

# [Arduino 5] Les capteurs et l'environnement autour d'Arduino

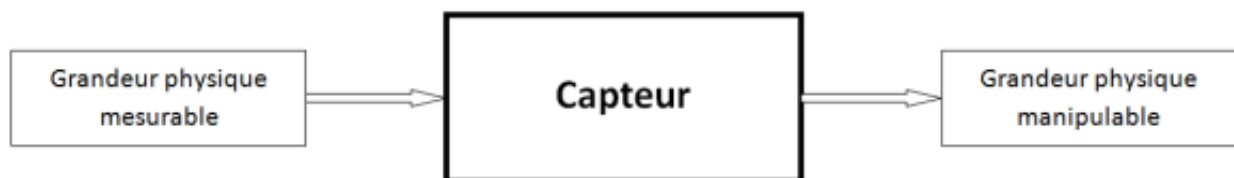
Savoir programmer c'est bien, mais créer des applications qui prennent en compte les événements de leur environnement, c'est mieux ! Cette partie va donc vous faire découvrir les capteurs couramment utilisés dans les systèmes embarqués. Alors oui, le bouton poussoir et le potentiomètre peuvent, en quelque sorte, être définis comme étant des capteurs, mais vous allez voir qu'il en existe plein d'autres, chacun mesurant une grandeur physique distincte et plus ou moins utile selon l'application souhaitée.

## [Arduino 501] Généralités sur les capteurs

Ce premier chapitre va vous présenter un peu ce que sont les capteurs, à quoi ils servent, où est-ce qu'on en trouve, etc. leur taille, leur forme et j'en passe. Je vais simplement vous faire découvrir le monde des capteurs qui, vous allez le voir, est merveilleux !

### Capteur et Transducteur

Un capteur est un dispositif capable de transformer une grandeur physique (telle que la température, la pression, la lumière, etc.) en une autre grandeur physique manipulable. On peut d'ailleurs prendre des exemples : un microphone est un capteur qui permet de transformer une onde sonore en un signal électrique ; un autre capteur tel qu'une photorésistance permet de transformer un signal lumineux en résistance variable selon son intensité.



Pour nous, utiliser un thermomètre à mercure risque d'être difficile avec Arduino, car ce capteur ne délivre pas d'information électrique ou de résistance qui varie. Il s'agit seulement d'un niveau de liquide. Tandis qu'utiliser un microphone ou une photorésistance sera beaucoup plus facile. On distingue deux types de capteurs.

### Capteurs Transducteurs passifs

Ces capteurs ont pour objet de "transformer" ou plus exactement : *donner une image* de la grandeur physique qu'ils mesurent par une **résistance électrique variable** (en fait il s'agit d'une impédance, mais restons simples). Par exemple, le potentiomètre donne une résistance qui varie selon la position de son axe. Pour ce qui est de leur utilisation, il faudra nécessairement les utiliser avec un montage pour



Photorésistance

pouvoir les utiliser avec Arduino. Nous aurons l'occasion de voir tout cela en détail plus loin. Ainsi, il ne s'agit pas réellement de capteur, mais de **transducteurs** car nous sommes obligés de devoir utiliser un montage additionnel pour assurer une conversion de la grandeur mesurée en un signal électrique exploitable.

Ce sont principalement des transducteurs que nous allons mettre en œuvre dans le cours. Puisqu'ils ont l'avantage de pouvoir fonctionner seul et donc cela vous permettra de vous exercer au niveau électronique ! 🤖

## Capteurs actifs



Thermocouple

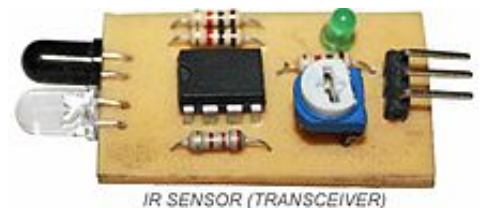
Cette autre catégorie de capteur est un peu spéciale et ne recense que très peu de capteurs en son sein. Il s'agit de capteur dont la grandeur physique elle-même mesurée va directement établir une relation électrique de sortie. C'est-à-dire qu'en sortie de ce type de capteur, il y aura une grandeur électrique, sans adjonction de tension à ses bornes. On peut dire que la présence de la tension (ou différence de potentiel, plus exactement) est générée par la grandeur physique. Nous n'entrerons pas dans le détail de ces capteurs et resterons dans ce qui est abordable à votre niveau.

