educaduino LAB est conçu autour sur le capteur MPX2010GP.



FIG. 15 – Capteur de pression différentielle Educaduino LAB (jauge)

Après adaptation (amplification), l'expression de la pression différentielle (en Pa) en fonction de la tension en sortie du

capteur est :

$$\Delta P = 2000 \times U \quad (\text{Pa})$$

6.6.4 Montage

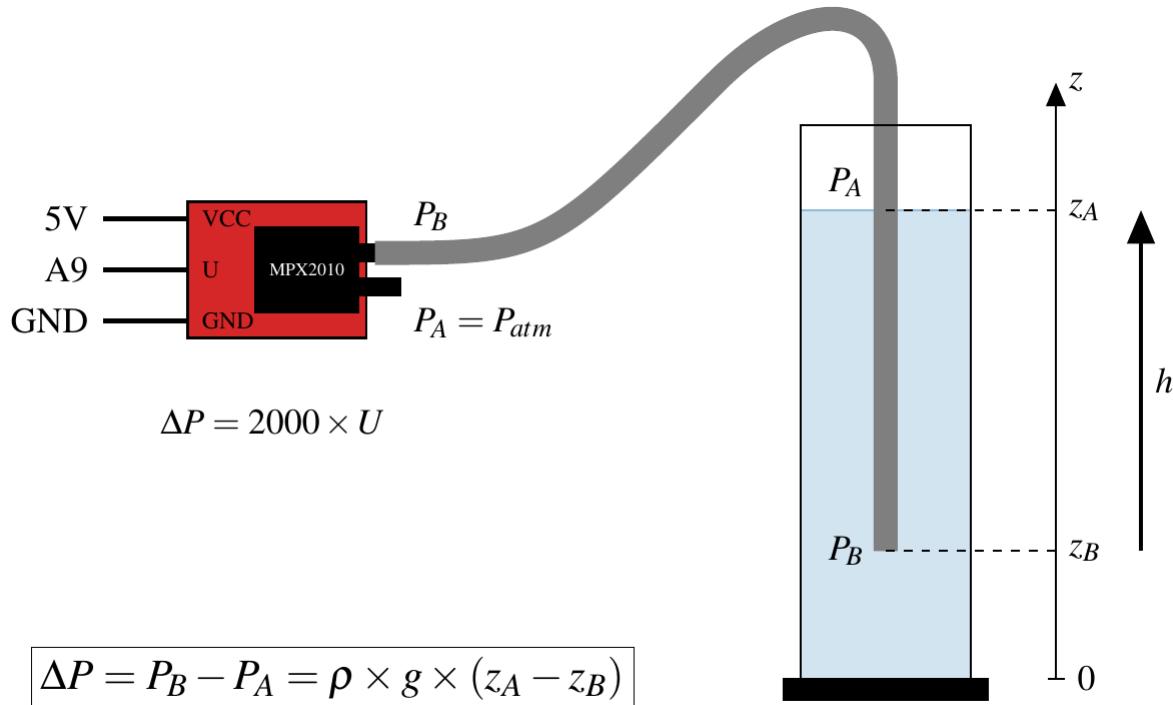


FIG. 16 – Montage de la vérification de loi de la statique des fluides à partir d'un Arduino

6.6.5 Programme Arduino

Avec écran LCD Educaduino LAB. La lecture de la tension analogique se fait sur la broche A9.

```
/*
 * Mesure d'une pression relative
 * Capteur Educaduino MPX2010GP 0 à 10 kPa
 * branché sur la broche A9
 */

#define brocheCapteur A9           // Numéro de broche connectée au capteur
#include <LiquidCrystal.h>        // Librairie de gestion de l'écran LCD

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Déclaration de l'écran LCD

int n;
float tension;                      // Tension mesurée
int pression;                       // Pression mesurée

void setup() {
    lcd.begin(16, 2);               // Paramétrage de l'écran LCD
```

(suite sur la page suivante)

(suite de la page précédente)

```
}

void loop() {
    n = analogRead(brocheCapteur) ;      // Lecture de la tension
    tension = n*5.0/1023 ;               // Lecture de la tension
    pression = round(tension*2000) ;     // Calcul de la pression en Pa
    lcd.clear();                        // Début affichage
    lcd.setCursor(0,0);
    lcd.print("N");
    lcd.setCursor(6,0);
    lcd.print("P (Pa)");
    lcd.setCursor(0,1);
    lcd.print(n);                      // Fin affichage
    lcd.setCursor(6,1);
    lcd.print(round(pression));        // Fin affichage
    delay(1000);
}
```

Chapitre 7

Cartes Micro :bit (langage Micropython)

7.1 Cartes Micro :bit



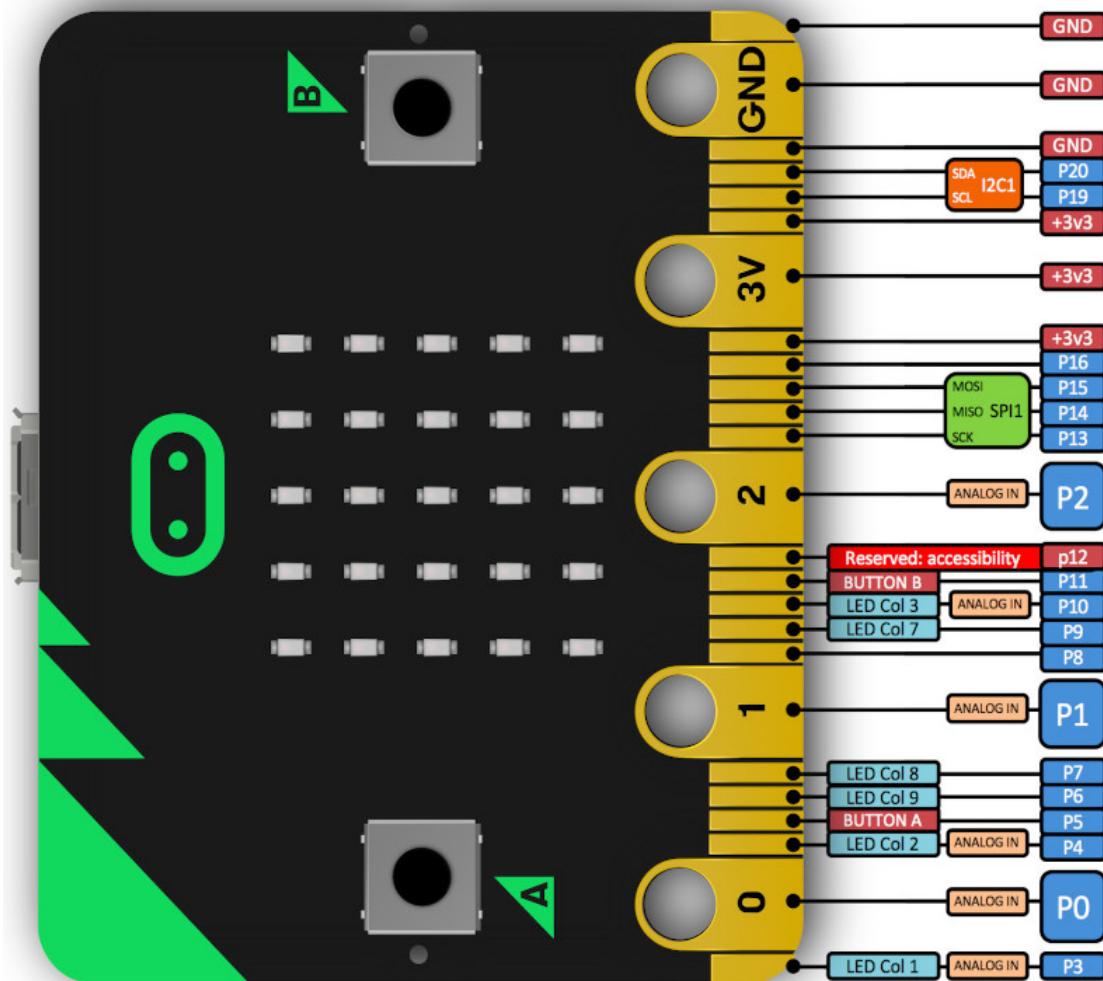
La carte Micro :bit a été développée par la BBC pour l'enseignement de l'informatique dans les écoles du Royaume Uni. Elle est programmable en MakeCode ou en MicroPython.

Principales caractéristiques de la carte :

- Microcontrôleur nRF51822 (ARM Cortex M0 - 32 bit - 16 Mhz) ;
- 256 ko de mémoire flash (ROM) ;
- 16 ko de mémoire vive (RAM) ;
- Tension de fonctionnement à 3,3V;
- 1 port micro USB (programmation REPL + accès mémoire flash) ;
- 25 ports E/S ;
- 3 entrées analogique ;
- 2 boutons, matrice 5x5 Leds, 1 ;
- Bluetooth 4.0 LE
- Magnétomètre 3D, accéléromètre 3D

Brochage :

Le brochage est assez particulier. Cinq anneaux (grosses broches) donnent accès à 3 entrées/sorties (P0, P1 et P2) et à l'alimentation. Les autres ports sont accessibles sur des petites broches.



7.2 Langage BBC MicroPython



7.2.1 Edition

BBC micro:bit MicroPython est une édition de MicroPython spécialement conçue pour Micro :bit.

Pour son développement, il est conseillé d'utiliser l'éditeur [Mu](#).

