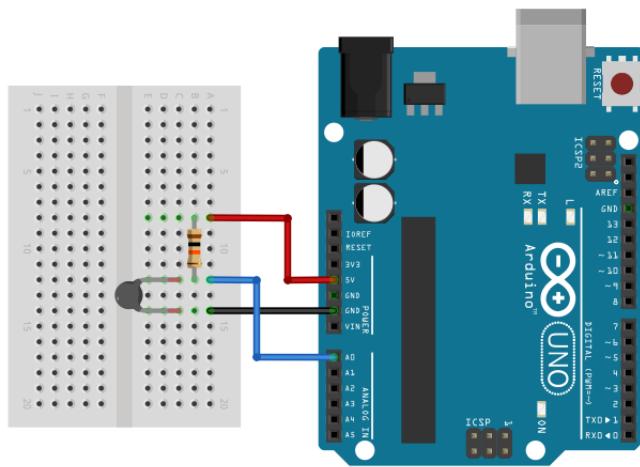


Microcontrôleurs pour les sciences physiques



David THERINCOURT
31 octobre 2019

Table des matières

Table des matières	i
1 Introduction aux microcontrôleurs	3
1.1 Qu'est-ce qu'un microcontrôleur ?	3
1.2 Les différents types de microcontrôleurs	5
1.3 Pourquoi des microcontrôleurs en sciences physiques	5
1.4 Que faire des microcontrôleurs en sciences physiques ?	5
2 Les microcontrôleurs Arduino	7
2.1 Introduction	7
2.2 Les cartes Arduino pour les sciences physiques	8
2.2.1 La carte Arduino UNO (Rev 3)	8
2.2.2 Les cartes spécifiques pour les sciences physiques	9
2.3 Le logiciel Arduino	11
2.3.1 Choix du port de communication avec la carte Arduino	12
2.3.2 Mise en œuvre d'un projet Arduino :	12
2.4 Premier programme : Blink	12
2.4.1 Édition	12
2.4.2 Compilation	14
2.4.3 Téléversement	15
2.4.4 Exécution	16
2.5 Particularité du langage Arduino	16
2.5.1 Syntaxe	17
2.5.2 Typage des variables	17
2.5.3 Constantes prédéfinies	18
2.5.4 Structure du programme	18
2.6 Pilotage d'une carte Arduino en Python avec Nanpy	18
2.6.1 Qu'est-ce que Nanpy ?	18
2.6.2 Principe de fonctionnement	18
2.6.3 Installation de firmware Nanpy sur une carte Arduino	18
2.6.4 Installer la librairie Nanpy sur l'ordinateur	21
2.6.5 Exemple : Blink	21
3 Les bases	23
3.1 Allumer une LED (sorties numériques)	23
3.1.1 Principe	23
3.1.2 Montage	23
3.1.3 Programme	24
3.1.4 A retenir	24
3.1.5 Applications	25
3.1.6 Avec Python et Nanpy	25
3.2 Modifier l'intensité lumineuse d'une LED (sorties PWM)	25
3.2.1 Principe	25
3.2.2 Montage	27
3.2.3 Programme	27
3.2.4 A retenir	27

3.2.5	Applications	27
3.3	Communication avec un ordinateur (port série)	28
3.3.1	Principe	28
3.3.2	Montage	28
3.3.3	Programme	28
3.3.4	A retenir	29
3.3.5	Applications	29
3.4	Afficheur LCD	29
3.4.1	Afficheur LCD 16x2 (port parallèle)	30
3.4.2	Afficheur LCD 16x2 (I2C)	31
3.5	Mesurer une tension (CAN)	32
3.5.1	Principe	32
3.5.2	Montage	33
3.5.3	Programme	33
3.5.4	A retenir	34
3.5.5	Applications	34
4	Nouveaux programmes du lycée	35
4.1	Capteur résistif - CTN (seconde générale)	35
4.1.1	Principe	35
4.1.2	Montage	36
4.1.3	Programme	37
4.1.4	A retenir	37
4.1.5	Aller plus loin	37
4.2	Émission d'un son (seconde générale)	39
4.2.1	Principe	39
4.2.2	Montage	40
4.2.3	Programme	40
4.3	Mesurer la célérité d'un son (première générale)	41
4.3.1	Principe	41
4.3.2	Montage	42
4.3.3	Programme	43
4.3.4	A retenir	43
4.3.5	Aller plus loin	43
4.4	Mesurer une pression - Loi de Mariotte (première générale)	45
4.4.1	Principe	45
4.4.2	Montage	46
4.4.3	Programme	47
4.4.4	Résultats	47
4.5	Mesurer une pression - Loi de la statique des fluides (première générale)	48
4.5.1	Principe	48
4.5.2	Montage	48
4.5.3	Programme	48
4.5.4	A retenir	48
4.6	Géométrie d'un condensateur (terminale générale)	48
4.6.1	Principe	49
4.6.2	Montage	49
4.6.3	Programme	49
4.6.4	A retenir	49
4.7	Capteur capacitif (terminale générale)	49
4.7.1	Principe	49
4.7.2	Montage	49
4.7.3	Programme	49
4.7.4	A retenir	49
5	Aller plus loin	51
5.1	Apdatation d'un capteur teslamètre Jeulin	51
5.1.1	Principe	51

5.1.2	Montage	51
5.1.3	Applications	51
5.2	Module Gravity wattmètre/joulemètre	51
5.2.1	Principe	51
5.2.2	Montage	51
5.2.3	Applications	51
5.3	Acquisition de données (mode temporel)	51
5.3.1	Principe	51
5.3.2	Montage	51

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.

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

Chapitre 1

Introduction aux microcontrôleurs

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 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). Par exemple, la célèbre carte Arduino UNO fonctionne avec un microcontrôleur 8 bits !

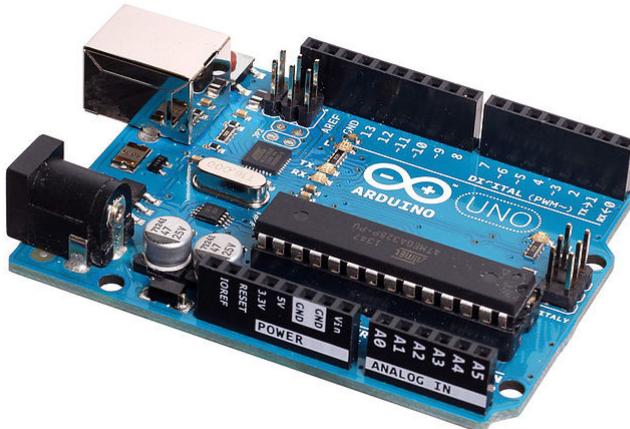


Fig. 1 – Carte Arduino UNO (microcontrôleur Atmel ATMEGA 328)

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)

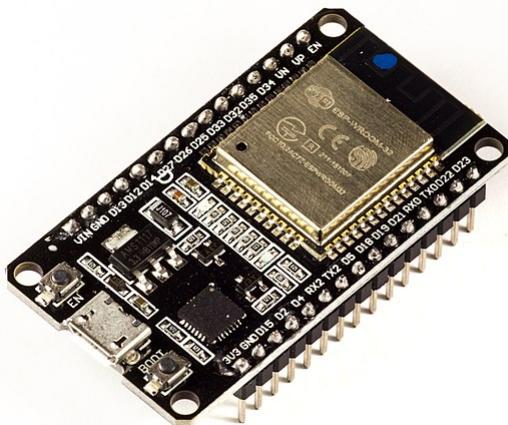


Fig. 4 – Carte ESP-WROOM-32 (microcontrôleur ESP32)

1.2 Les différents types de microcontrôleurs

Plusieurs critères permettent de différencier les microcontrôleurs : leur architecture, le nombre et le type de périphériques d'entrées/sorties, le langage de programmation, ...

Ces deux derniers points sont à prendre en considération. En particulier, au lycée, le choix de Python comme langage de programmation des microcontrôleurs paraît logique.

En pratique, la plupart des manuels de sciences physiques et des fabricants de matériel spécialisé se sont tournés vers les populaires cartes Arduino même si le langage de programmation utilisé n'est pas du Python mais du C/C++ !

1.3 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 la valeur d'une **grandeur physique analogique** est obtenue sur un appareil numérique.

Il en est de même pour la génération de signaux (ex. son) à partir d'un appareil numérique.

1.4 Que faire des microcontrôleurs en sciences physiques ?

En sciences physiques, les microcontrôleurs peuvent-être utiliser pour :

- des **petites applications** (ex. thermomètre, télémètre à ultrasons, ...) ou dans des **projets** en enseignement scientifique ;
- **générer de signaux** (ex. génération d'un son, génération d'une commande, ...);
- **mesurer des durées** (ex. période, fréquence, temps caractéristique, ...).

Mais il est également possible de :

- réaliser des **appareils de mesure programmables et modulables** (ex. pressionmètre, teslamètre, joulemètre, ...);
- 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

Les microcontrôleurs Arduino

2.1 Introduction

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



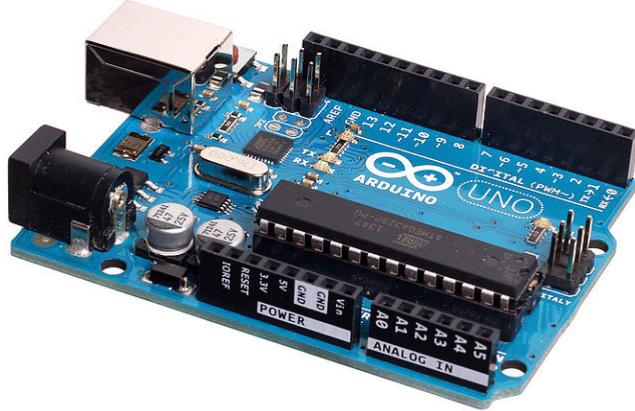
A l'origine conçue pour la **création artistique**, la carte Arduino trouve des applications dans des domaines d'applications aussi variés qu'insolites. Une carte Arduino s'utilise généralement comme :

- **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 est aussi 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 obtenu.

2.2 Les cartes Arduino pour les sciences physiques

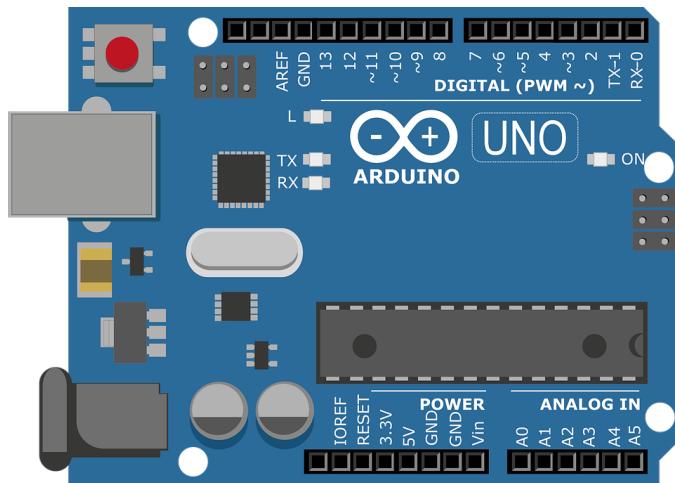
2.2.1 La carte Arduino UNO (Rev 3)



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.