## Les autres capteurs

En fait, il n'en existe pas réellement d'autres...

Ces "autres capteurs", dont je parle, sont les capteurs ou détecteurs tout prêts que l'on peut acheter dans le commerce, entre autres les détecteurs de mouvements ou capteur de distance. Ils ne font pas partie des deux catégories précédemment citées, puisqu'ils possèdent toute une électronique d'adaptation qui va s'occuper d'adapter la grandeur physique mesurée par le capteur et agir en fonction.

Par exemple allumer une ampoule lorsqu'il détecte un mouvement. Sachez cependant qu'il en existe beaucoup d'autres ! Ce sont donc bien des capteurs qui utilisent un ou des transducteurs. On pourra en fabriquer également, nous serons même obligés afin d'utiliser les transducteurs que je vais vous faire découvrir.



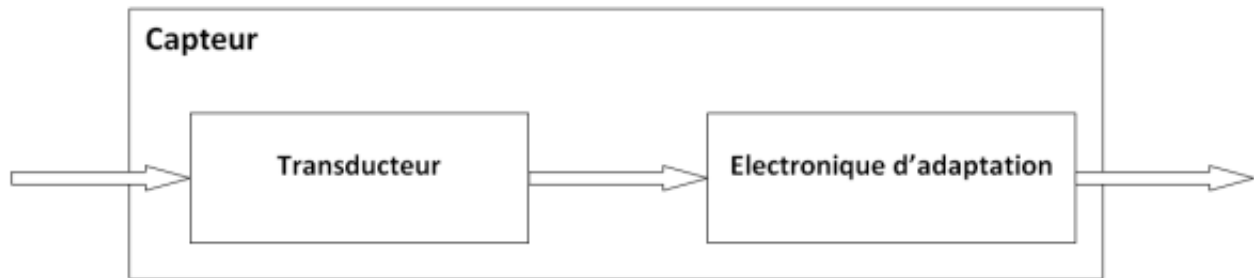
Détecteur infrarouge

Retenez donc bien la différence entre transducteur et capteur : un transducteur permet de donner une image de la grandeur physique mesurée par une autre grandeur physique, mais il doit être additionné à un montage pour être utilisé ; un capteur est nécessairement constitué d'un transducteur et d'un montage qui adapte la grandeur physique donnée par le transducteur en une information facilement manipulable.

## Un capteur, ça capte !

Un capteur, on l'a vu, est donc constitué d'un transducteur et d'une électronique d'adaptation. Le transducteur va d'abord mesurer la grandeur physique à mesurer, par exemple la luminosité. Il va donner une image de cette grandeur grâce à une autre

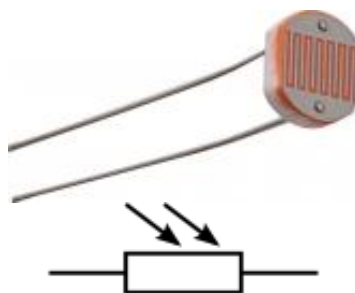
grandeur, dans ce cas une résistance électrique variable. Et l'électronique d'adaptation va se charger, par exemple, de "transformer" cette grandeur en une tension électrique image de la grandeur mesurée. Attention cependant, cela ne veut pas dire que la sortie sera **toujours** une tension variable. Ainsi, on pourrait par exemple plutôt avoir un courant variable (et tension fixe), ou carrément un message via une liaison de communication (voir série par exemple). Un capteur plus simple par exemple pourrait simplement nous délivrer un niveau logique pour donner une information telle que "obstacle présent/absent".