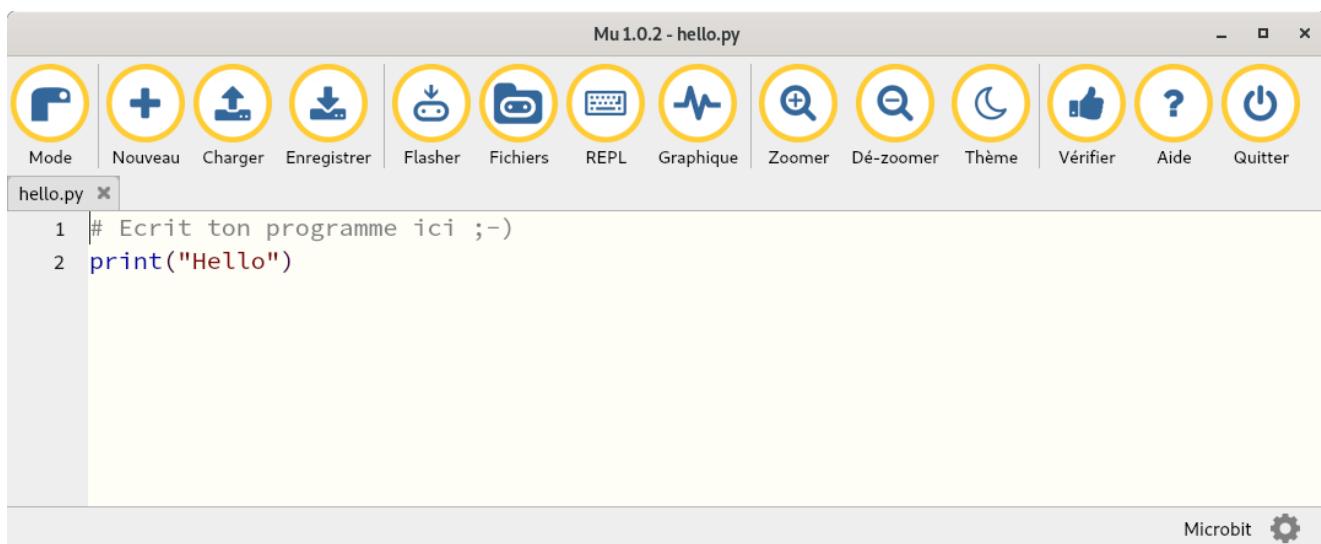


FIG. 1 – Editeur Mu

Sinon l'éditeur [Thonny](#) est la meilleure alternative à Mu.

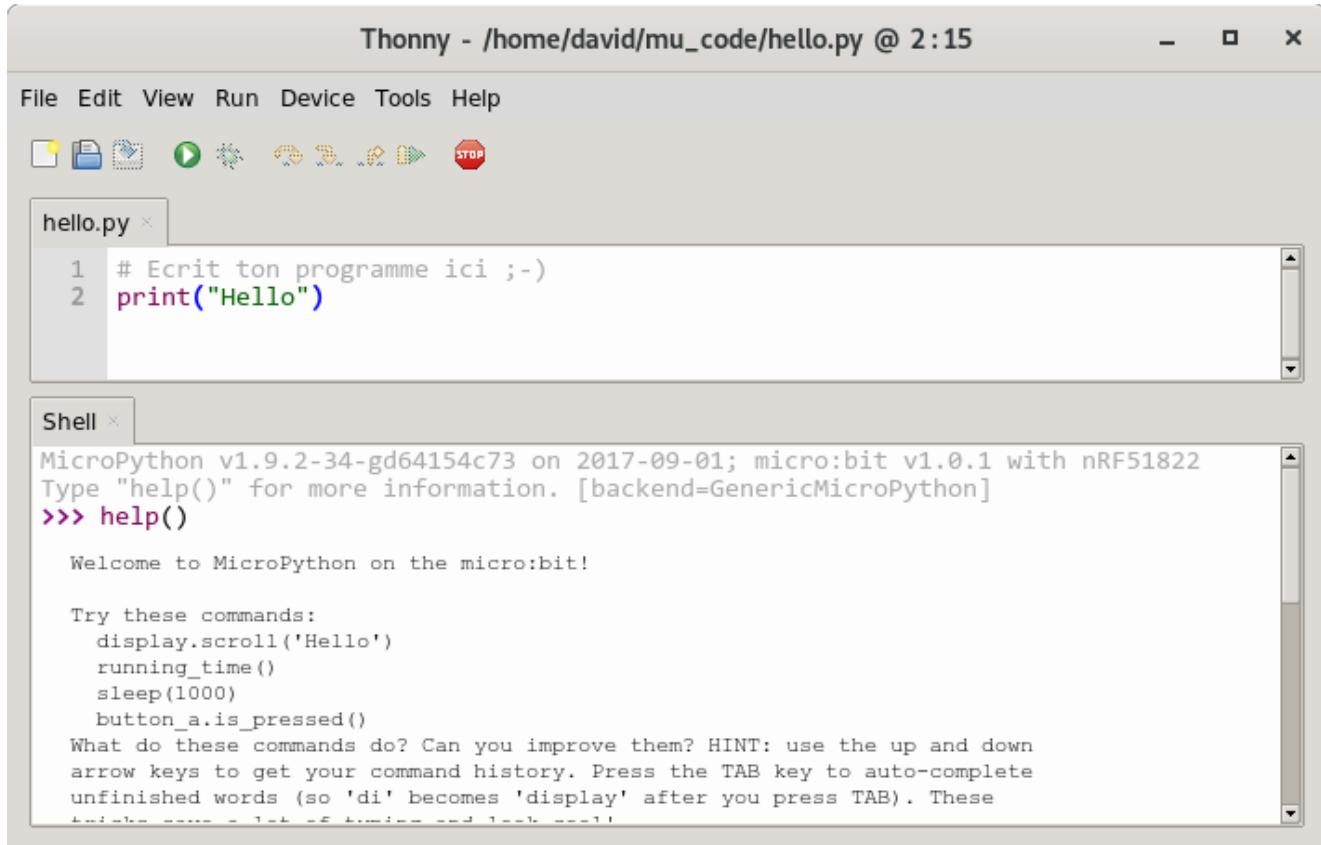


FIG. 2 – Editeur Thonny

7.2.2 Librairie

Les fonctionnalités spécifiques à la micro :bit sont gérées par librairie `microbit`.

Les broches sont notées sous la forme `pinN` où `N` est le numéro de la broche (ex. `pin0`, `pin1`, ...).

7.2.3 Écrire sur une sortie digitale

La fonction `write_digital(val)` impose l'état logique `val` (0 ou 1) sur une sortie digitale.

```
from microbit import *
pin0.write_digital(1)  # Etat 1 sur P0
pin2.write_digital(0)  # Etat 0 sur P2
```

7.2.4 Lire une entrée digitale

La fonction `read_digital()` renvoie le niveau logique sur un broche.

```
from microbit import *
val = pin0.read_digital()  # Renvoie le niveau logique sur P0
print(val)                # Affichage du niveau logique
```

La micro :bit intègre deux boutons notés A et B respectivement avec les attributs `button_a` et `button_b`. Ils sont connectés à P5 et P11.

```
from microbit import *
val = button_a.is_pressed() # renvoie True ou False
print(val)                 # Affichage de l'état du bouton A
```

7.2.5 Générer une tension MLI (PWM)

Comme avec Arduino, il est possible de générer une tension Modulée en Largeur d'Impulsion (MLI ou PWM en anglais) avec la fonction `write_analog(duty)`. Le paramètre `duty` est le rapport cyclique codé sur 12 bits (de 0 à 1023 pour un rapport cyclique de 0 à 100%).

La fréquence du signal est fixée par les fonctions `set_analog_period(T)` ou `set_analog_period_microseconds(T)` où `T` est la période respectivement en millisecondes et microsecondes.

```
from microbit import *
Pin0.set_analog_period(100) # fixe une période de 100 ms
pin0.write_analog(767)      # rapport cyclique à 75% sur P0
```

Note

Il est intéressant ici de mesurer la tension moyenne au voltmètre numérique (entre GND et P0) en position DC.

7.2.6 Mesurer une tension (CAN)

La lecture sur 12 bits d'une tension entre 0 V et 3,3 V est effectuée par la méthode `read_analog()`.

```
from microbit import *
val = Pin0.read_analog() # renvoie un nombre entre 0 à 1023
print(val*3.3/1023)     # affichage de la tension
```

7.2.7 Générer une tension analogique (CNA)

La carte ne dispose pas de vraies sorties analogiques (pas de CNA) !

7.2.8 Faire une pause

Les fonctions `sleep(T)`, `sleep_ms(T)` et `sleep_us(T)` du module `utime` permettent de faire une pause de durée `T` respectivement en seconde, milliseconde et microseconde.

```
from utime import sleep
while True:
    Pin0.write_digital(1)
    sleep(1)
    Pin0.write_digital(0)
    sleep(1)
```

7.2.9 Mesurer une durée

Il est possible de mesurer la durée d'une impulsion à l'état haut ou l'état bas avec la fonction `time_pulse_us()` du module `machine` commun à tous les microcontrôleurs sous MicroPython.

exemple

mesurer la durée à l'état haut d'une impulsion sur l'entrée X1.

```
from microbit import *
from machine import time_pulse_us
duree = time_pulse_us(Pin0,1)
print(duree)
```

7.3 Les bases de la carte Micro :bit

7.4 Programmes du lycée

7.4.1 Capteur résistif - CTN (seconde générale)

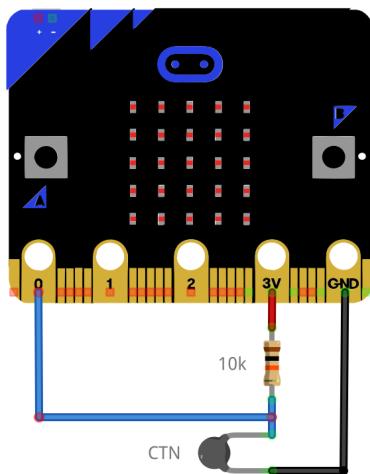


FIG. 3 – Branchement d'une CTN sur l'entrée analogique pin0

```
# Mesure de la resistance d'une CTN
from microbit import *

Ro = 10e3                      # Résistance série

while True:
    N = pin0.read_analog()      # Mesure de la tension
    R = Ro*N/(1023-N)          # Calcul de R
    print("R =", R)            # Affichage
    sleep(1000)                 # Temporisation
```

7.4.2 Application : thermomètre numérique

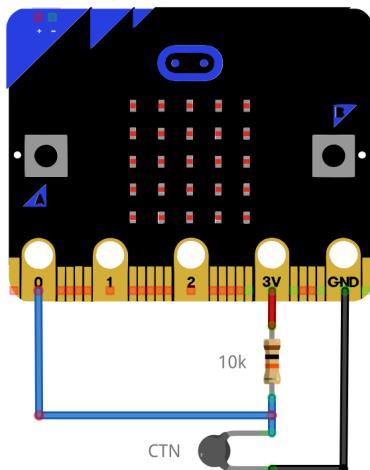


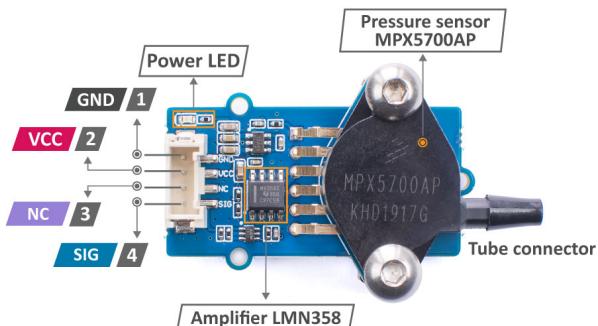
FIG. 4 – Branchement d'une CTN sur l'entrée analogique pin0

```
# Mesure de la résistance d'une CTN et calcul de la température
# Calcul de la température à partir de la relation de Steinhart-Hart
from microbit import *
from math import log

Ro = 10e3      # Résistance série
A = 1.0832e-3 # Coefficients de Steinhart-Hart
B = 2.1723e-4 # ...
C = 3.2770e-7 # ...

while True:
    N = pin0.read_analog()                      # Mesure de la tension
    R = Ro*N/(1023-N)                          # Calcul de R
    T = 1/(A + B*log(R) + C*log(R)**3) - 273.15 # Relation de Steinhart-Hart
    print("R =", R, "T =", T)                  # Affichage
    sleep(1000)                                # Temporisation
```