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 !

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).

2.2.2 Les cartes spécifiques pour les sciences physiques

Il s'agit de cartes spécialement concues pour les sciences physiques avec des **protections sur les ports d'entrée/sortie** contre les mauvaises manipulations (ce type de protections n'existe pas sur les cartes classiques comme l'Arduino Uno). Ces cartes disposent de leurs **propres capteurs** avec une connectique particulière.

2.2.2.1 Educaduino Lab (Eurosmart)

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

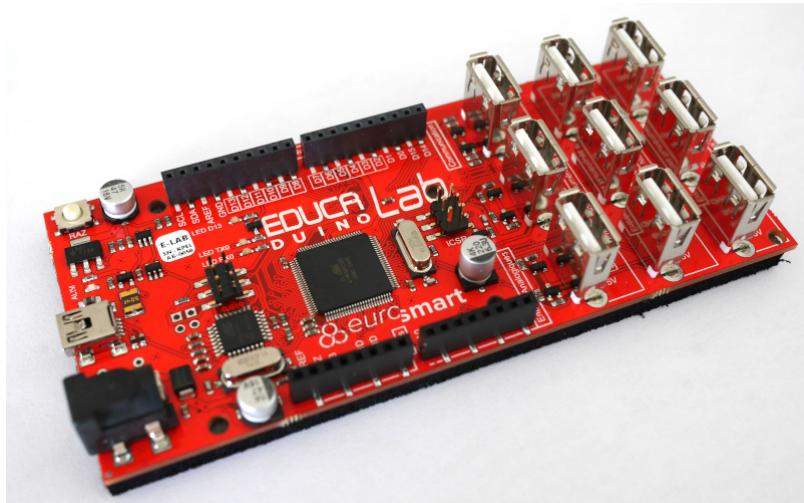


Fig. 1 – La carte Educaduino-Lab (E-LAB)

La carte **Educaduino Lab** a été concue 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 ;

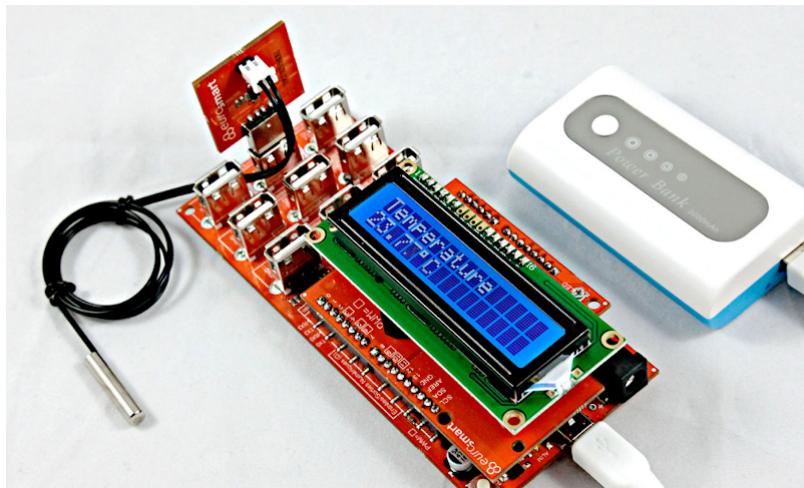


Fig. 2 – Mesure d'une température (image : Eurosmart)

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



Fig. 3 – Kit sciences-physiques 2nde/1ère (image : Eurosmart)

2.2.2.2 Plug'Uino® Uno (Sciencéthic)

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

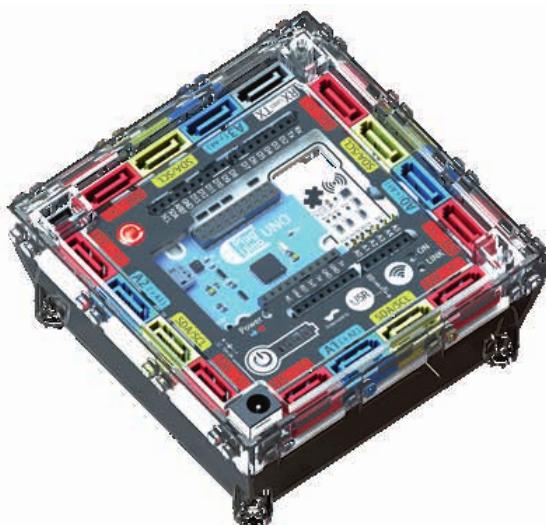


Fig. 4 – La carte Plug'Uino ® Uno (image : Sciencéthic)

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'Uino ;



Fig. 5 – Capteur de pression et loi de Mariotte (image : Sciencéthic)

2.3 Le logiciel Arduino

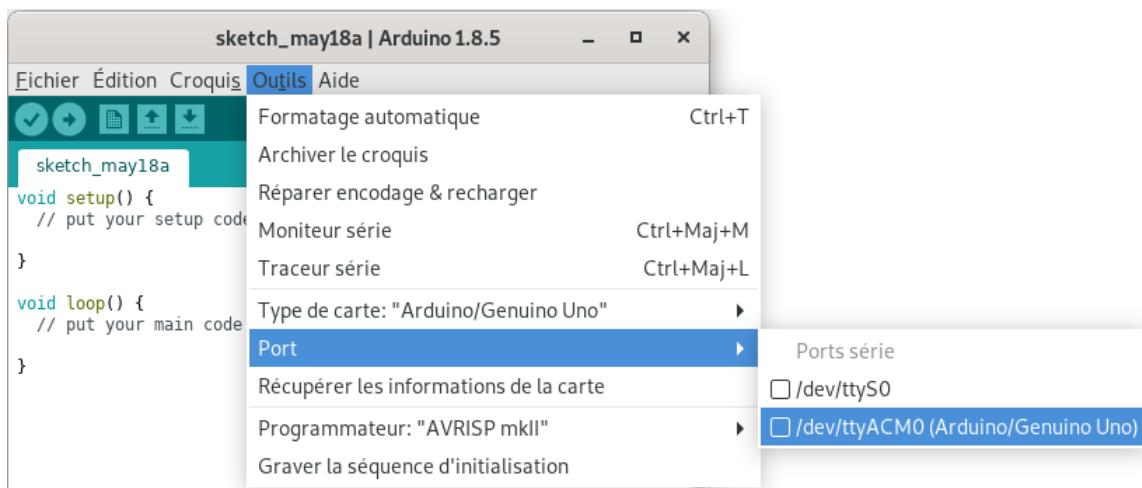
Le logiciel **Arduino** est un environnement intégré de développement (IDE) multiplateforme. Il est téléchargeable sur le site officiel <http://www.arduino.cc/en/>

```
sketch_may18a | Arduino 1.8.5
Fichier Édition Croquis Outils Aide
sketch_may18a
void setup() {
    // put your setup code here, to run once:
}
void loop() {
    // put your main code here, to run repeatedly:
}

Arduino/Genuino Uno sur COM1
```

Note : Arduino.cc propose une **version Web** (<https://create.arduino.cc/>) de son environnement de développement. Elle nécessite l'installation d'un plugin afin de programmer la carte par le port USB.

2.3.1 Choix du port de communication avec la carte Arduino



Avertissement : Pour programmer une carte Arduino, il est nécessaire de connecter cette dernière à l'ordinateur par l'intermédiaire d'un câble USB. Une fois le logiciel Arduino lancé, il est **impératif de sélectionner le port série** donnant accès à la carte Arduino (ex. COM3). Sinon il ne sera pas possible de téléverser le futur programme.

2.3.2 Mise en œuvre d'un projet Arduino :



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

1. **Édition** du programme dans l'éditeur de l'interface ;
2. **Vérification** du programme (compilation) ;
3. **Téléversement** du programme ;
4. **Exécution** sur la carte Arduino (de façon autonome sans ordinateur).

2.4 Premier programme : Blink

Le programme **Blink** propose de faire clignoter la LED intégrée à la carte de développement (connectée sur la broche 13).

2.4.1 Edition

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

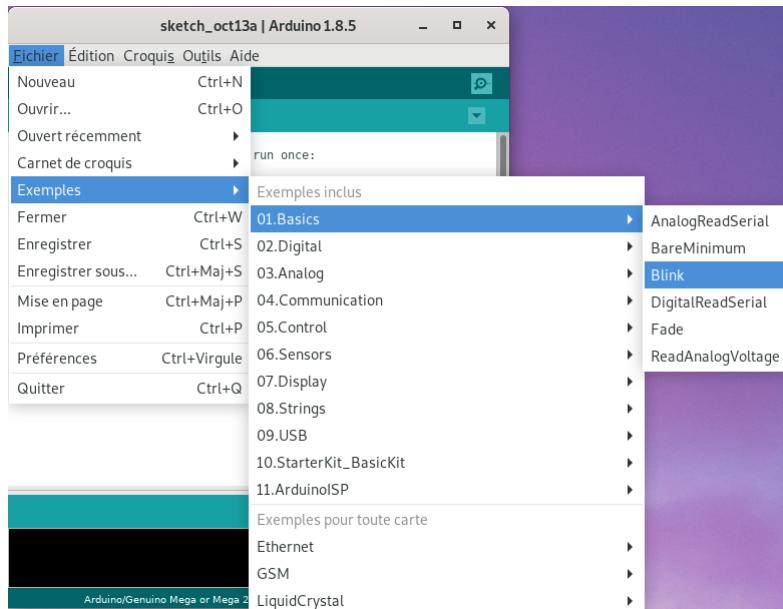


Fig. 6 – Ouvrir le programme Blink

```

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

```

Fig. 7 – Edition du programme Blink

Note :

- Un programme Arduino écrit en langage C/C++ est composé d'une suite d'instructions.
- Ces instructions sont effectuées 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.

```

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
}

```

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

Fig. 8 – Une version modifiée 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).