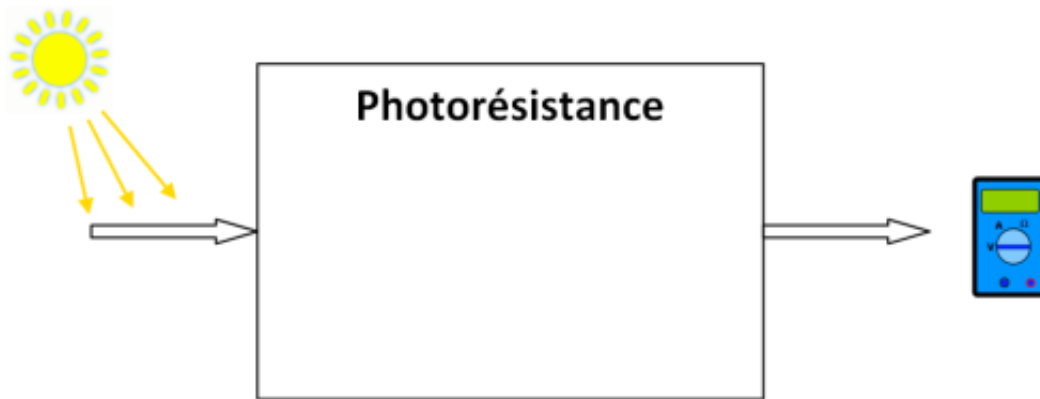


A gauche se trouve la grandeur physique mesurable. En sortie du transducteur c'est une autre grandeur physique, manipulable cette fois. Et en sortie de l'électronique d'adaptation, c'est l'information qui peut être sous forme de signal électrique ou d'une simple image de la grandeur physique mesurée par une autre grandeur physique telle qu'une tension électrique ou un courant.

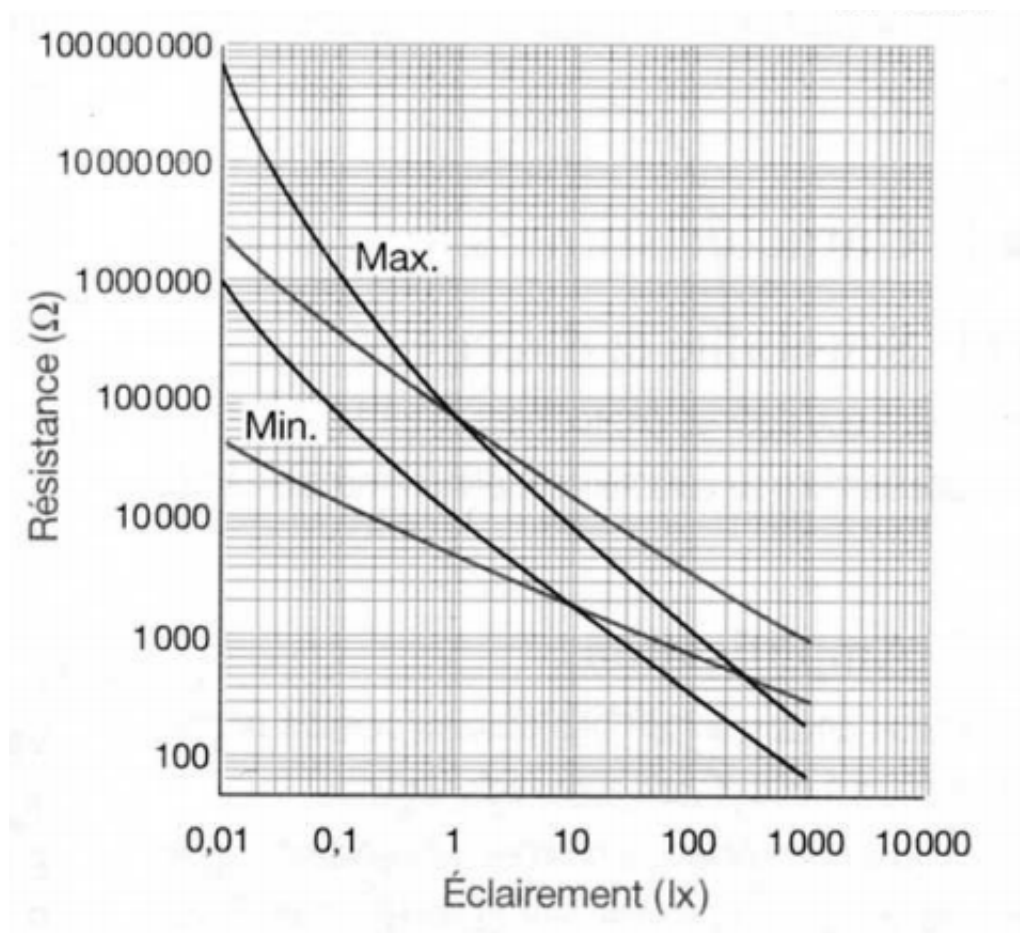
## Mesure, le rôle du transducteur

Gardons notre exemple avec un capteur, pardon, transducteur qui mesure la luminosité. Le transducteur qui opère avec cette grandeur est une *photorésistance* ou *LDR* (Light Depending Resistor). C'est une résistance photo-sensible, ou si vous préférez qui réagit à la lumière. La relation établie par la [photorésistance](#) entre la luminosité et sa résistance de sortie permet d'avoir une image de l'intensité lumineuse par une résistance électrique qui varie selon cette intensité. Voici son symbole électrique et une petite photo d'identité :