7.4.3 Mesurer une pression - Loi de Mariotte (première générale)



- 1** : Connected to the system GND
- 2** : Power supply from grove 5V/3.3V
- 3** : Not connected
- 4** : Analog signal output

FIG. 5 – Capteur MPX5700AP Grove (15 kPa à 700 kPa) (image : seeedstudio)

Cet exemple utilise également un module Grove MPX5700AP (15-700 kPa). Les mesures sont affichées au format CSV pour exploitation avec un tableur, Rgressi, Latis ou Python par un copier-coller.

```
# Vérification de la loi de Boyle-Mariotte avec module Grove MPX5700AP 15-700 kPa > 0.2-
˓4.7 V
from microbit import *

Vcc = 3.09      # Mesure au voltmètre entre 3V3 et GND
No  = 4         # Décalage (entier) obtenu pour une tension de 0V

Pmin = 15       # Pression minimale
Pmax = 700      # Pression maximale
      # Amplification = 3,3/4,7 = 0.702
Umin = 0.14     # Tension minimale (0.2V * 0.702 = 0.140V) N=44
Umax = 3.3      # Tension maximale (4.7V * 0.702 = 3.300V) N=1023

volume   = [60,50,40,35,30,25]    # Proposition de volumes - 40 mL pour pression
˓atmosphérique
pression = []                  # Tableau des pressions

# Mesures
for vol in volume :           # Parcours des volumes prédefinis
    input("Régler le volume sur " + str(vol) + " mL") # Validation du réglage du volume
    N = pin1.read_analog()        # Lecture de la tension numérique
˓(10 bit)
    U = (N-No)*Vcc/(1023-No)    # Calcul de la tension
    P = (Pmax-Pmin)/(Umax-Umin)*(U-Umin) + Pmin    # Calcul de la pression
    print(P, "kPa")              # Affichage de la pression
    pression.append(P)           # Ajout de la mesure dans le
˓tableau de pression

# Affichage au format CSV
```

(suite sur la page suivante)

(suite de la page précédente)

```
print("V ; P")                      # Affichage entête des grandeurs
print("mL ; hPa")                   # Affichage entête des unités
for i in range(len(volume)):        # Parcours des points de mesures
    print(volume[i],";",pression[i]) # Affichage des mesures
```

Résultats :

```
Régler le volume sur 60 mL
79.3083 kPa
```

```
Régler le volume sur 50 mL
91.1404 kPa
```

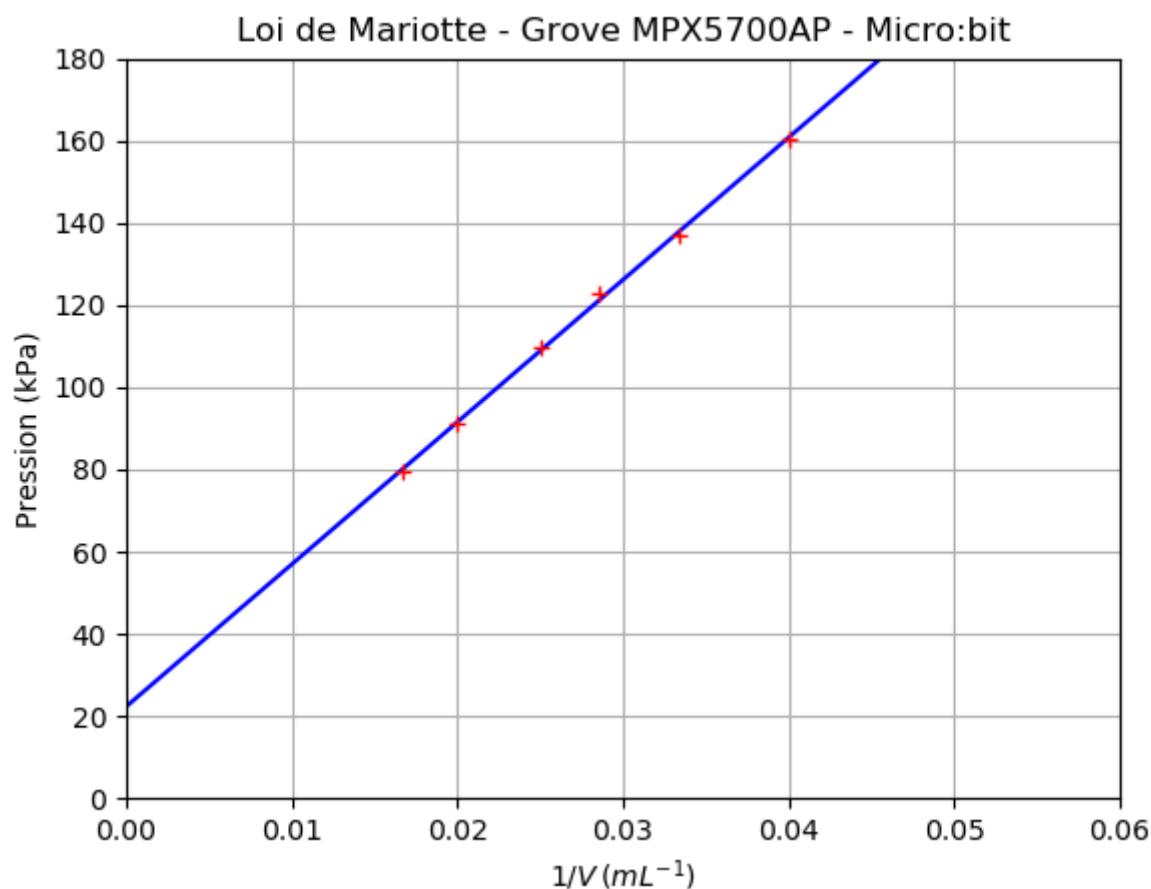
```
Régler le volume sur 40 mL
109.546 kPa
```

```
Régler le volume sur 35 mL
122.693 kPa
```

```
Régler le volume sur 30 mL
137.154 kPa
```

```
Régler le volume sur 25 mL
160.161 kPa
```

```
V ; P
mL ; kPa
60 ; 79.3083
50 ; 91.1404
40 ; 109.546
35 ; 122.693
30 ; 137.154
25 ; 160.161
```



Chapitre 8

Cartes Pyboard (langage Micropython)

8.1 Cartes Pyboard

8.1.1 La carte officielle Pyboard



FIG. 1 – Carte Pyboard v1.1

Créée par le physicien australien Damien George, la Pyboard est la carte officielle de la distribution MicroPython. Construit autour d'un microcontrôleur STM32 de STMicroelectronics, il s'agit d'une des plus puissantes cartes de développement du marché.

Principales caractéristiques de la Pyboard v1.1 :

- Microcontrôleur STM32F405 (ARM Cortex M4 - 32 bit - 168 Mhz) ;
- 1024 ko de mémoire flash (ROM) ;* 192 ko de mémoire vive (RAM) ;
- niveau de tension à 3,3V;
- 1 port micro USB (programmation REPL + accès mémoire flash) ;
- lecteur micro SD ;
- 2 boutons, 4 LED, 1 accéléromètre 3 axes ;

- 35 ports E/S ;
- 3 CAN disponibles sur 16 broches ;
- 2 CNA disponibles sur 2 broches (X5 et X6) ;
- 17 timers.

Brochage :

Les broches d'entrée/sortie sont réparties en deux blocs : X1 à X12 et Y1 à Y12.