2.4.2 Compilation

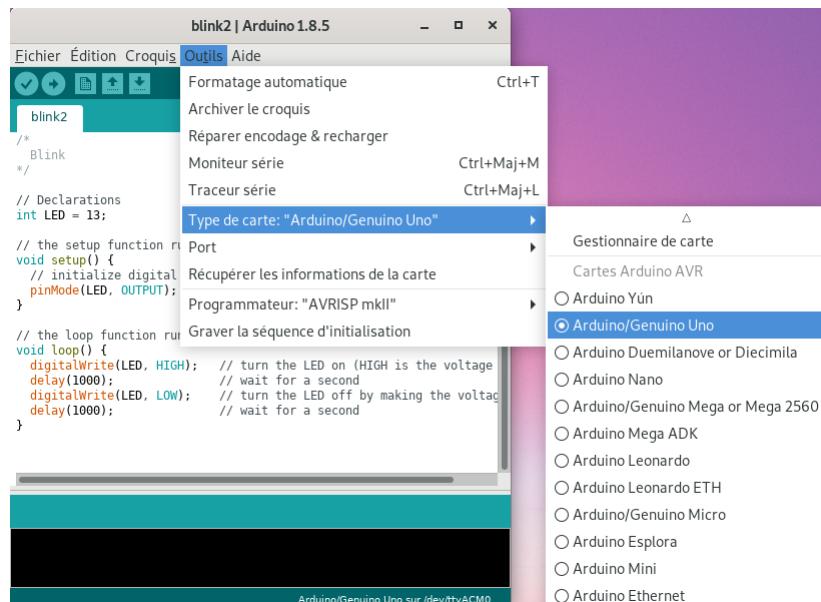


Fig. 9 – Choix du type de carte

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.

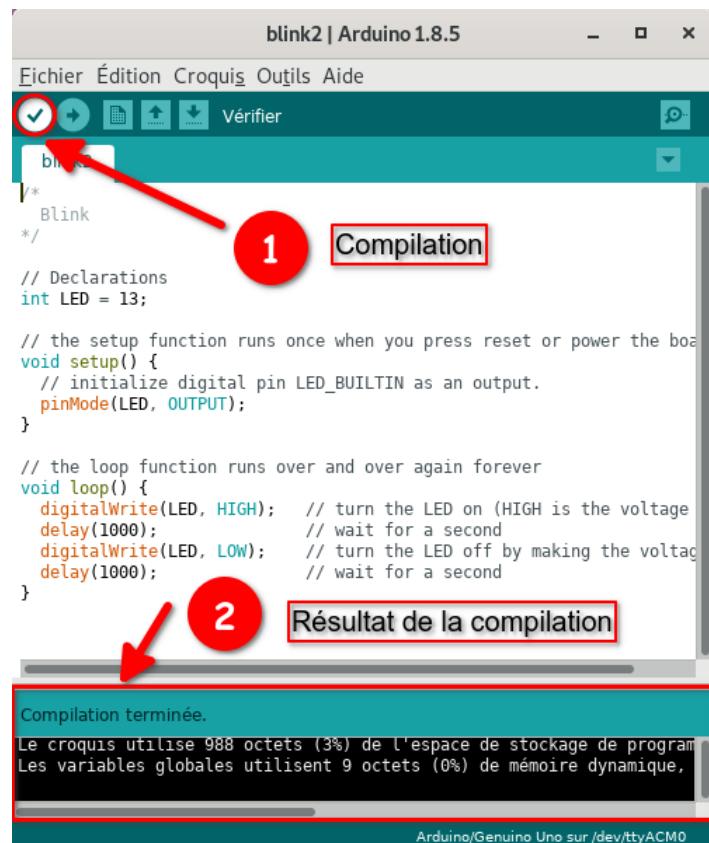


Fig. 10 – Puis la compilation peut s'effectuée !

2.4.3 Téléversement

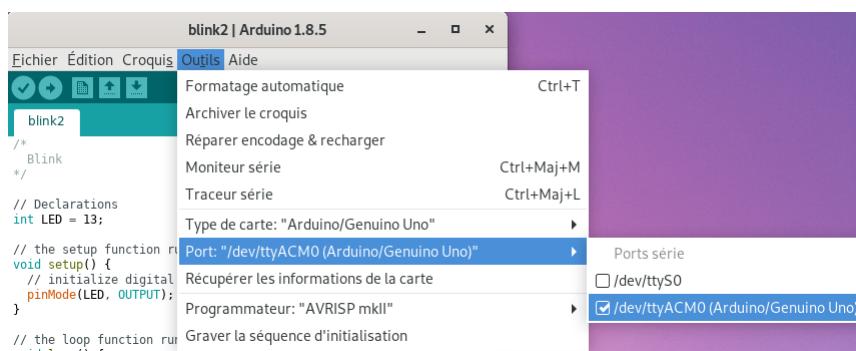


Fig. 11 – Choix du port de communication

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

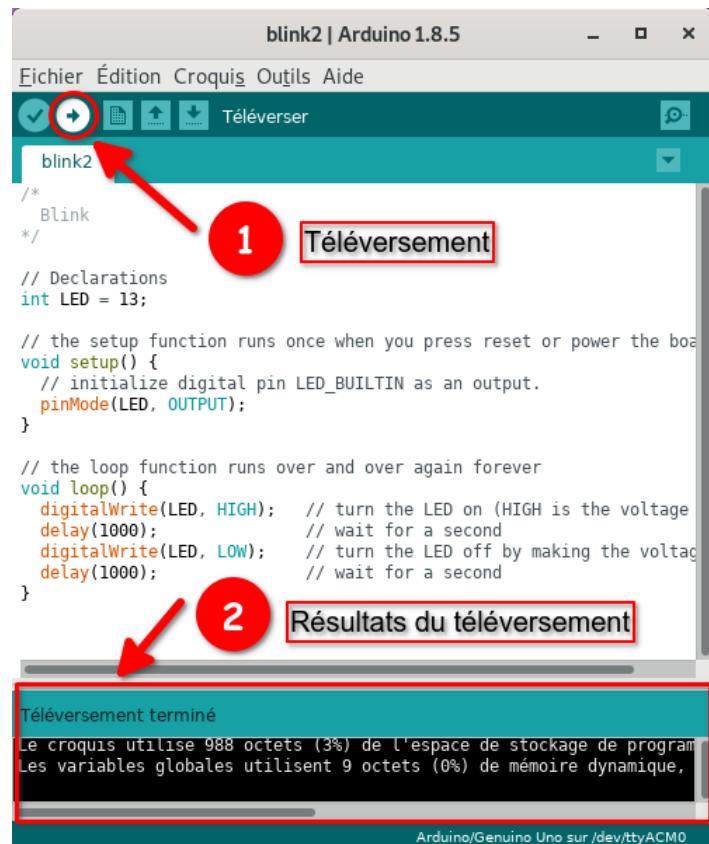
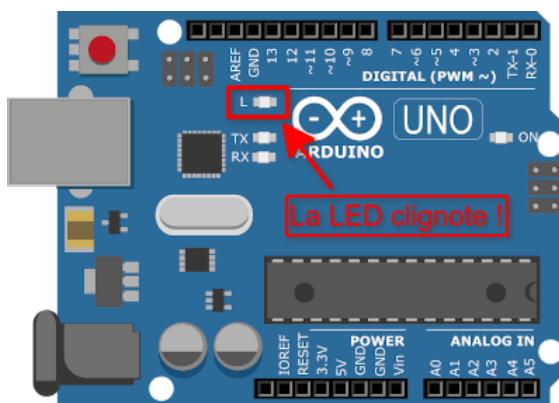