On a donc, en sortie du transducteur, une relation du type  $y$  en fonction de  $x$  :  $y = f(x)$  . Il s'agit simplement du **rapport** entre la grandeur physique d'entrée du capteur et sa grandeur physique de sortie. Ici, le rapport entre la luminosité et la résistance électrique de sortie. Dans les docs techniques, vous trouverez toujours ce rapport exprimé sous forme graphique (on appelle ça une **courbe caractéristique**). Ici, nous avons donc la résistance en fonction de la lumière :



## L'intérêt d'adapter

Adapter pour quoi faire ? Eh bien je pense déjà avoir répondu à cette question, mais reprenons les explications avec l'exemple ci-dessus. La photorésistance va fournir une résistance électrique qui fluctue selon la luminosité de façon quasi-proportionnelle (en fait ce n'est pas réellement le cas, mais faisons comme si 🙄). Eh bien, que va-t-on faire d'une telle grandeur ? Est-ce que nous pouvons l'utiliser avec notre carte Arduino ?

Directement ce n'est pas possible. Nous sommes obligé de l'adapter en une **tension qui varie** de façon proportionnelle à cette résistance, puisque nous ne sommes pas capable de mesurer une résistance directement. Ensuite nous pourrons simplement utiliser la fonction `analogRead()` pour lire la valeur mesurée. Néanmoins, il faudra certainement faire des calculs dans le programme pour donner une réelle image de la luminosité. Et ensuite, éventuellement, afficher cette grandeur ou la transmettre par la liaison série (ou l'utiliser de la manière qui vous fait plaisir 😊 !). De nouveau, voici la relation établissant le rapport entre les deux grandeurs physiques d'entrée et de sortie d'un transducteur :

$$y = f(x)$$

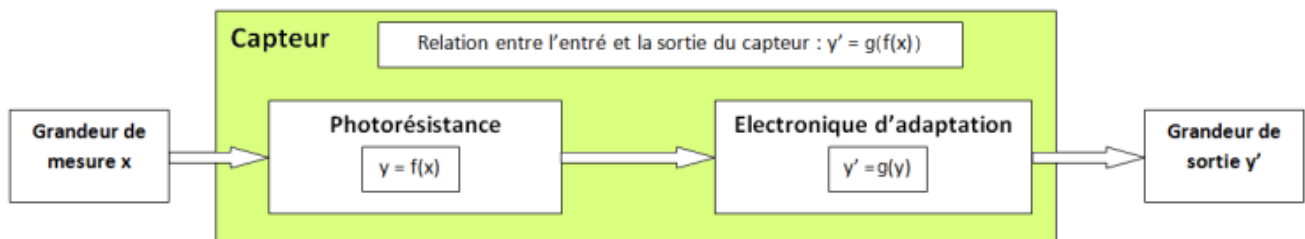
A partir de cette relation, on va pouvoir gérer l'électronique d'adaptation pour faire en sorte d'établir une nouvelle relation qui soit également une image de la mesure réalisée par le capteur. C'est à dire que l'on va créer une image proportionnelle de la grandeur physique délivrée en sortie du capteur par une nouvelle grandeur physique qui sera, cette fois-ci, bien mieux exploitable. En l'occurrence une tension dans notre cas. La nouvelle relation sera du style  $y'$  (attention, il ne s'agit pas de la dérivée de  $y$  mais de  $y$  *image*) en fonction de  $y$  :

$$y' = g(y)$$

Ce qui revient à dire que  $y'$  est la relation de sortie du capteur en fonction de la grandeur de mesure d'entrée. Soit :