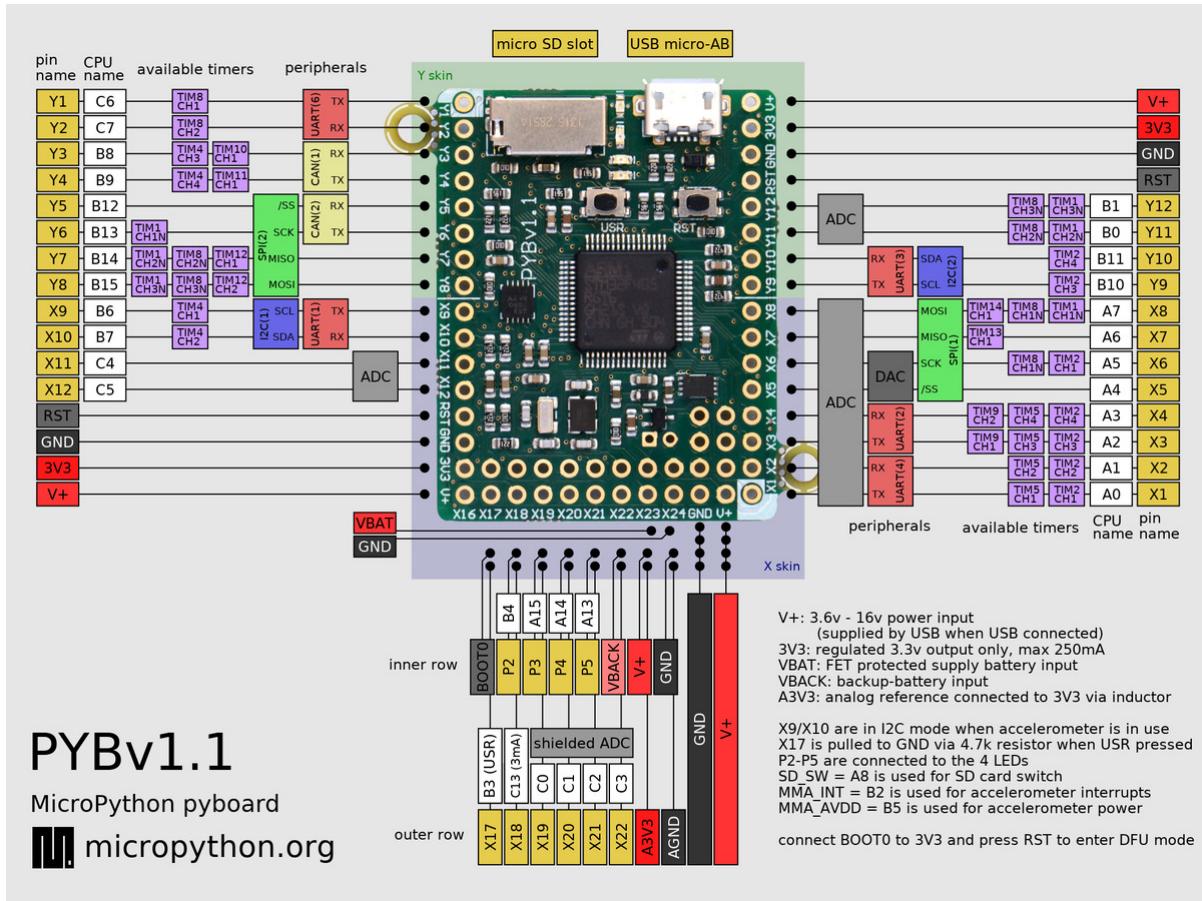


FIG. 2 – Image : micropython.org

8.1.2 La carte Plug'Uino PY de Sciencthetic

Le fournisseur [Sciencthetic](#) propose une carte Pyboard (STM32F405) spécialement conçue pour les sciences physiques **Plug'Uino PY**.



FIG. 3 – Image : Sciencthetic

Par rapport à la carte officielle, cette carte a l'avantage de disposer :

- de **protections des entrées/sorties** contre les mauvaises manipulations ;
- d'une **adaptation du niveau des tensions à 5V** en logique et en analogique ;
- d'un brochage Arduino en plus des connecteurs SATA pour les capteurs.

Elle dispose en autres :

- d'un écran OLED 64x32 pixels ;
- d'un accéléromètre 3D ;
- d'un lecteur de carte SD.

8.1.3 La carte Feather STM32F405 Express d'Adafruit

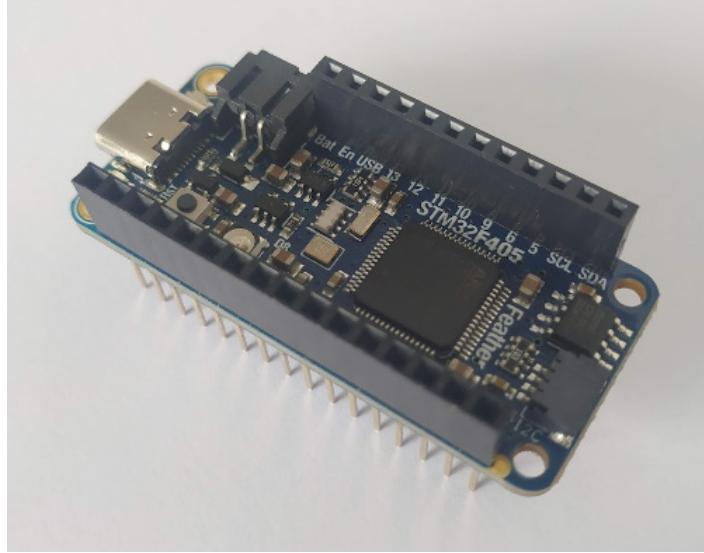


FIG. 4 – Carte Feather STM32F405 Express

Dans la famille Feather d'Adafruit, la carte de développement [Feather STM32F405 Express](#) est équivalent à la Pyboard. Associée au [Grove Shield Featherwing](#), cela peut-être une alternative intéressante pour les sciences physiques.

8.1.3.1 Précautions

⚠️ Attention

Les cartes Pyboard sont alimentées sous 3,3 V. Même si les microcontrôleur STM32F405 ont une tolérance de 5 V sur les entrées logiques, ce n'est pas le cas pour les tensions appliquées sur les entrées analogiques qui ne doivent pas dépassées les 3,3 V sous peine de détruire la carte.

Il faudra donc bien faire attention à la compatibilité des niveaux de tension lors du choix des capteurs (ex. capteur de pression). Les capteurs actifs alimentés sous 5 V ne fonctionneront pas !

Ces précautions ne s'appliquent pas pour la carte [Plug'Uino Py](#) car ce dernière est compatible 5 V.

8.2 Langage Micropython

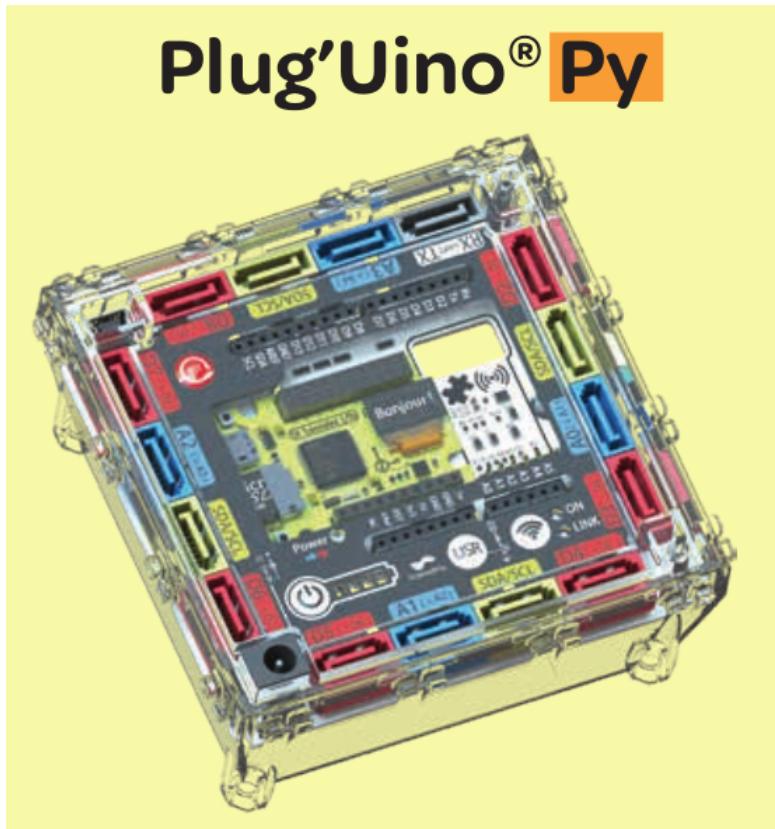


FIG. 5 – Image : Sciencthetic

8.2.1 Edition

Pour de programmer la PyBoard en langage Python natif, un micrologiciel (firmware) contenant l'interpréteur MicroPython est installé dans la mémoire flash de la carte. Ce firmware peut-être mis à jour à partir d'une version plus récente disponible sur le site de [MicroPython](#).

Les logiciels [Thonny](#) et [uPyCraft](#) exploitent pleinement la programmation des microcontrôleurs en MicroPython. Ils permettent deux types de programmation :

- directement dans l'interpréteur MicroPython (REPL) pour tester des instructions par exemple ;
- ou dans un script (fichier avec l'extension .py) qui peut-être sauvegarder sur la mémoire de la carte pour être exécuté de façon autonome sur la carte.

Avertissement

À chaque démarrage de la carte ou lors d'un reset, les fichiers `setup.py` puis `main.py` sont exécutés dans l'ordre. C'est dans le fichier `main.py` que doit être écrit le programme !

8.2.2 Librairie

C'est la librairie `pyb` qui rassemble les fonctions spécifiques à la Pyboard.

8.2.3 Écrire sur une sortie digitale

Les broches de X1 à X12 et de Y1 à Y12 sont utilisables comme entrées/sorties digitales.

L'exemple suivant met la broche X1 en sortie et à l'état bas.

```
>>> from pyb import Pin          # importation de Pin dans la librairie pyb
>>> pinX1 = Pin('X1',Pin.OUT)    # La broche X1 en sortie
>>> pinX1.low()                 # LED éteinte
```

Puis à l'état haut.

```
>>> pinX1.high()
```

Note

Quatre LED internes sont également prises en charge par la librairie pyb.

```
>>> from pyb import LED
>>> led = LED(1)                # 1=rouge, 2=verte, 3=jaune, 4=bleu
>>> led.toggle()               # Permutation de l'état
>>> led.on()                   # Allume la LED
>>> led.off()                  # Éteint la LED
```

8.2.4 Lire une entrée digitale

L'exemple suivant met la broche X2 en entrée puis lit et affiche son état.

```
from pyb import Pin          # importation de Pin de la librairie pyb
pinX2 = Pin('X2',Pin.IN)    # La broche X2 en entrée
val = pinX2.value()         # Lecture de l'état de la broche
print(val)                  # Affichage
```

Note

La carte incorpore un bouton utilisateur USR connectée sur la broche X17 avec une résistance de tirage vers le haut. L'appui sur ce bouton donne donc un niveau bas !

```
from pyb import Switch
btn = Switch()
val = btn.value()           # Retourne l'état courant du bouton USR
print(val)                  # Affiche la valeur (True ou False)
```

8.2.5 Mesurer une tension (CAN)

La conversion analogique numérique sur 12 bits est disponible sur les broches X1 à X8, X11, X12, X19 à X22, Y11 et Y12.

L'exemple suivant lit une tension sur l'entrée X19.

```
from pyb import Pin, ADC
can = ADC(Pin('X19'))      # CAN sur la broche X19
N = can.read()              # lecture d'un entier de 0 à 4095
print(N*3.3/4095)           # Affichage de la tension
```

8.2.6 Générer une tension analogique (CNA)

La carte Pyboard intègre deux convertisseurs numérique-analogique respectivement sur les broches X5 et X6. L'exemple suivant

L'exemple suivant applique une tension de 1,55 V sur la broche X5.

```
from pyb import Pin, DAC
cna = DAC(Pin('X5'))      # CNA sur X5
cna.write(120)              # Ecriture de 120*3,3/255 = 1,55 V
```

Les CNA définis par défaut sur 8 bits peuvent être paramétrés sur 12 bits.

```
from pyb import Pin, DAC
cna = DAC(Pin('X5'))      # CNA sur X5
cna.init(bits=12)          # Paramétrage sur 12 bit
cna.write(1500)             # Ecriture de 1500*3,3/4095 = 1,21 V
```

 Note

Il est aussi possible de générer une tension périodique de forme quelconques.