Fig. 12 – Téléversement du programme

2.4.4 Exécution

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



2.5 Particularité du langage Arduino

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



The screenshot shows the Arduino IDE interface. The top menu bar includes "Fichier", "Édition", "Croquis", "Outils", and "Aide". Below the menu is a toolbar with icons for file operations like Open, Save, and Print. The main workspace displays the following C++ code:

```

Eichier 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" lines of code and "Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) sur /dev/ttyACM0".

2.5.1 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.

2.5.2 Typage des variables

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

Quelques types disponibles :

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

Exemples :

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

2.5.3 Constantes prédefinies

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

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

2.5.4 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).

2.6 Pilotage d'une carte Arduino en Python avec Nanpy

2.6.1 Qu'est-ce que Nanpy ?

Nanpy est une librairie pour Python utilisée pour le **pilotage** d'une carte Arduino par le câble USB (port série).

<https://nanpy.github.io/>

2.6.2 Principe de fonctionnement

La carte Arduino a été préalablement programmer avec le firmware **Nanpy** 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.

Avertissement : Il est important de retenir que **Nanpy 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 !

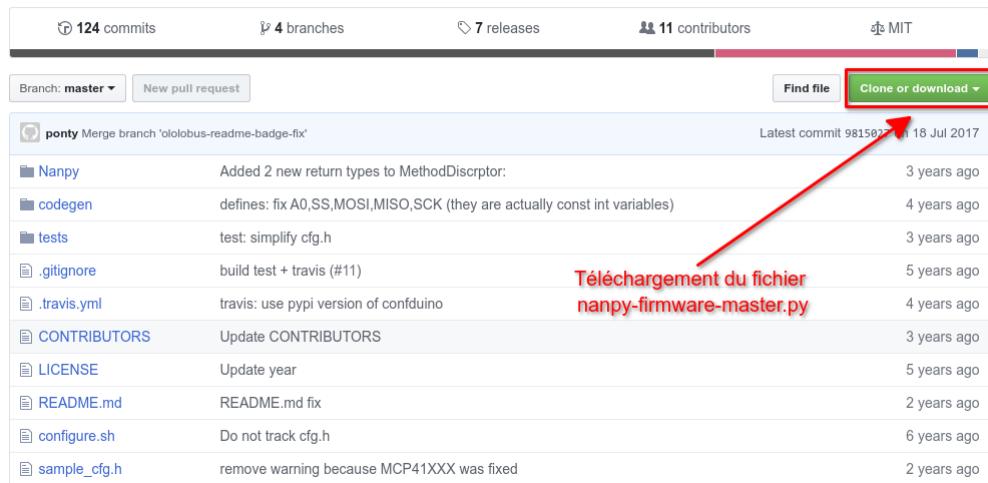
2.6.3 Installation de firmware Nanpy sur une carte Arduino

2.6.3.1 Téléchargement du firmware

Le fichier **nanpy-firmware-master.zip** est une librairie Arduino contenant le firmware Nanpy à installer sur la carte Arduino.

Ce fichier est à télécharger sur le site <https://github.com/nanpy/nanpy-firmware>.

Nanpy firmware repository

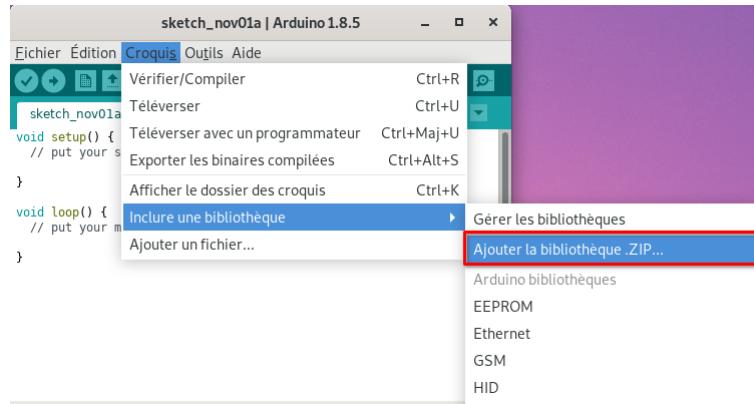


The screenshot shows the GitHub repository page for 'Nanpy'. At the top, it displays statistics: 124 commits, 4 branches, 7 releases, 11 contributors, and MIT license. Below this, a dropdown shows 'Branch: master' and a 'New pull request' button. A 'Find file' button is followed by a prominent green 'Clone or download' button with a downward arrow icon. To the right of the button, a red arrow points towards it from the text 'Téléchargement du fichier nanpy-firmware-master.py'. The main area lists commits from various authors, with the latest commit being 'Merge branch 'ololobus-readme-badge-fix'' by 'ponty' on Jul 18, 2017.

Branch: master	New pull request	Find file	Clone or download
ponty	Merge branch 'ololobus-readme-badge-fix'	Latest commit 9815027 18 Jul 2017	
Nanpy	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	

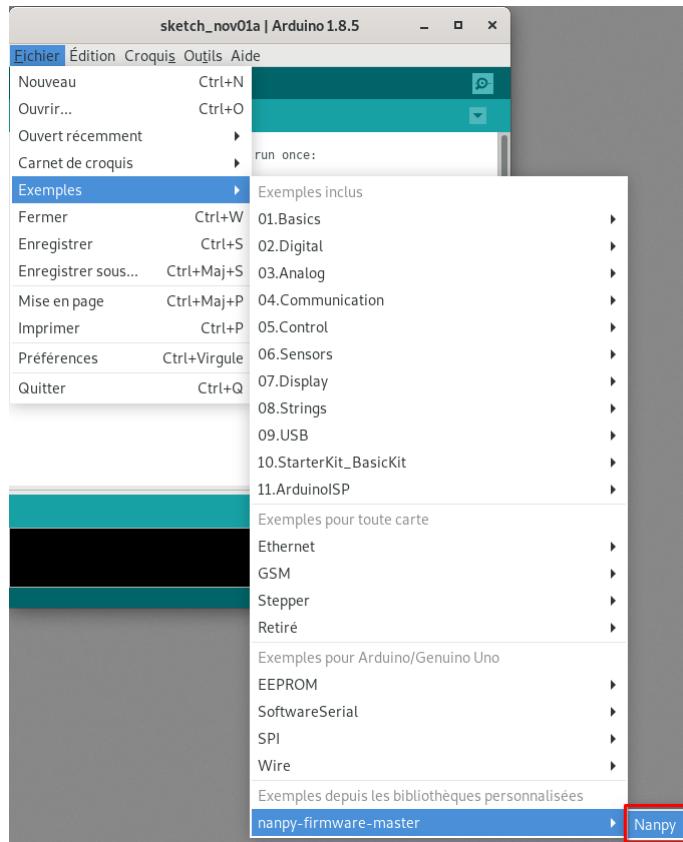
2.6.3.2 Installation du firmware dans le logiciel Arduino

Le fichier `nanpy-firmware-master.zip` est à installer dans les librairies d'Arduino à partir du menu *Croquis > Include une bibliothèque > Ajouter la bibliothèque au format .Zip ...*.

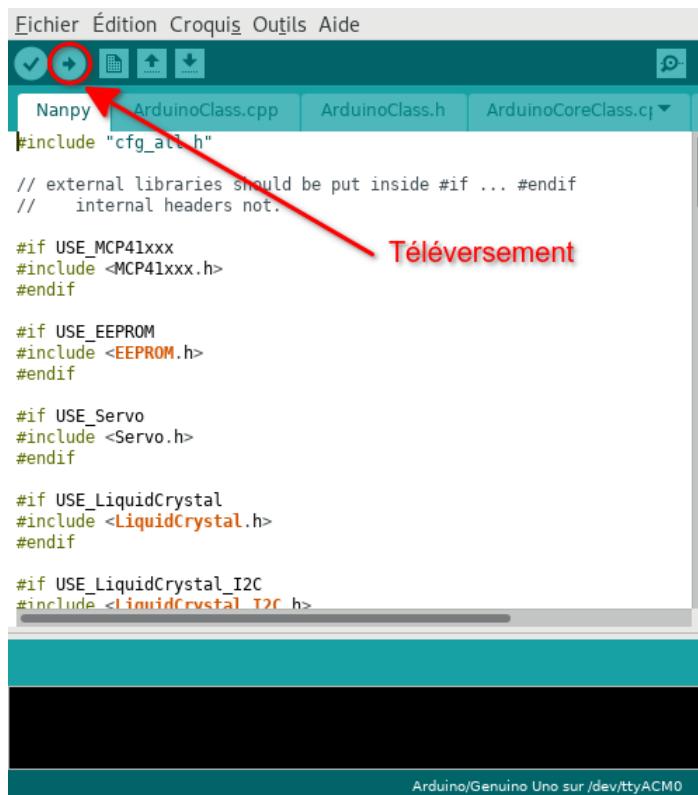
**2.6.3.3 Téléversement du firmware dans la carte Arduino**

Ouvrir le fichier d'exemple Numpy à partir du menu :

Fichier > Exemples > nanpy-firmware-master > Numpy



Puis téléverser le programme dans la carte Arduino.



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 !

La carte est maintenant prête pour un fonctionnement avec Numpy !

2.6.4 Installer la librairie Numpy sur l'ordinateur

L'installation dépend de la distribution Python utilisée sur l'ordinateur.

2.6.4.1 Pour EduPython

<https://edupyter.tuxfamily.org/>

Il n'y a rien à faire car la librairie Numpy est installée par défaut.

2.6.4.2 Pour les autres distributions

Il faut installer manuellement à partir du dépôt internet Pypi (<https://pypi.org/project/numpy/>) à l'aide de la commande pip :

```
pip install numpy
```

2.6.5 Exemple : Blink

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

```
from numpy import ArduinoApi          # Librairie du gestion des carte Arduino
from numpy import SerialManager        # Librairie de gestion du port série
from time import sleep               # Importation fonction sleep()

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

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

while True:                          # Boucle infinie
    uno.digitalWrite(pinLed, 1)       # Led allumée
    sleep(1)                         # Attendre 1 s
    uno.digitalWrite(pinLed, 0)       # Led éteinte
    sleep(1)                         # Attendre 1 s
```

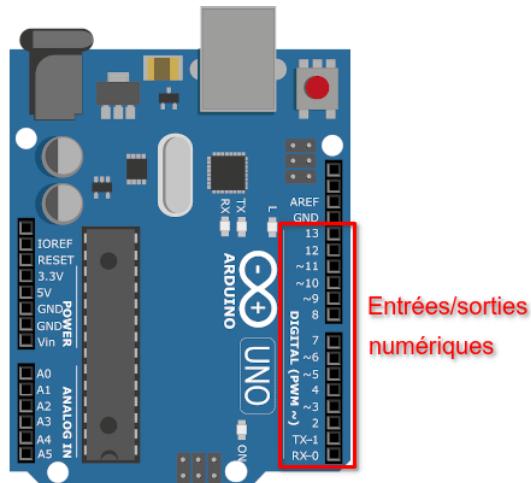

Chapitre 3

Les bases

3.1 Allumer une LED (sorties numériques)

3.1.1 Principe

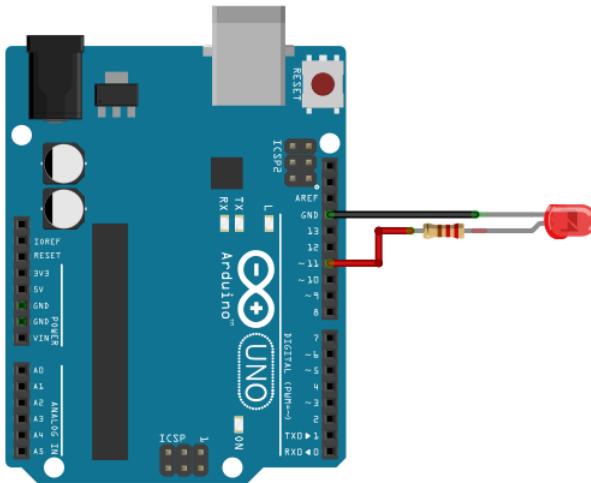
Les entrées/sorties numériques de la carte Arduino UNO sont accessibles sur les broches 0 à 13.



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 !

3.1.2 Montage

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



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 $220\ \Omega$ est un bon compromis.

3.1.3 Programme

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

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

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

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.

3.1.4 A retenir

Une **s**ortie **n**umé**q**uique présente deux états logiques : état bas (0 V) ou état haut (5 V).

Instruction	Description
<code>pinMode(pin, OUTPUT)</code>	Configure la broche <code>pin</code> en sortie
<code>digitalWrite(pin, LOW)</code>	Met la broche <code>pin</code> à l'état bas (0 V)
<code>digitalWrite(pin, HIGH)</code>	Met la broche <code>pin</code> à l'état haut (5 V)

3.1.5 Applications

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

3.1.6 Avec Python et Nanpy

```

from nanpy import ArduinoApi          # Librairie du gestion des carte Arduino
from nanpy import SerialManager       # Librairie de gestion du port série
from time import sleep               # Importation fonction sleep()