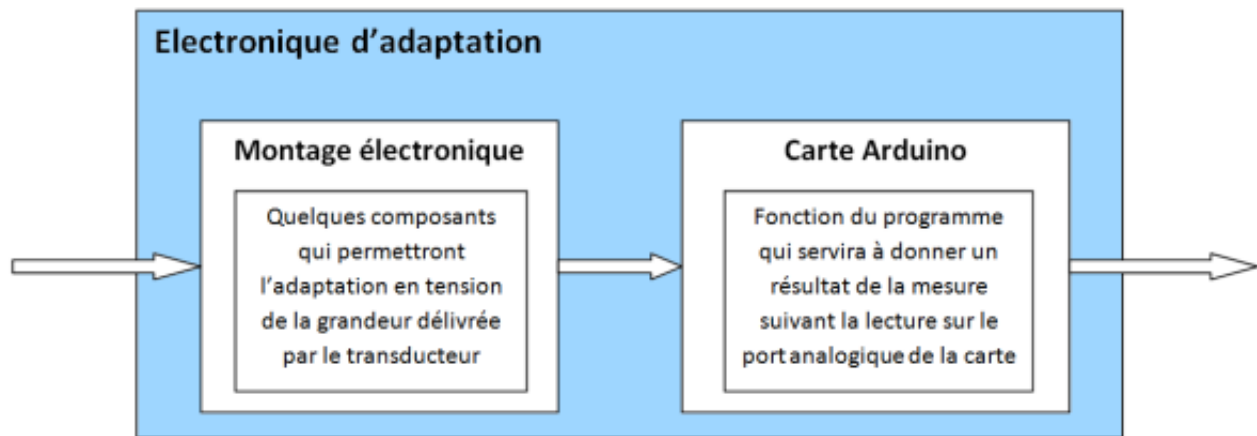
$$y' = g(y) \quad y' = g(f(x))$$

Concrètement, nous retrouvons ces formules dans chaque partie du capteur :



## L'électronique d'adaptation

Elle sera propre à chaque capteur. Cependant, je l'ai énoncé au début de ce chapitre, nous utiliserons principalement des transducteurs qui ont en sortie une résistance électrique variable. L'électronique d'adaptation sera donc quasiment la même pour tous. La seule chose qui changera certainement, c'est le programme. Oui car la carte Arduino fait partie intégrante du capteur puisque c'est avec elle que nous allons "fabriquer" nos capteurs. Le programme sera donc différent pour chaque capteur, d'autant plus qu'ils n'ont pas tous les mêmes relations de sortie... vous l'aurez compris, on aura de quoi s'amuser ! 😊 Pour conclure sur l'intérieur du capteur, rentrons dans la partie électronique d'adaptation. La carte Arduino faisant partie de cette électronique, on va avoir un schéma tel que celui-ci :



A l'entrée de l'électronique d'adaptation se trouve la grandeur de sortie du transducteur ; à la sortie de l'électronique d'adaptation se trouve la grandeur de sortie du capteur. On peut après faire ce que l'on veut de la mesure prise par le capteur. Toujours avec la carte Arduino, dans une autre fonction du programme, on pourra alors transformer la valeur mesurée pour la transmettre via la liaison série ou simplement l'afficher sur un écran LCD, voir l'utiliser dans une fonction qui détermine si la mesure dépasse un seuil limite afin de fermer les volets quand il fait nuit...

---

## Les caractéristiques d'un capteur

Pour terminer cette introduction générale sur les capteurs, nous allons aborder les caractéristiques essentielles à connaître.

### Les critères à ne pas négliger

#### *La plage de mesure*

La **plage de mesure**, ou *gamme de mesure*, est la première chose à regarder dans le choix d'un capteur ou d'un transducteur. C'est elle qui définit si vous allez pouvoir mesurer la grandeur physique sur une grande plage ou non. Par exemple pouvoir mesurer une température de  $-50^{\circ}\text{C}$  à  $+200^{\circ}\text{C}$ . Tout dépendra de ce que vous voudrez mesurer.