8.2.7 Générer une tension MLI (PWM)

La génération d'une tension MLI est possible sur un canal d'un timer (4 canaux par timer). Voir brochage pour repérer les canaux des timers.

L'exemple suivant génère une tension MLI sur la broche X2 avec un rapport cyclique de 30%.

```
from pyb import Pin, Timer

pwm = Timer(2).channel(2, Timer.PWM, pin=Pin('X2'))  # pwm sur le canal 2 du Timer 2
pwm.pulse_width_percent(30)                            # réglage du rapport cyclique
```

 Note

Il est intéressant ici de mesurer la tension moyenne au voltmètre numérique (entre GND et X2) en position DC.

8.2.8 Faire une pause

```
from time import sleep

sleep(1)  # Pause de 1 s
```

8.2.9 Mesurer une durée

Il est possible de mesurer la durée d'une impulsion à l'état haut ou l'état bas avec la fonction `time_pulse_us()` du module `machine` commun à tous les microcontrôleurs sous MicroPython.

L'exemple suivant mesure la durée (en microsecondes) à l'état haut d'une impulsion sur l'entrée X1.

```
from pyb import Pin
from machine import time_pulse_us

duree = time_pulse_us(Pin('X1'), 1)
```

8.3 Les bases

8.4 Programme du lycée

8.4.1 Capteur résistif - CTN (seconde générale)

Le montage ci-dessous utilise une carte Feather STM32F405 Express. L'entrée analogique A0 mesure la tension du capteur.

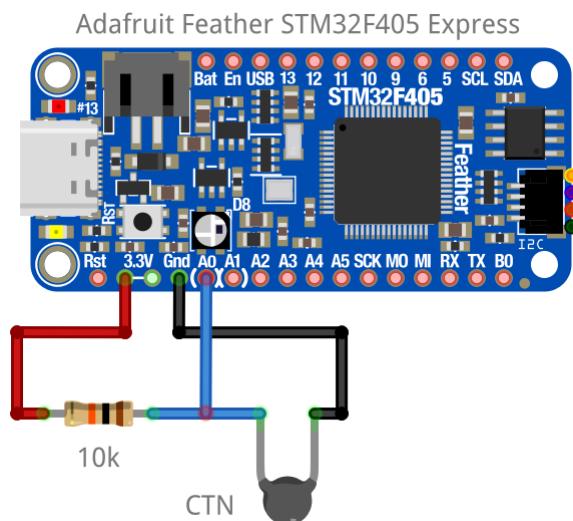


FIG. 6 – Branchement d'une CTN sur l'entrée analogique A0

```
# Mesure de la resistance d'une CTN
from pyb import Pin, ADC, delay

adc = ADC(Pin("A0")) # Déclaration du CAN
Ro = 10e3 # Résistance série

while True:
    N = adc.read() # Mesure de la tension
    R = Ro*N/(4095-N) # Calcul de R
    print("R =", R) # Affichage
    delay(1000) # Temporisation
```

8.4.2 Application : thermomètre numérique

Adafruit Feather STM32F405 Express

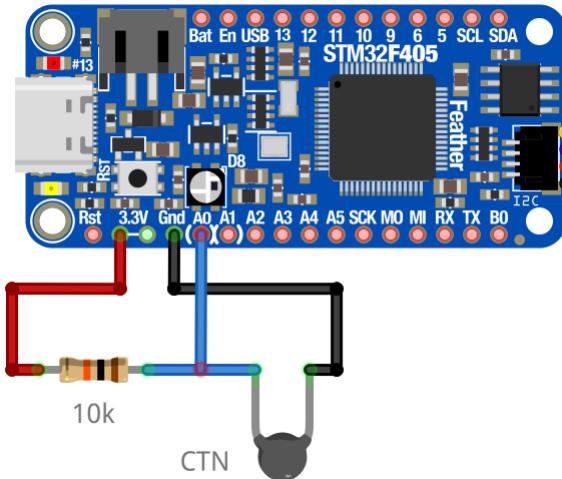


FIG. 7 – Branchement d'une CTN sur l'entrée analogique A0

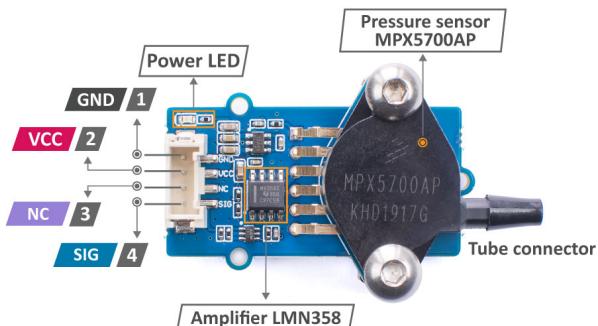
```
# Mesure de la résistance d'une CTN et calcul de la température
# Calcul de la température à partir de la relation de Steinhart-Hart
from pyb import Pin, ADC, delay
from math import log

adc = ADC(Pin("A0"))          # Déclaration du CAN

Ro = 10e3                      # Résistance série
A = 0.0010832035972923174    # Coeff. de Steinhart-Hart
B = 0.00021723460553451255  # ...
C = 3.276999926128753e-07   # ...

while True:
    N = adc.read()                  # Mesure de la tension
    R = Ro*N/(4095-N)              # Calcul de R
    T = 1/(A + B*log(R) + C*log(R)**3) - 273.15 # Relation de Steinhart-Hart
    print("R =", R, "T =", T)       # Affichage
    delay(1000)                     # Temporisation
```

8.4.3 Mesurer une pression - Loi de Mariotte (première générale)



- 1** : Connected to the system GND
- 2** : Power supply from grove 5V/3.3V
- 3** : Not connected
- 4** : Analog signal output

FIG. 8 – Capteur MPX5700AP Grove (15 kPa à 700 kPa) (image : seeedstudio)

Cet exemple utilise également un module Grove MPX5700AP (15-700 kPa). Les mesures sont affichées au format CSV pour exploitation avec un tableur, Rgressi, Latis ou Python par un copier-coller.

```
# Vérification de la loi de Boyle-Mariotte avec module Grove MPX5700AP 15-700 kPa > 0.2-
˓4.7 V
from pyb import Pin, ADC

adc = ADC(Pin("A0")) # Déclaration du CAN

Pmin = 15 # Pression minimale
Pmax = 700 # Pression maximale
Umin = 174 # Amplification = 3,3/4,7 = 0.702
Umax = 4095 # Tension minimale (0.2V * 0.702 = 0.140V) N=174
# Tension maximale (4.7V * 0.702 = 3.300V) N=4095

volume    = [60,50,40,35,30,25] # Proposition de volumes - 40 mL pour pression
˓atmosphérique
pression = [] # Tableau des pressions

# Mesures
for vol in volume : # Parcours des volumes prédefinis
    input("Régler le volume sur " + str(vol) + " mL") # Validation du réglage du volume
    U = adc.read() # Lecture de la tension numérique
˓(12 bit)
    P = (Pmax-Pmin)/(Umax-Umin)*(U-Umin) + Pmin # Calcul de la pression du capteur
    print(P, "kPa") # Affichage de la pression
    pression.append(P) # Ajout de la mesure dans le
˓tableau de pression

# Affichage au format CSV
print("V ; P") # Affichage entête des grandeurs
print("mL ; hPa") # Affichage entête des unités
```

(suite sur la page suivante)

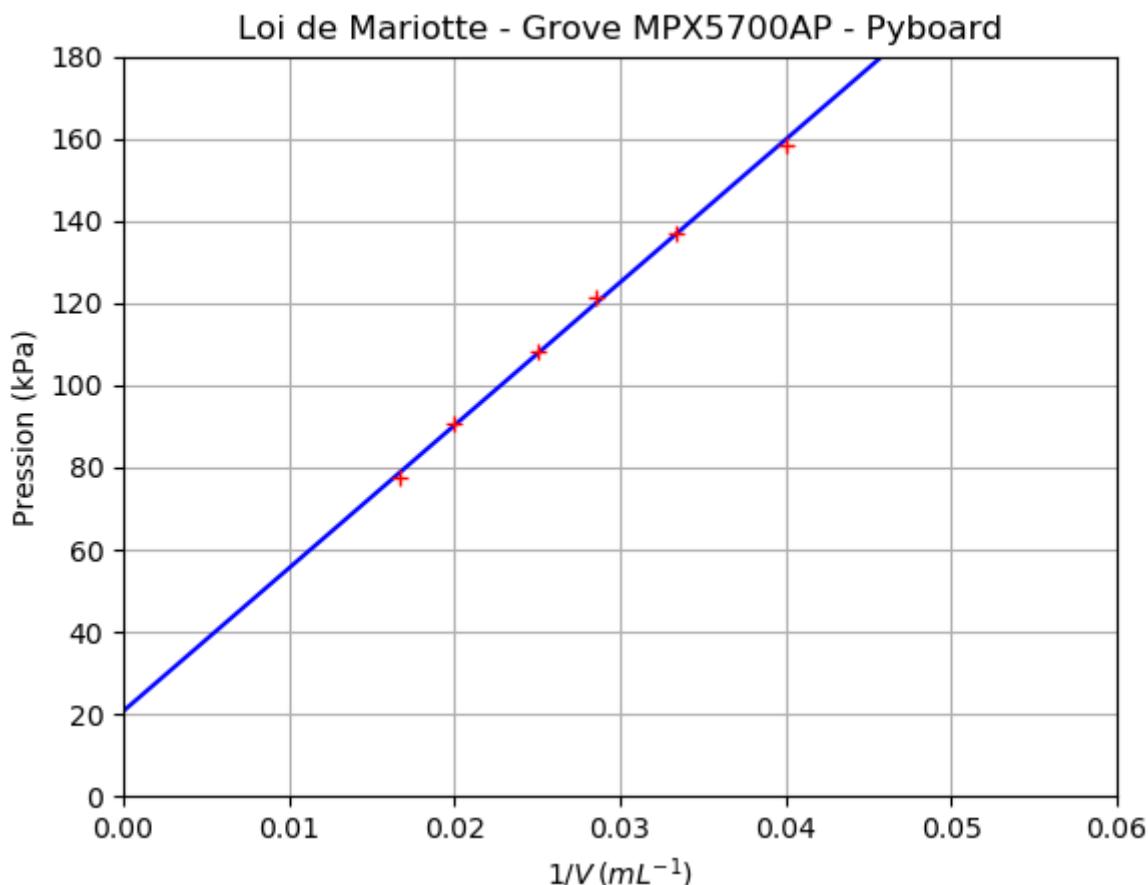
(suite de la page précédente)

```
for i in range(len(volume)):          # Parcours des points de mesures
    print(volume[i],";",pression[i])  # Affichage des mesures
```

Résultats :

```
Régler le volume sur 60 mL
79.81383 kPa
Régler le volume sur 50 mL
91.69345 kPa
Régler le volume sur 40 mL
109.1635 kPa
Régler le volume sur 35 mL
122.0913 kPa
Régler le volume sur 30 mL
137.4649 kPa
Régler le volume sur 25 mL
157.7302 kPa

V ; P
mL ; kPa
60 ; 79.81383
50 ; 91.69345
40 ; 109.1635
35 ; 122.0913
30 ; 137.4649
25 ; 157.7302
```



Chapitre 9

Cartes Arduino (librairie Numpy de Python)

9.1 Langage Python avec Numpy



FIG. 1 – Arduino Uno R3

9.1.1 Qu'est-ce que Numpy?

Numpy est une librairie pour Python utilisée pour le **pilotage** d'une carte Arduino par le port USB d'un ordinateur programmé en Python.