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

pinLed = 11                         # N° de broche où la Led est branchée
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

```

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

3.2.1 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.

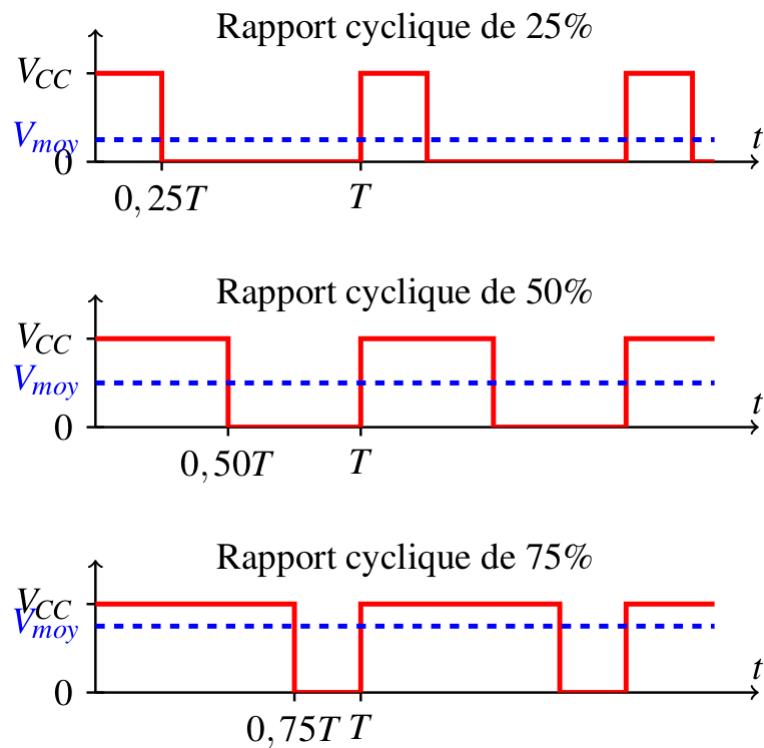
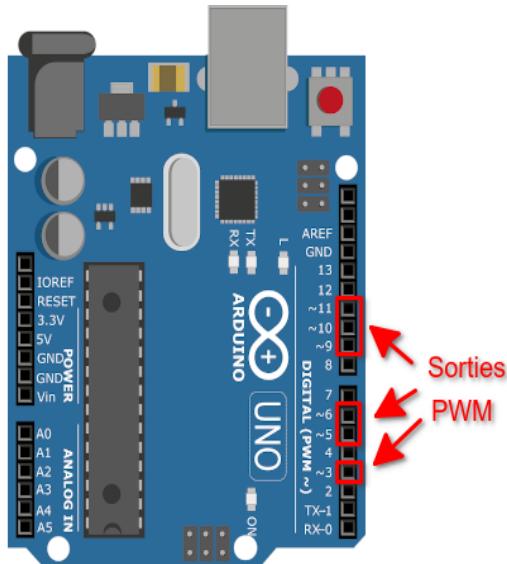


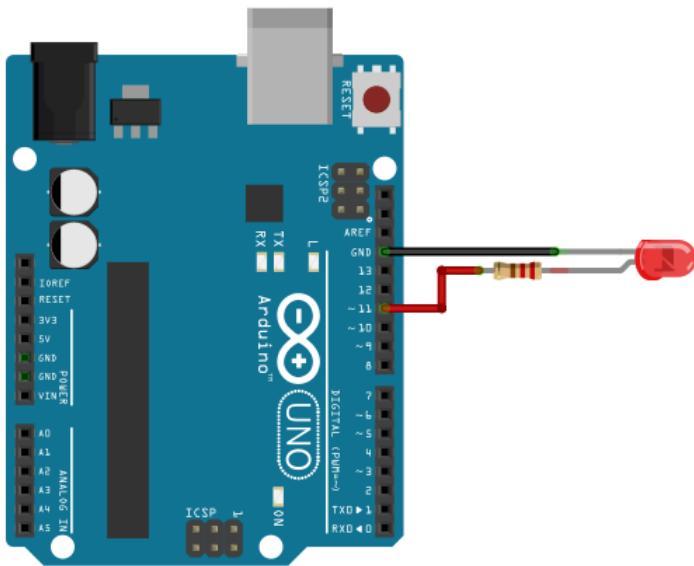
Fig. 1 – Principe de la MLI ou PWM

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



Avertissement : La fréquence d'un signal PWM est fixée à 490 Hz !

3.2.2 Montage



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

3.2.3 Programme

```
#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.
- L'argument `duty` est un nombre entier entre 0 et 255 respectivement pour un rapport cyclique entre 0% et 100%.

3.2.4 A retenir

Seules les broches 3, 5, 6, 9, 10 et 11 de l'Arduino Uno permettent de générer des signaux PWM.

Instruction	Description
<code>pinMode(pin,OUTPUT)</code>	Configure le broche <code>pin</code> en sortie
<code>analogWrite(pin,duty)</code>	Génère une tension PWM sur le broche <code>pin</code> avec le rapport cyclique <code>duty</code> (0 à 255)

3.2.5 Applications

- Variation de l'intensité lumineuse d'une LED.

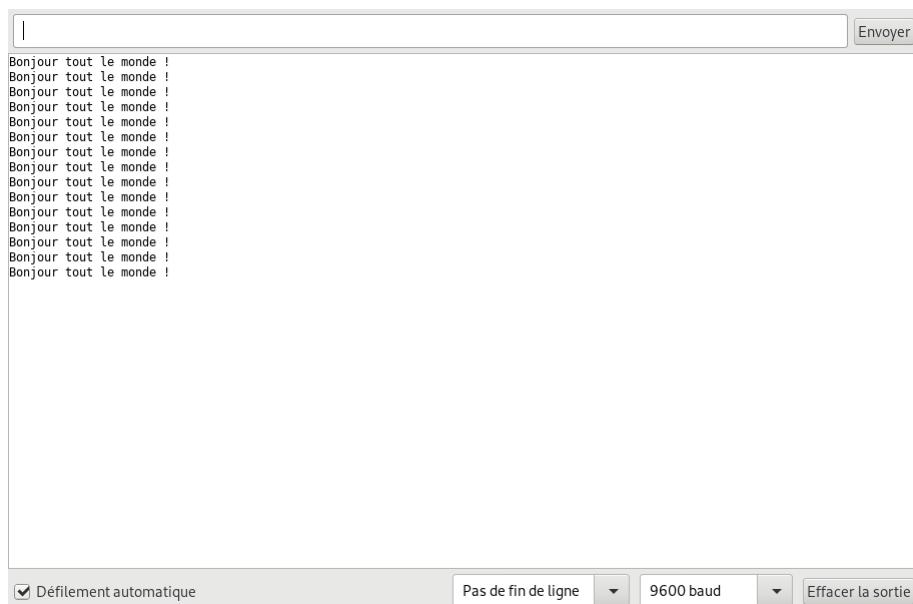
- Variation de la vitesse d'un moteur à courant continu.
- Obtention d'une tension constante par filtrage passe bas (limitée en fréquence).

3.3 Communication avec un ordinateur (port série)

3.3.1 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 communiquer avec une carte Arduino.

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



Le moniteur série permet également de transmettre des données vers le microcontrôleur !

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

3.3.2 Montage

Une carte Arduino est connectée par le port USB à un ordinateur.

3.3.3 Programme

```
/*
 * Affichage le contenu d'une variable dans un moniteur série.
 */

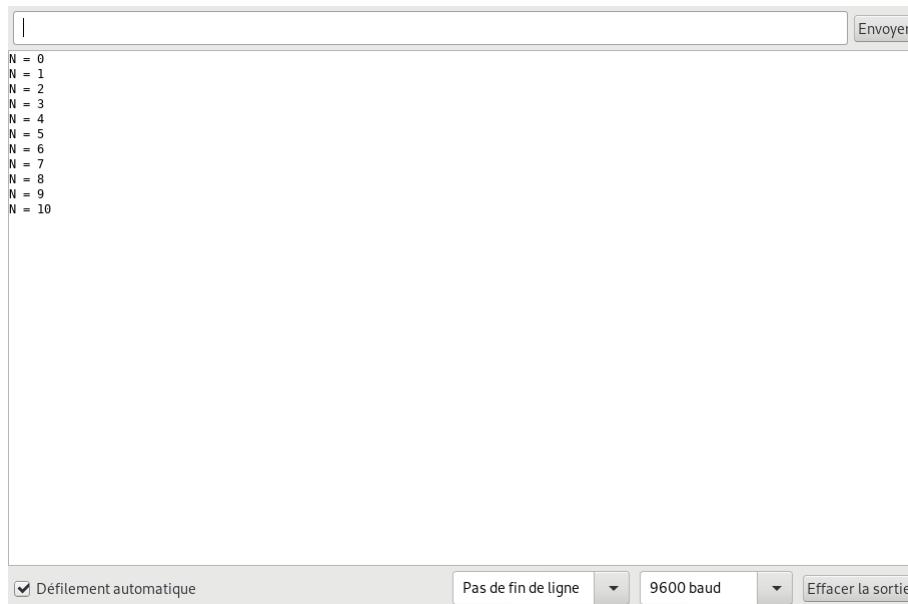
int n = 0;

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

(suite sur la page suivante)

(suite de la page précédente)

```
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
}
```



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

3.3.4 A retenir

Instruction	Description
<code>Serial.begin(speed)</code>	Fixe la vitesse de transmission en bits par seconde (baud)
<code>Serial.print()</code>	Ecrit des données (format texte) sur le port série
<code>Serial.println()</code>	Ecrit des données (format texte) sur le port série suivi d'un saut de ligne

3.3.5 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 ou des logiciels spécialisés tels que Regressi, Latis, ...

3.4 Afficheur LCD

Ce sont généralement des afficheurs génériques de 32 caractères disposés sur 2 lignes. Ils ont la particularité d'être simple à mettre en œuvre.

3.4.1 Afficheur LCD 16x2 (port parallèle)

3.4.1.1 Principe

Le pilotage d'un afficheur LCD 16x2 nécessite de 6 broches numériques.

Pour éviter un câblage trop complexe, le plus simple est de fixer sur la carte de développement un « shield » afficheur.



Fig. 2 – Afficheur Educaduino-Lab (image : Eurosmart)

Le logiciel Arduino dispose de la librairie `LiquidCrystal` (installée par défaut) pour piloter ce type d'afficheur.

3.4.1.2 Montage

3.4.1.3 Programme

```
/*
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);

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 !");
}
```

3.4.1.4 À retenir

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

3.4.2 Afficheur LCD 16x2 (I2C)

3.4.2.1 Principe

Ce sont les mêmes afficheurs que précédemment mais avec un port série de données (I2C) nécessitant moins de câbles (4 en tout) !

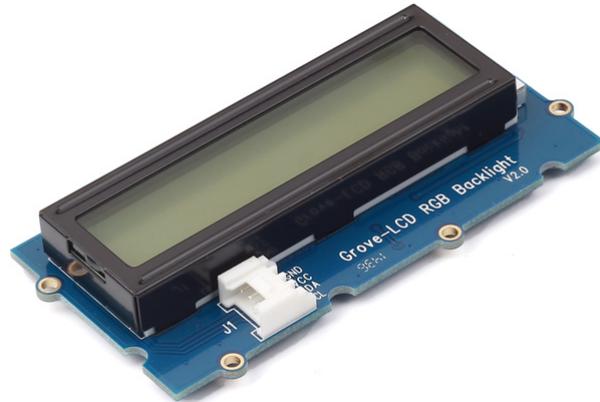


Fig. 3 – Module Grove - LCD RGB Backlight (image : <http://wiki.seeedstudio.com>)

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.

3.4.2.2 Montage



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

3.4.2.3 Programme

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

rgb_lcd lcd;

const int colorR = 255;
const int colorG = 0;
const int colorB = 0;
```