Avertissement

Il est important de retenir que Numpy ne permet pas un fonctionnement autonome de la carte Arduino puisque la carte doit-être constamment connectée à l'ordinateur sur lequel le programme Python est exécuté !

9.1.2 Principe de fonctionnement

La carte Arduino a été préalablement programmer avec le micro-logiciel **Numpy-firmware** téléchargé à partir du logiciel Arduino. Ce firmware est un **programme particulier** (écrit en langage Arduino C/C++) qui gère le **protocole de communication** entre le programme Python exécuté sur l'ordinateur et la carte Arduino.

9.1.3 Téléversement du firmware Numpy sur la carte Arduino

Il faut d'abord installer le firmware dans les croquis du logiciel Arduino. Le fichier `numpy-firmware-master.zip` est à télécharger sur le site <https://github.com/numpy/numpy-firmware>.

Numpy firmware repository

Branch: master ▾ New pull request

Find file

Clone or download ▾

**Téléchargement du fichier
numpy-firmware-master.py**

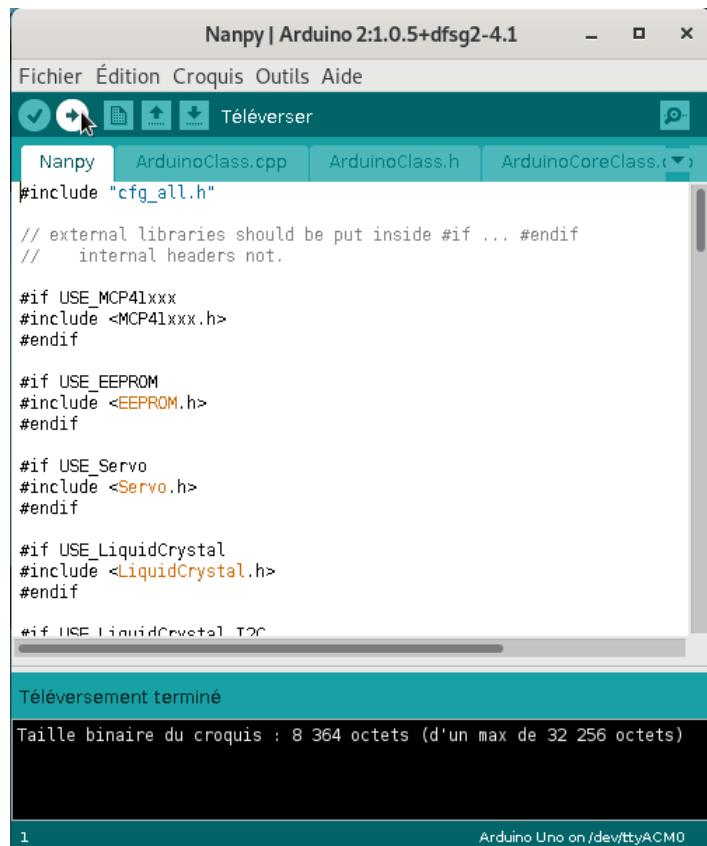
Commit	Message	Date
pony	Merge branch 'ololobus-readme-badge-fix'	18 Jul 2017
Numpy	Added 2 new return types to MethodDescriptor:	3 years ago
codegen	defines: fix A0,SS,MOSI,MISO,SCK (they are actually const int variables)	4 years ago
tests	test: simplify cfg.h	3 years ago
.gitignore	build test + travis (#11)	5 years ago
.travis.yml	travis: use pypi version of confduino	4 years ago
CONTRIBUTORS	Update CONTRIBUTORS	3 years ago
LICENSE	Update year	5 years ago
README.md	README.md fix	2 years ago
configure.sh	Do not track cfg.h	6 years ago
sample_cfg.h	remove warning because MCP41XXX was fixed	2 years ago

Puis il faut procéder dans l'ordre suivant :

- extraire l'archive `numpy-firmware-master.zip` dans le répertoire de travail Arduino ;
- aller dans le répertoire `numpy-firmware-master`
- copier le fichier `sample_cfg.h` dans le répertoire Numpy ;
- pour finir, renommer le nouveau fichier `sample_cfg.h` en `cfg.f`.

Ensuite, il faut procéder au téléchargement du croquis du firmware sur la carte Arduino. Dans le logiciel Arduino, ouvrir le carnet de croquis `Numpy.ino` à partir du répertoire Numpy.

Puis téléverser le programme sur la carte Arduino.



La carte est maintenant prête pour un fonctionnement avec Nanpy !

Avertissement

Ne pas oublier de sélectionner le type de carte (ex. Arduino Uno) et le port de communication série (ex. COM3) dans le menu *Outils* avant le téléversement du firmware !

9.1.4 Installation la librairie Nanpy sur l'ordinateur

L'installation dépend de la distribution Python utilisée sur l'ordinateur. La librairie Nanpy peut-être installée par défaut.

Si ce n'est pas le cas, il faut installer manuellement à partir du dépôt internet Pypi (<https://pypi.org/project/nanpy/>) avec la commande pip :

```
python -m pip install numpy
```

Ou bien à partir de l'archive Zip `numpy-master.zip` téléchargée sur le site Github de Nanpy (<https://github.com/numpy>) :

```
python -m pip install numpy-master.zip
```

Avertissement

L'archive Zip doit être enregistré dans le répertoire où la commande pip a été exécutée.

9.1.5 Exemple : le programme Blink

Voici un exemple du programme **Blink** en Python.

```
from numpy import ArduinoApi, SerialManager    # Fonctions de Numpy
from time import sleep                         # Importation fonction sleep()

port = SerialManager(device='COM3')            # Sélection du port série (exemple : device =
                                             # 'COM6')
uno = ArduinoApi(connection=port)              # Déclaration de la carte Arduino Uno

pinLed = 13                                    # Led intégrée sur broche 13
uno.pinMode(pinLed, uno.OUTPUT)                # Broche Led en sortie

for i in range(100):                          # Boucle : répéter 100 fois
    uno.digitalWrite(pinLed, 1)                # Led allumée
    sleep(1)                                   # Attendre 1 s
    uno.digitalWrite(pinLed, 0)                # Led éteinte
    sleep(1)                                   # Attendre 1 s
```

9.2 Les bases de Numpy

9.2.1 Allumer une LED (sorties numériques)

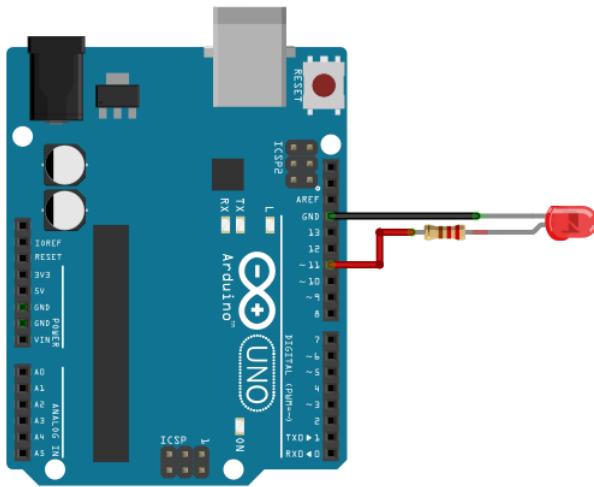


FIG. 2 – Branchement d'une LED sur la broche 11

```
from numpy import ArduinoApi, SerialManager
from time import sleep                         # Importation fonction sleep()

port = SerialManager(device='COM6')            # Sélection du port série à modifier
uno = ArduinoApi(connection=port)              # Déclaration de la carte Arduino Uno

pinLed = 11                                    # Led branchée sur broche 11
uno.pinMode(pinLed, uno.OUTPUT)                # Broche Led en sortie

for i in range(100):                          # Boucle : répéter 100 fois
    uno.digitalWrite(pinLed, 1)                # Led allumée
    sleep(1)                                   # Attendre 1 s
    uno.digitalWrite(pinLed, 0)                # Led éteinte
    sleep(1)                                   # Attendre 1 s
```

(suite sur la page suivante)

(suite de la page précédente)

```
sleep(1)                                # Attendre 1 s
uno.digitalWrite(pinLed, 1)               # Led allumée
sleep(1)                                # Attendre 1 s
```

- La fonction `SerialManager()` fixe le port série sur lequel l'Arduino est connecté.
- La fonction `ArduinoApi()` déclare un objet (ici `uno`) qui représente la carte Arduino.
- La méthode `pinMode()` fixe la broche `pinLed` en sortie (`uno.OUTPUT`).
- La méthode `digitalWrite()` écrit un niveau logique sur la broche `pinLed` (0 pour 0 V et 1 pour 5 V).

9.2.2 Modifier l'intensité lumineuse d'une LED (sorties PWM)

Le montage reste le même.

```
from numpy import ArduinoApi, SerialManager
from time import sleep

port = SerialManager(device='COM6')          # Sélection du port série à modifier
uno = ArduinoApi(connection=port)            # Déclaration de la carte Arduino Uno

pinLed = 11                                 # Led branchée sur broche 11
uno.pinMode(pinLed, uno.OUTPUT)              # Broche Led en sortie

for i in range(9):
    uno.analogWrite(pinLed, 10)             # PWM à 10/255
    sleep(1)                               # Attendre 1s
    uno.analogWrite(pinLed, 50)             # PWM à 50/255
    sleep(1)                               # Attendre 1s
    uno.analogWrite(pinLed, 200)            # PWM à 200/255
    sleep(1)                               # Attendre 1s

port.close()                                # Fermeture du port série
```

9.2.3 Afficher des messages dans la console Python

Avec Numpy, le code Python est exécuté sur l'ordinateur et non sur le microcontrôleur. La fonction native `print()` de Python s'utilise tout naturellement !

```
from numpy import ArduinoApi, SerialManager
from time import sleep

port = SerialManager(device='COM6')          # Sélection du port série à modifier
uno = ArduinoApi(connection=port)            # Déclaration de la carte Arduino Uno

N = 0                                     # Initialisation du compteur

while True:
    print("N =", N)                      # Affichage
    N = N + 1                            # Incrémentation du compteur
    sleep(1)                             # Temporisation
```

9.2.4 Mesurer une tension (CAN)

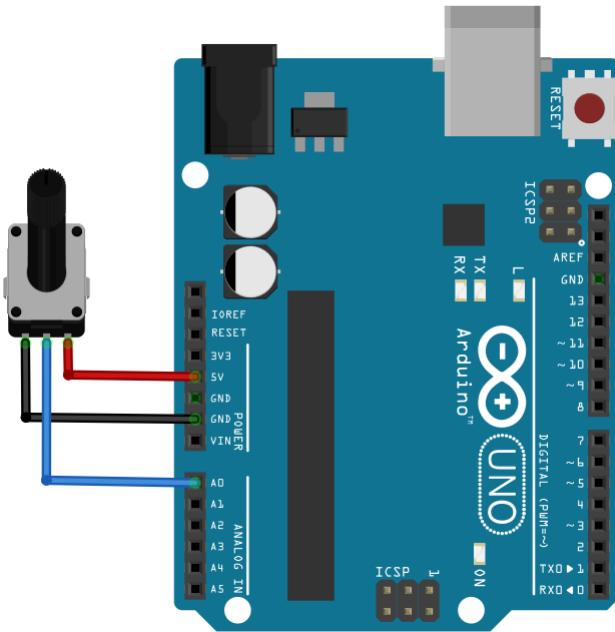


FIG. 3 – Montage potentiométrique sur l'entrée analogique A0

```

from numpy import ArduinoApi, SerialManager
from time import sleep # Importation fonction sleep()

port = SerialManager(device='COM6') # Sélection du port série à modifier
uno = ArduinoApi(connection=port) # Déclaration de la carte Arduino Uno

for i in range(10):
    N = uno.analogRead(0) # Lecture la tension numérique sur A0
    print("N = ", N) # Affichage
    U = N*5/1023 # Calcul de la tension en volt
    print("U = ", round(U, 3), " V") # Affichage
    sleep(1) # Temporisation d'une seconde

uno.connection.close() # Deconnexion de Arduino
port.close() # Fermeture du port série

```

9.3 Programmes du lycée

9.3.1 Capteur résistif - CTN (seconde générale)

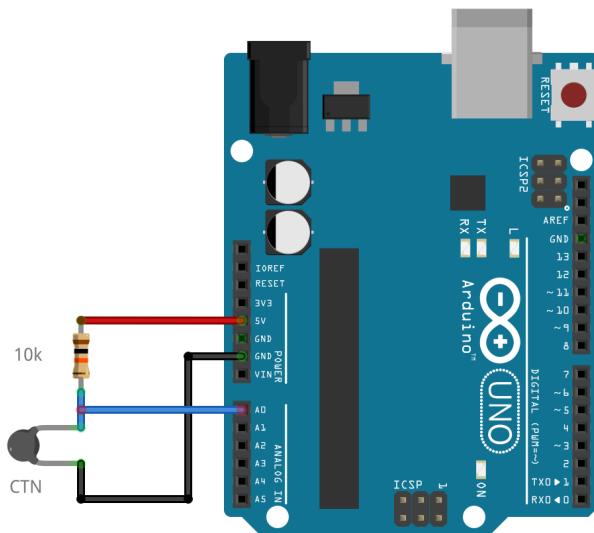


FIG. 4 – Branchement d'une CTN sur l'entrée analogique A0

```
# Mesure de la résistance d'une CTN
from numpy import ArduinoApi
from numpy import SerialManager
from time import sleep

Vcc = 5.0
Ro = 10000

port = SerialManager(device='COM6')
uno = ArduinoApi(connection=port)

while True :
    U = uno.analogRead(0)*5/1023
    R = Ro*U/(Vcc-U)
    print("R = ", R)
    sleep(1)

port.close() # Fermeture du port série
```

Gestion de l'Arduino
Gestion port série
Importation de sleep(seconde)

Tension d'alimentation
Résistance du pont

Sélection du port série (à remplacer)
Déclaration de la carte Arduino

Lecture la tension sur A0
Calcul de la résistance
Affichage
Temporisation d'une seconde

9.3.2 Application : thermomètre numérique

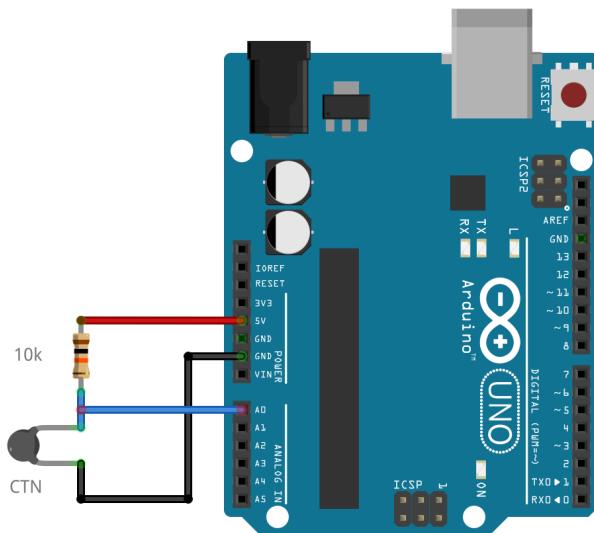


FIG. 5 – Branchement d'une CTN sur l'entrée analogique A0

```
# Mesure de la résistance d'une CTN et calcul de la température
# Calcul de la température à partir de la relation de Steinhart-Hart
from numpy import ArduinoApi           # Gestion de l'Arduino
from numpy import SerialManager         # Gestion port série
from time import sleep                # Importation de sleep(seconde)
from math import log                  # Importation du logarithme népérien

Vcc = 5.0      # Tension d'alimentation
Ro = 10000     # Résistance du pont
A = 1.0832e-3  # Coeff. de Steinhart-Hart
B = 2.1723e-4  # ...
C = 3.2770e-7  # ...

port = SerialManager(device='COM6')      # Sélection du port série (à remplacer)
uno = ArduinoApi(connection=port)        # Déclaration de la carte Arduino

while True :
    U = uno.analogRead(0)*5/1023          # Lecture la tension sur A0
    R = Ro*U/(Vcc-U)                      # Calcul de la résistance
    T = 1.0 / (A + B*log(R) + C*log(R)**3) # Calcul de la température en Kelvin
    T = T-273.15                          # Calcul de la température en Celsius
    print("R = ", R, "T = ", T)            # Affichage
    sleep(1)                                # Temporisation d'une seconde

port.close()                            # Fermeture du port série
```

9.3.3 Émission d'un son (seconde générale)

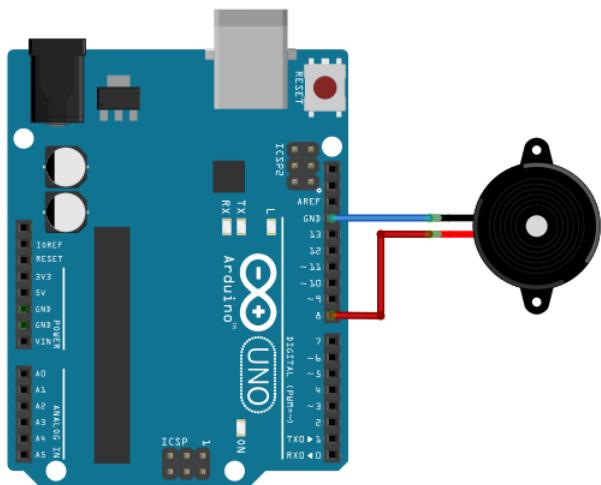


FIG. 6 – Branchement d'un haut-parleur sur la broche 8

La classe Note de Nanpy dispose des méthodes :

- play(fréquence, durée) pour jouer une note pendant un durée en ms;
- stop() pour arrêter la lecture de la note.

```
# Nanpy v0.96
from nanpy import ArduinoApi, SerialManager, Tone
from time import sleep

port = SerialManager(device='/dev/ttyACM0') # Sélection du port série (exemple :  
#device = 'COM6')
uno = ArduinoApi(connection=port)           # Déclaration de la carte Arduino Uno

note = [Tone.NOTE_C4, Tone.NOTE_G3, Tone.NOTE_G3, Tone.NOTE_A3,
       Tone.NOTE_G3, 0 , Tone.NOTE_B3, Tone.NOTE_C4]
noteDuration = [4, 8, 8, 4,
                4, 4, 4]

hp = Tone(8)                                # Haut parleur sur broche 8

for i in range(8):
    duree = 1000/noteDuration[i]              # durée en ms
    hp.play(note[i] , duree)                  # jouer la note
    sleep(1.3*duree*1E-3)                    # pause en les notes

hp.stop()                                     # Arrêt de la lecture
```

Note

Une méthode tone() équivalent à celle du langage Arduino a été ajoutée dans la version modifiée de Nanpy d'Euros mart.

```
# Version modifiée de Nanpy par Euros mart
from nanpy import ArduinoApi, SerialManager, Tone
```

(suite sur la page suivante)

(suite de la page précédente)

```

from time import sleep

port = SerialManager(device='/dev/ttyACM0') # Sélection du port série (exemple :  

#device = 'COM6')  

uno = ArduinoApi(connection=port) # Déclaration de la carte Arduino Uno

melody = [Tone.NOTE_C4, Tone.NOTE_G3, Tone.NOTE_G3, Tone.NOTE_A3, Tone.NOTE_G3, 0 ,  

# Tone.NOTE_B3, Tone.NOTE_C4]  

noteDuration = [4, 8, 8, 4, 4, 4, 4]

pinHP = 8 # Haut-parleur sur broche 8

for i in range(8):
    duree = 1/noteDuration[i] # durée en ms
    uno.tone(pinHP, melody[i]) # Lecture de la note
    sleep(duree) # Attendre la lecture
    uno.noTone(pinHP) # Arrêt de la note
    sleep(duree*1.3) # Pause entre les notes