(suite sur la page suivante)

```

void setup()
{
    // set up the LCD's number of columns and rows:
    lcd.begin(16, 2);

    lcd.setRGB(colorR, colorG, colorB);

    // Print a message to the LCD.
    lcd.print("hello, world!");

    delay(1000);
}

void loop()
{
    // set the cursor to column 0, line 1
    // (note: line 1 is the second row, since counting begins with 0):
    lcd.setCursor(0, 1);
    // print the number of seconds since reset:
    lcd.print(millis()/1000);

    delay(100);
}

```

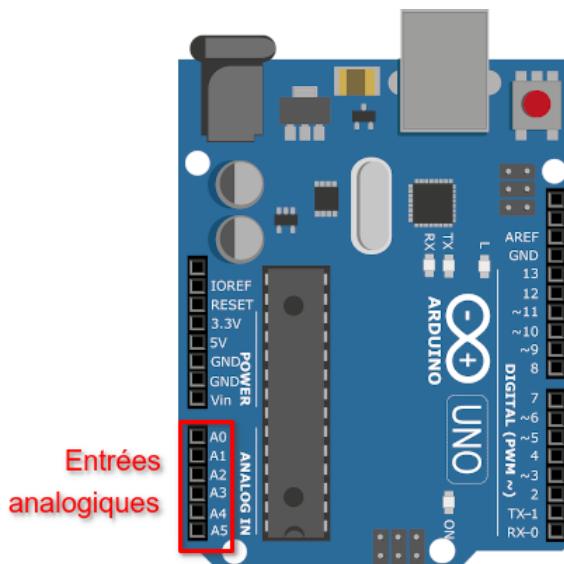
3.5 Mesurer une tension (CAN)

3.5.1 Principe

La mesure d'une tension par un microcontrôleur est réalisée en interne par un **convertisseur analogique-numérique**.

Ce type de conversion est importante en sciences physique. Elle permet par exemple d'obtenir la mesure d'une grandeur physique provenant d'un **capteur**.

Arduino UNO dispose de **6 entrées analogiques** disponibles sur les broches A0 à A5. Par défaut, la tension en entrée doit-être comprise entre 0 V et 5 V (Vref). La résolution de la conversion est de 10 bits.



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

3.5.2 Montage

Un pont diviseur de tension branché entre la masse (GND) et la tension d'alimentation (5V) délivre une tension réglable sur l'entrée A0.

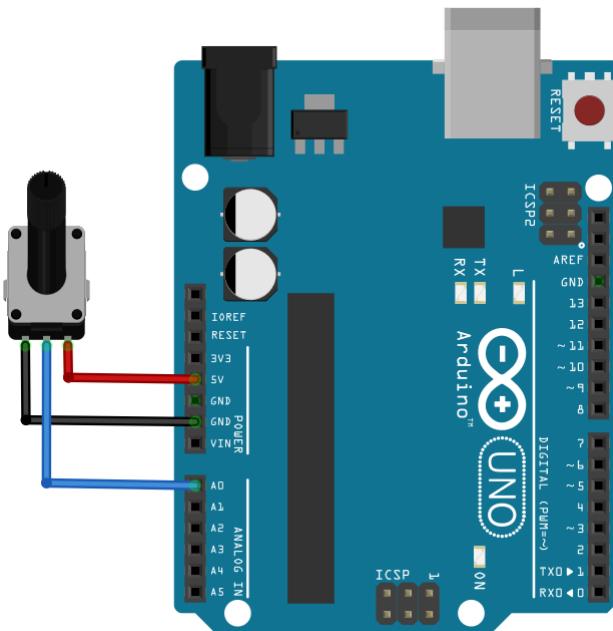


Fig. 5 – Montage potentiométrique

3.5.3 Programme

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

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

int valAnalog;      // Entier compris entre 0 et 1023 (10bits)
float tension;      // La tension calculée est un nombre à virgule

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

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

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

3.5.4 A retenir

Instruction	Description
<code>analogRead(A0)</code>	Retourne un entier entre 0 et 1023 proportionnel à la tension appliquée

3.5.5 Applications

- Interface avec un circuit comportant un capteur.
- Mesure d'une position (potentiomètre).

Chapitre 4

Nouveaux programmes du lycée

4.1 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.

4.1.1 Principe

Un capteur résistif est un composant électronique dont la résistance varie en fonction de la grandeur physique à mesurer. Par exemple, une CTN est un **capteur résistif à coefficient de température négatif**.

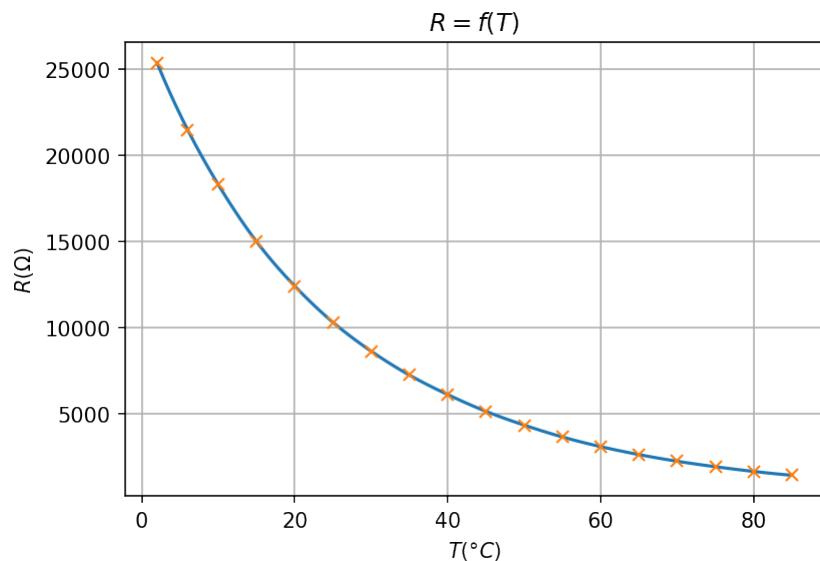
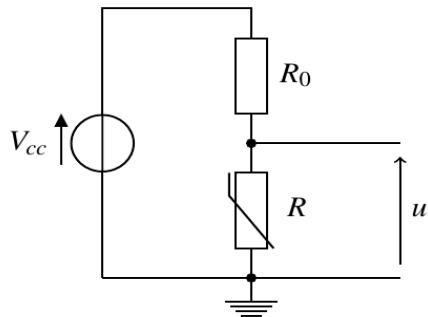


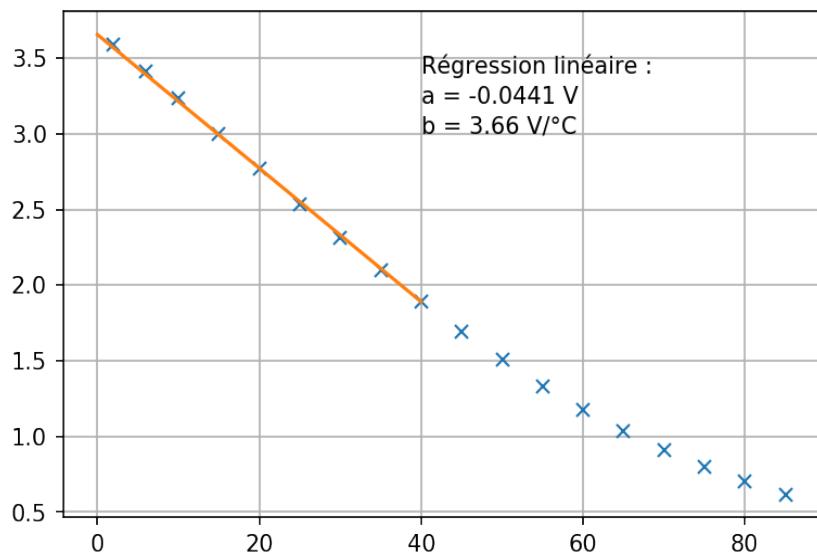
Fig. 1 – Courbe d'étalonnage d'une CTN 10k

Note : Pour 25°C la résistance mesurée est égale à $10\text{ k}\Omega$!

Pour une tension qui varie en fonction de la résistance donc de la température, la solution la plus simple est de placer la CTN dans un pont diviseur de tension.



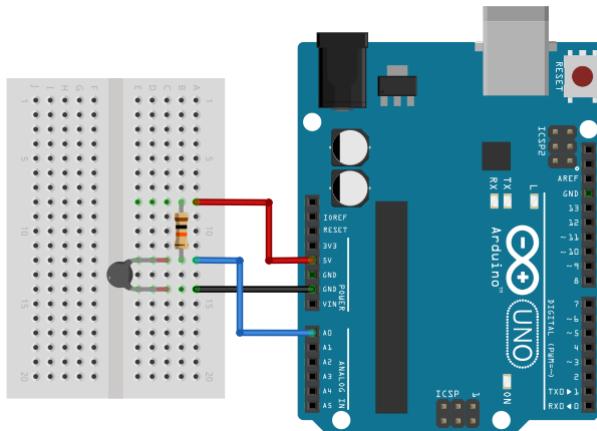
En choisissant la résistance R_0 également à $10\text{ k}\Omega$, la tension obtenue est partiellement linéarisée.



Entre 0 et 40°C, la température en fonction de la tension est donnée par la relation suivante :

$$T = \frac{u - b}{a}$$

4.1.2 Montage



4.1.3 Programme

```
/*
 * Mesure d une température avec une CTN 10k (25°C)
 * placée dans un pont diviseur de tension avec
 * une résistance de 10k.
 */

float tension;
int temperature; // Arrondi à l entier
float a = -0.0441; // Coeff. directeur modèle
float b = 3.66; // Ordonnée à l origine modèle

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

void loop() {
    tension = analogRead(A0)*5.0/1023; // Lecture tension
    temperature = (tension-b)/a; // Calcul température
    Serial.print("U = "); // Affichage dans moniteur série
    Serial.println(tension);
    Serial.print("T= ");
    Serial.println(temperature);
    delay(1000); // Temporisation d une seconde
}
```

4.1.4 A retenir

Placer un **capteur résistif** (température, pression, lumière, ...) dans un **pont diviseur de tension** reste une solution simple d'interfacing avec un microcontrôleur. Mais pas la plus efficace !