```

9.3.4 Mesurer la célérité d'un son (première générale)

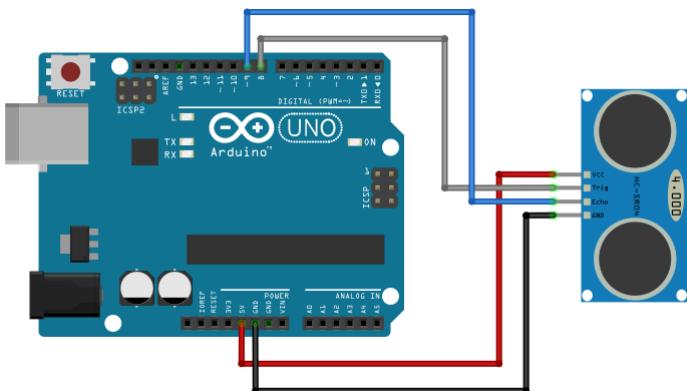


FIG. 7 – Montage célérité son

```

# Réalisé avec une version modifiée de Nanpy par Eurosmart
from nanpy import SerialManager, Ultrasonic
from time import sleep

port = SerialManager(device='/dev/ttyACM0') # Sélection du port série (exemple :  

#device = 'COM6')

moduleUltrason = Ultrasonic(trig=8, echo=9, useInches=False, connection=port) #  

#Déclaration du module HC-SR04
sleep(0.1) # Temporisation

for i in range(10):
    duree = moduleUltrason.get_duration() # Durée en µs pour l'aller-retour du son
    print("Durée =", duree, "µs") # Affichage
    sleep(1) # Temporisation

```

Résultats :

```
Durée = 1456 µs
Durée = 1453 µs
Durée = 1451 µs
Durée = 1450 µs
Durée = 1453 µs
Durée = 1453 µs
Durée = 1447 µs
Durée = 1444 µs
Durée = 1452 µs
Durée = 1446 µs
```

9.3.5 Application : télémètre (première générale)

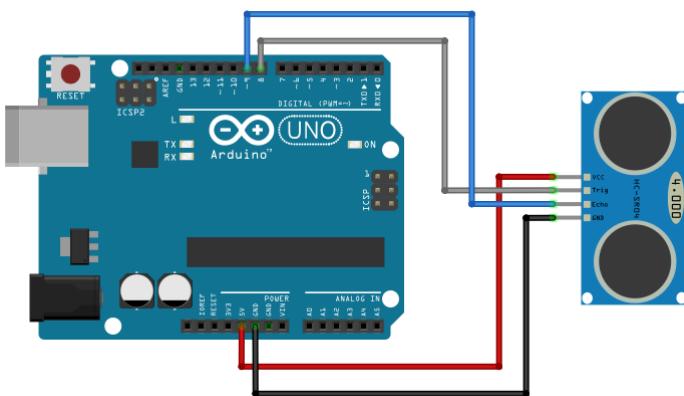


FIG. 8 – Télémètre

```
# Réalisé avec une version modifiée de Nanpy par Eurosmart
from nanpy import SerialManager, Ultrasonic
from time import sleep

port = SerialManager(device='/dev/ttyACM0')      # Sélection du port série (exemple :  
#device = 'COM6')

moduleUltrason = Ultrasonic(trig=8, echo=9, useInches=False, connection=port) #  
#Déclaration du module HC-SR04
sleep(0.1)                                         # Temporisation

vitesse_son = 345                                  # vitesse du son 345 m/S dans l'air

for i in range(10):
    duree = moduleUltrason.get_duration()          # Durée en µs pour l'aller-retour du son
    print("Durée =", duree, "µs")                  # Affichage
    distance = (vitesse_son*duree*1E-6)/2;        # Calcul de la distance en m
    print("Distance =", distance, "m")             # Affichage
    sleep(1)                                       # Temporisation
```

Résultats :

```
Durée = 1997 µs
Distance = 0.3444824999999997 m
Durée = 1996 µs
```

(suite sur la page suivante)

```

Distance = 0.34431 m
Durée = 2019 µs
Distance = 0.3482774999999996 m
Durée = 8214 µs
Distance = 1.416915 m
Durée = 8181 µs
Distance = 1.4112225 m
Durée = 8177 µs
Distance = 1.4105325 m
Durée = 1822 µs
Distance = 0.314295 m
Durée = 1915 µs
Distance = 0.3303375 m

```

9.3.6 Mesurer une pression - Loi de Mariotte (première générale)

Cet exemple utilise un module Grove MPX5700AP (15-700 kPa). Les mesures sont affichées au format CSV pour exploitation avec un tableur, Rgressi, Latis ou Python par un copier-coller.

```

# Vérification de la loi de Boyle-Mariotte avec module Grove MPX5700AP 15-700 kPa > 0.2-
→4.7 V
from nanop import ArduinoApi                      # Gestion de la carte Arduino
from nanop import SerialManager                     # Gestion du port série

port = SerialManager(device='/dev/ttyACM0')         # Sélection du port série (exemple : ↴
→device = 'COM6')
uno = ArduinoApi(connection=port)                  # Déclaration de la carte Arduino

Pmin = 15      # Pression minimale en kPa
Pmax = 700     # Pression maximale en kPa
Umin = 41      # Tension minimale 0.2/5*1023 = 41
Umax = 962     # Tension maximale 4.7/5*1023 = 962

volume    = [60,50,40,35,30,25]      # Proposition de volumes - 40 mL pour pression ↴
→atmosphérique
pression = []                                # Tableau des pressions

# Mesures
for vol in volume :                          # Parcours des volumes prédéfinis
    input("Régler le volume sur " + str(vol) + " mL") # Validation du réglage du volume
    U = uno.analogRead(0)                         # Lecture de la tension numérique ↴
→(10 bit)
    P = (Pmax-Pmin)/(Umax-Umin)*(U-Umin) + Pmin # Calcul de la pression
    print(P, "kPa")                             # Affichage de la pression
    pression.append(P)                           # Ajout de la mesure dans le ↴
→tableau de pression

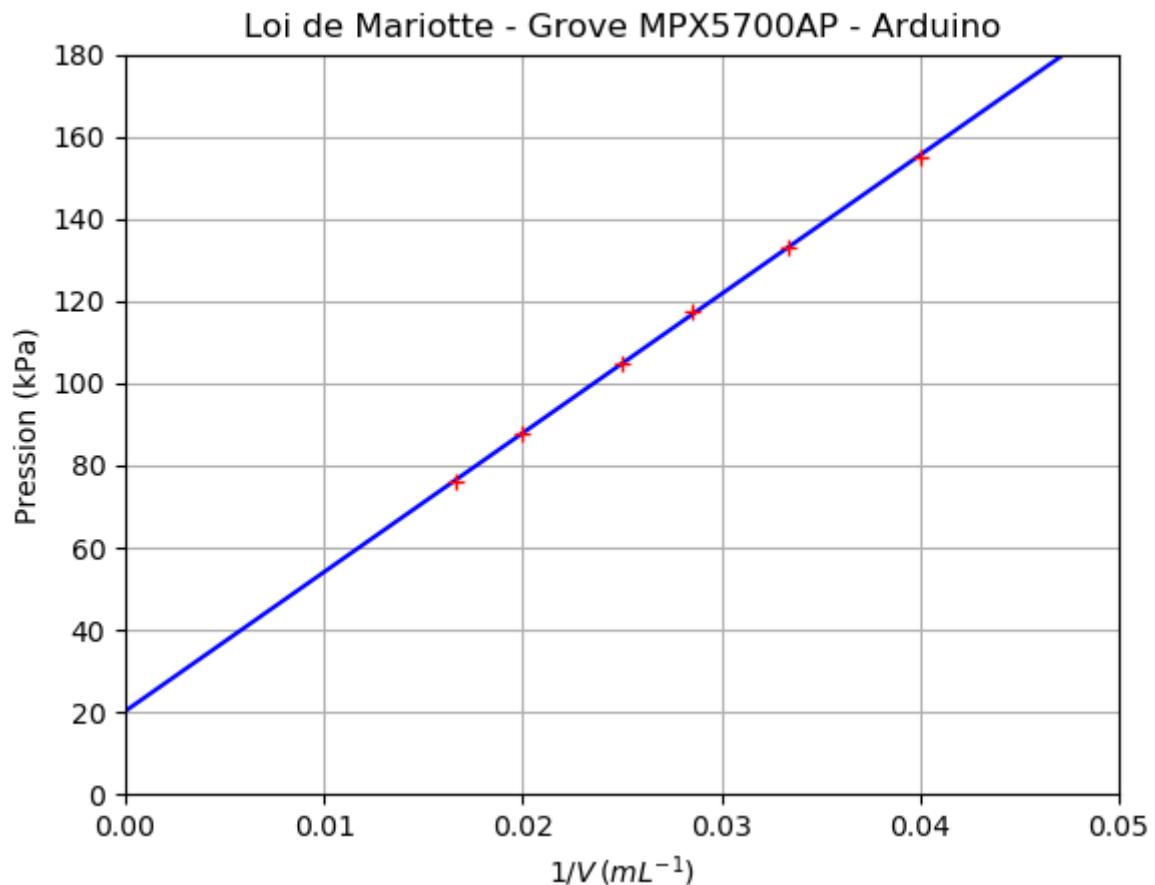
# Affichage au format CSV
print("V ; P")                               # Affichage entête des grandeurs
print("mL ; hPa")                            # Affichage entête des unités
for i in range(len(volume)):                 # Parcours des points de mesures
    print(volume[i], ";" , pression[i]) # Affichage des mesures

```

Résultats :

Régler le volume sur 60 mL
 75.98805646036917 kPa
 Régler le volume sur 50 mL
 87.88816503800217 kPa
 Régler le volume sur 40 mL
 104.99457111834963 kPa
 Régler le volume sur 35 mL
 117.63843648208469 kPa
 Régler le volume sur 30 mL
 133.25732899022802 kPa
 Régler le volume sur 25 mL
 154.82627578718783 kPa

V ; P
 mL ; kPa
 60 ; 75.98805646036917
 50 ; 87.88816503800217
 40 ; 104.99457111834963
 35 ; 117.63843648208469
 30 ; 133.25732899022802
 25 ; 154.82627578718783



Télécharger la documentation en PDF.