4.1.5 Allez plus loin

4.1.5.1 Mesurer la résistance de la CTN

Dans le pont diviseur de tension, la résistance de la CTN s'exprime par la relation suivante :

$$R = R_0 \cdot \frac{u}{V_{cc} - u}$$

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

#define Vcc 5 // Tension d'alimentation
#define Ro 10000 // Résistance du pont

float u; // Tension CTN
float R; // Résistance CTN

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

void loop() {
    u = analogRead(A0)*5.0/1023; // Lecture tension
```

(suite sur la page suivante)

(suite de la page précédente)

```
R = Ro * u/(Vcc-u);           // Calcul de la résistance
Serial.print("U = ");         // Début affichage
Serial.println(u);
Serial.print("R = ");
Serial.println(R);            // Fin affichage
delay(1000);                 // Temporisation en milli seconde
}
```

4.1.5.2 Mesure de la température avec la relation de Steinhart-Hart

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

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

A, B et C sont les coefficients de Steinhart-Hart. Ils sont donnés par le constructeur ou peuvent se déterminer expérimentalement à partir de trois points de mesure.

Note : Un programme Python pour déterminer ces trois coefficients est disponible sur Wikipédia (https://fr.wikipedia.org/wiki/Relation_de_Steinhart-Hart).

```
/*
 * Mesure de la température avec la relation de Steinhart-Hart
 */

#define Vcc 5          // Tension d'alimentation
#define Ro 10000       // Résistance du pont
#define A 1.0832e-3
#define B 2.1723e-4
#define C 3.2770e-7

float u;    // Tension CTN
float R;    // Résistance CTN
float logR;
float T;

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

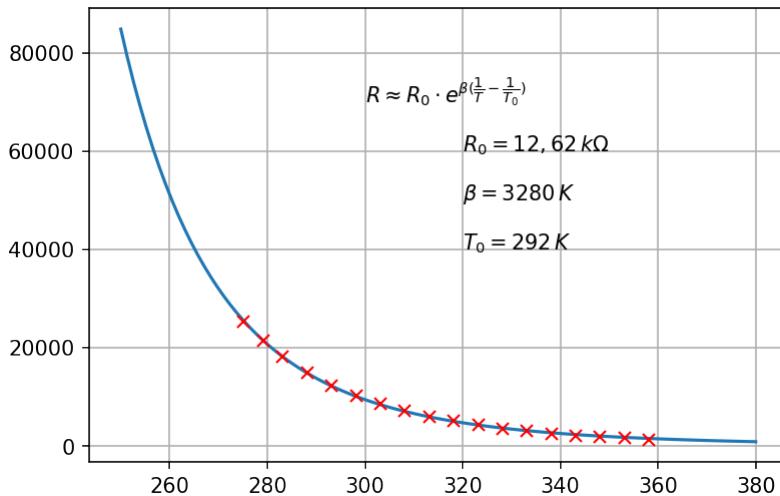
void loop() {
    u = analogRead(A0)*5.0/1023;           // Lecture tension
    R = Ro * u/(Vcc-u);                  // Calcul de la résistance
    logR = log(R);                      // Calcul de ln(R)
    T = (1.0 / (A + B*logR + C*logR*logR*logR)); // Calcul de la température
    T = T - 273.15;                     // Conversion en °C
    Serial.print("R = ");                // Début affichage
    Serial.println(R);
    Serial.print("T = ");
    Serial.println(T);                  // Fin affichage
    delay(1000);                      // Temporisation en mille seconde
}
```

4.1.5.3 Simplification de relation de Steinhart-Hart

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

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

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



La détermination de la température (en K) s'effectue à l'aide de la relation suivante :

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

4.2 Émission d'un son (seconde générale)

Programme de seconde générale 2019 - Enseignement commun

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

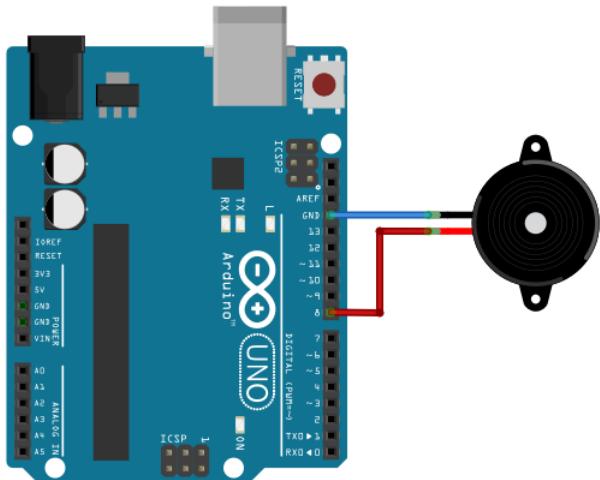
4.2.1 Principe

Les microcontrôleurs Arduino ne possèdent 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 V et 5 V) 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 $3f, 5f, 7f, \dots$

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

4.2.2 Montage



4.2.3 Programme

4.2.3.1 Méthode 1 : construire le signal carré

```
/*
 * Générer un son simple
 */

#define brocheHP 44

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

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

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

4.2.3.2 Méthode 2 : avec la fonction tone()

```
/*
 * Génération d'un son
 */

#define Do3 262
#define Re3 294
#define Mi3 330
#define Fa3 349
#define Sol3 392
#define La3 440
#define Si3 494
```

(suite sur la page suivante)

(suite de la page précédente)

```
#define brocheHP 44

void setup() {
    pinMode(brocheHP, OUTPUT);
    tone(brocheHP, La3, 4000);
}

void loop() {
}
```

4.3 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.

4.3.1 Principe

4.3.1.1 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.



Fig. 2 – Module HC-SR04

4.3.1.2 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.

4.3.1.3 Chronogrammes

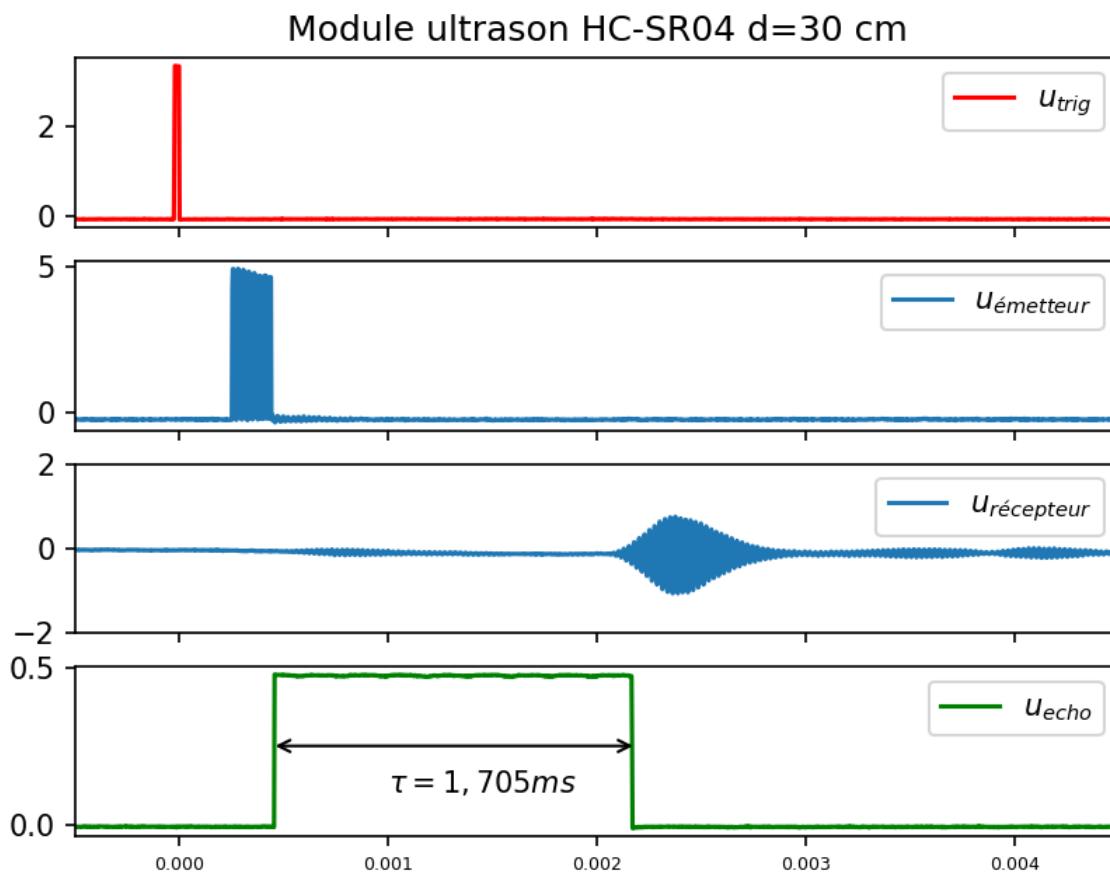
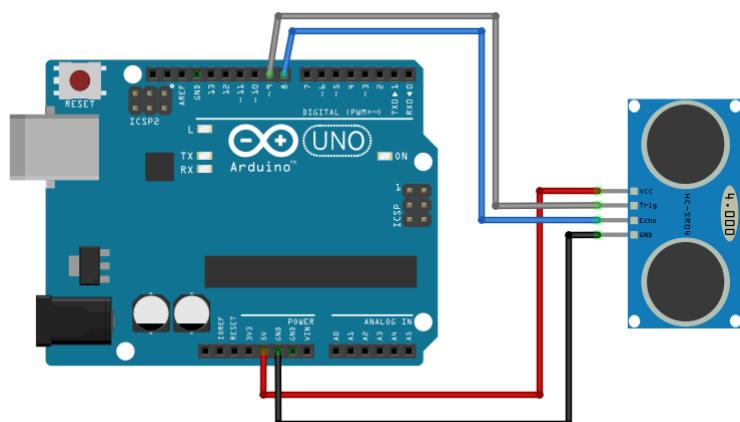


Fig. 3 – Mesures réalisées à l'oscilloscope sur un module HC-SR04 (5 V)

Dans cet exemple, le calcul de la célérité du son donne 352 m/s.

4.3.2 Montage



4.3.3 Programme

```
/*
 * Mesurer vitesse son
 */

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

float distance = 0.3;    // Distance en module et réflecteur
long dureeEcho;          // Durée mesurée
float vitesse;           // 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
    vitesse = 2*distance/dureeEcho * 1E6; // Calcul de la vitesse
    Serial.print("Durée (s) = ");
    Serial.println(dureeEcho);
    Serial.print("Vitesse (m/s) = ");
    Serial.println(vitesse);
    delay(1000);                  // Attendre 1s
}
```

4.3.4 A retenir

Les modules du type HC-SR04 délivre une **impulsion à l'état haut** dont la durée est égale au temps que prend le son pour partir de l'émetteur puis revenir au récepteur.

Dans le programme la fonction `pulseIn(pin,HIGH)` est chargée de la **mesure de cette durée**.

4.3.5 Aller plus loin

Pour améliorer la précision, il est possible de réaliser plusieurs mesures et d'en faire une moyenne ou même de tracer un histogramme de ces mesures !

```
/*
 * Exporter plusieurs mesures de la vitesse du son
 * au format CSV pour exploitation par tableur ou
 * logiciels spécialisés (Regressi, Latis, ...)
 */

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

float distance = 0.297; // Distance en module et réflecteur
long dureeEcho;          // Durée mesurée
int vitesse;             // Vitesse obtenue
int n=1;
```

(suite sur la page suivante)

```
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  
    Serial.println("n;v");        // Entête du fichier CSV  
}  
  
void loop() {  
    if (n<=20) {  
        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  
        vitesse = round(2*distance/dureeEcho*1E6); // Calcul de la vitesse  
        Serial.print(n);                   // Début d écriture d une ligne de mesure  
        Serial.print(";");  
        Serial.println(vitesse);  
        delay(100);                      // Attendre 1s  
        n++;  
    }  
}
```



Fig. 4 – Mesures au format CSV obtenues dans le moniteur série

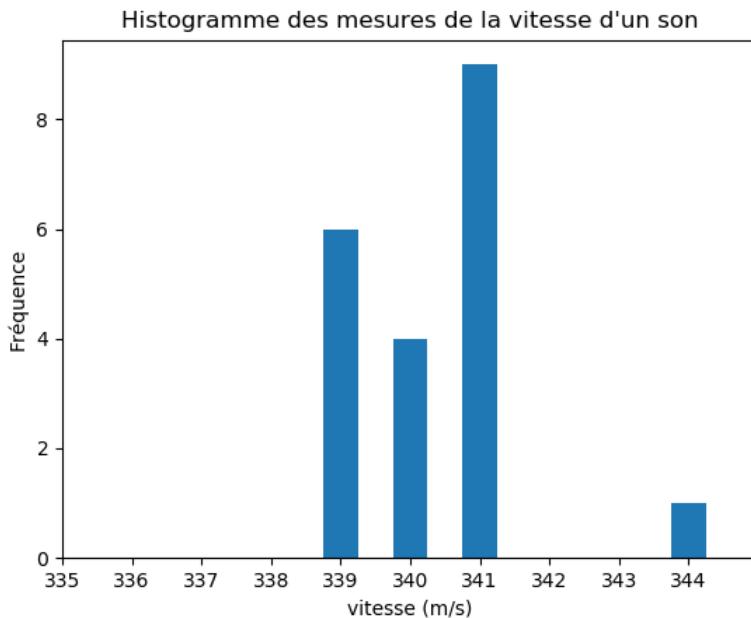


Fig. 5 – Histogramme des mesures tracé à l'aide du module `matplotlib` de Python

4.4 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.

4.4.1 Principe

La manipulation consiste à vérifier la loi de Mariotte $P \times V = constante$ (à température et quantité de matière constantes).

La mesure de pression s'effectue avec un capteur de pression absolue du type MPX5700 (700 kPa - 5V). La tension de sortie est linéaire à la pression mesurée.



Fig. 6 – Capteur MPX5700GP (0 à 700 kPa) (image : farnell.fr)

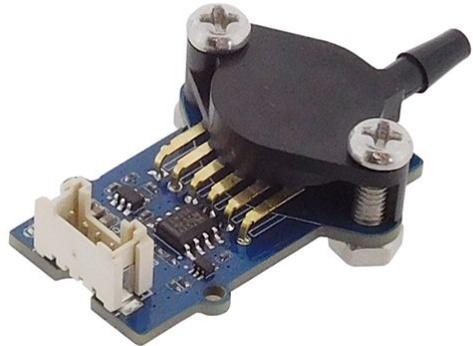
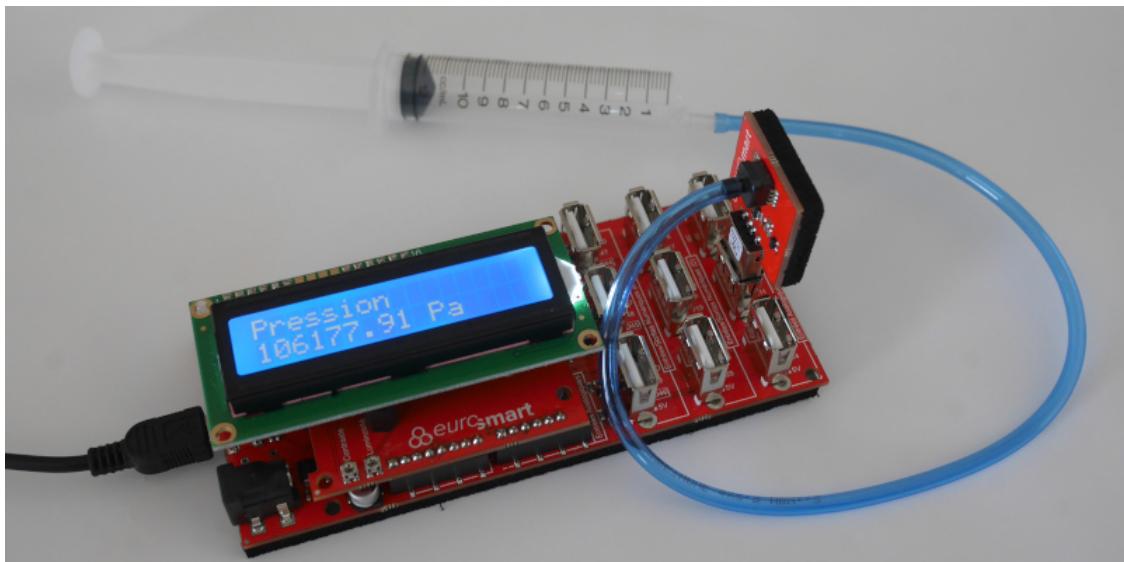


Fig. 7 – Capteur MPX5700AP Grove (15 kPa à 700 kPa) (image : gotronic.fr)

4.4.2 Montage

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



Note : Dans cette manipulation, il est important de tenir compte du volume d'air V_0 présent dans le tube. La loi de Mariotte s'écrit alors

$$P \times (V + V_0) = \text{constante}$$

4.4.3 Programme

```
/*
 * Mesure d une pression absolue
 * Capteur Educaduino 20 kPa à 400 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

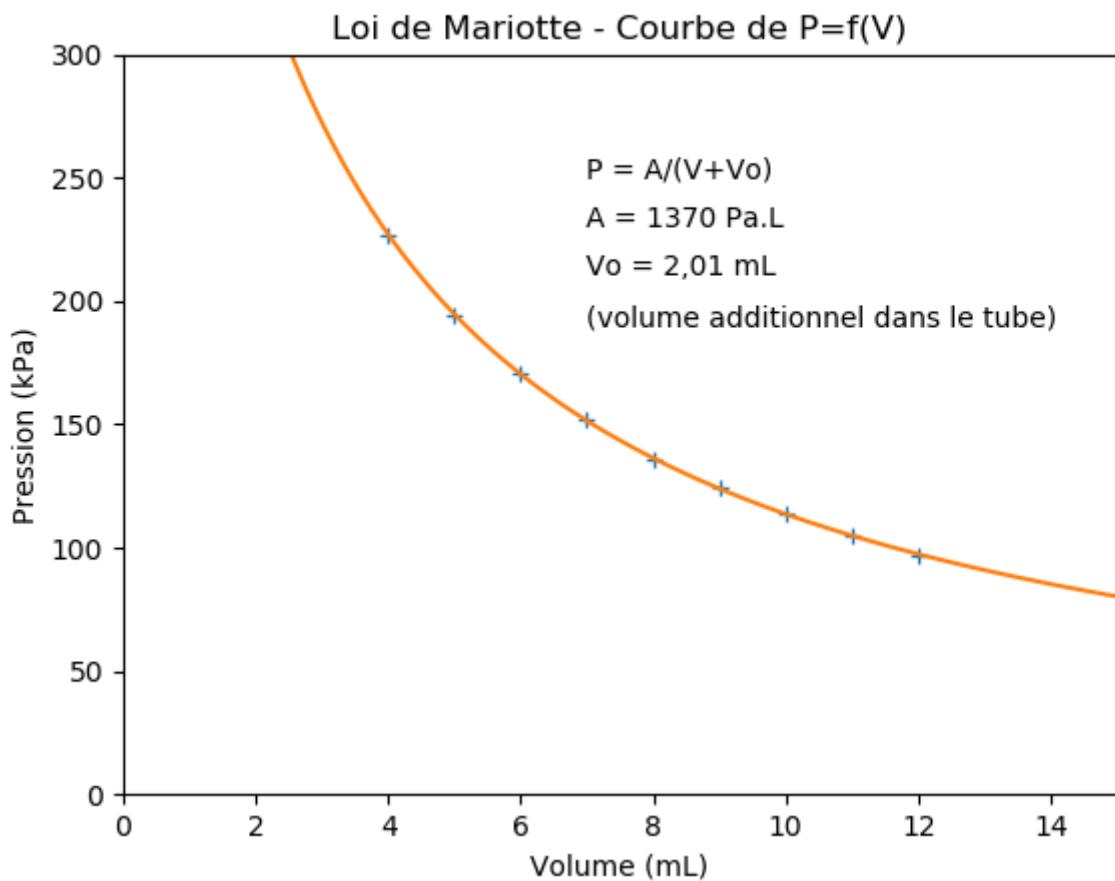
float tension ;                  // Tension mesurée
float pression ;                // Pression mesurée

void setup() {
    lcd.begin(16, 2);           // Paramétrage de l ecran LCD
}

void loop() {
    tension = analogRead(brocheCapteur)*5.0/1023 ;   // Lecture de la tension
    pression = tension * 76 + 20 ;                      // Calcul de la pression en kPa
    lcd.clear();                                         // Début affichage
    lcd.setCursor(0,0);
    lcd.print("Pression en kPa");
    lcd.setCursor(0,1);
    lcd.print(pression);                                // Fin affichage
    delay(1000);
}
```

4.4.4 Résultats

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



4.5 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.

4.5.1 Principe

4.5.2 Montage

4.5.3 Programme

4.5.4 A retenir

4.6 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.**

4.6.1 Principe**4.6.2 Montage****4.6.3 Programme****4.6.4 A retenir****4.7 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.

4.7.1 Principe**4.7.2 Montage****4.7.3 Programme****4.7.4 A retenir**

Chapitre 5

Aller plus loin

5.1 Adaptation d'un capteur teslamètre Jeulin

5.1.1 Principe

5.1.2 Montage

5.1.2.1 Programme

5.1.3 Applications

5.2 Module Gravity wattmètre/joulemètre

5.2.1 Principe

5.2.2 Montage

5.2.2.1 Programme

5.2.3 Applications

5.3 Acquisition de données (mode temporel)

5.3.1 Principe

5.3.2 Montage

5.3.2.1 Programme

5.3.2.2 Exploitations