#### *La précision*

La précision est le deuxième critère de choix le plus important. En effet, si votre capteur de température a une précision de  $1^{\circ}\text{C}$ , vous aurez du mal à l'utiliser dans un projet qui demande une précision de mesure de températures de  $0.1^{\circ}\text{C}$  ! En règle générale, la précision est plus grande lorsque la plage de mesure est faible et inversement elle devient moins grande lorsque la plage de mesure augmente. Il est en effet assez difficile de fabriquer des capteurs qui ont une plage de mesure très grande par exemple un voltmètre qui mesurerait jusqu'à  $1000\text{V}$  avec une précision de  $0.001\text{V}$  ! Et puis, c'est rarement utile d'avoir ces deux paramètres à leur valeur la plus élevée (grande plage de mesure et grande précision). Dans un cas le plus général, à prix égal un capteur qui mesure une plus grande plage aura sûrement une précision plus faible qu'un capteur mesurant une plage plus réduite.



## ***Sa tension d'alimentation***

Il est en effet important de savoir à quelle tension il fonctionne, pour ne pas avoir de mauvaises surprises lorsque l'on veut l'utiliser !

## **D'autres caractéristiques à connaître**

### ***La résolution***

Certains capteurs proposent une sortie via une communication (série, I<sup>2</sup>C, SPI...). Du coup, la sortie est dite "numérique" (puisque l'on récupère une information logique plutôt qu'analogique). Un facteur à prendre en compte est la résolution proposée. Certains capteurs seront donc sur 8 bits (la valeur de sortie sera codée sur 256 niveaux), d'autres 10 bits, 16 bits, 32 bits... Il est évident que plus la résolution est élevée et plus la précision offerte est grande.

### ***La reproductibilité***

Ce facteur sert à déterminer la fiabilité d'une mesure. Si par exemple vous souhaitez mesurer une température à 0.1°C près, et que le capteur que vous utilisez oscille entre 10.3° et 10.8°C lorsque vous faites une série de mesures consécutives dans un intervalle de temps court, vous n'êtes pas précis. La reproductibilité est donc le critère servant à exprimer la fiabilité d'une mesure au travers de répétitions consécutives, et le cas échéant exprime l'écart-type et la dispersion de ces dernières. Si la dispersion est élevée, il peut-être utile de faire plusieurs mesures en un court-temps pour ensuite faire une moyenne de ces dernières.

### ***Le temps de réponse***

Comme son nom l'indique, cela détermine la vitesse à laquelle le capteur réagit par rapport au changement de l'environnement. Par exemple, les changements de température sont des phénomènes souvent lents à mesurer. Si vous passez le capteur d'un milieu très chaud à un milieu très froid, le capteur va mettre du temps (quelques secondes) pour proposer une information fiable. A contrario, certains capteurs réagissent très vite et ont donc un temps de réponse très faible.

### ***La bande passante***

Cette caractéristique est plus difficile à comprendre et est liée au temps de réponse. Elle correspond à la capacité du capteur à répondre aux sollicitations de son environnement. Si sa bande passante est élevée, il peut mesurer aussi bien des phénomènes lents que des phénomènes rapides. Si au contraire elle est faible, sa capacité à mesurer des phénomènes lents **ou** rapides sera réduite sur une certaine plage de fréquences.

### ***La gamme de température d'utilisation***

Ce titre est assez explicite. En effet, lorsque l'on mesure certains phénomènes physiques, le capteur doit avoir une certaine réponse. Cependant, il arrive que le phénomène soit conditionné par la température. Le capteur doit donc être utilisé dans certaines conditions pour avoir une réponse correcte (et ne pas être détérioré).

Lorsque vous utilisez un capteur pour la première fois, il est souvent utile de pratiquer un **étalonnage**. Cette opération consiste à prendre quelques mesures pour vérifier/corriger la justesse de sa caractéristique par rapport à la datasheet ou aux conditions ambiantes.

Nous verrons tout au long des chapitres certaines caractéristiques. Après ce sera à vous de choisir vos capteurs en fonction des caractéristiques dont vous aurez besoin.

## [Arduino 502] Différents types de mesures

Pour commencer ce chapitre sur les capteurs, j'ai rassemblé un petit nombre de capteurs qui sont très utilisés en robotique et en domotique. Vous pourrez ainsi comprendre certains concepts généraux tout en ayant accès à une base de connaissance qui puisse vous aider lorsque vous aurez besoin de mettre en place vos projets personnels. On va commencer en douceur avec les capteurs logiques, aussi appelés **Tout Ou Rien** (TOR). Puis on continuera vers les capteurs utilisant des transducteurs à résistance de sortie variable. On verra ensuite des capteurs un peu particuliers puis pour finir des capteurs un peu compliqués. C'est parti !

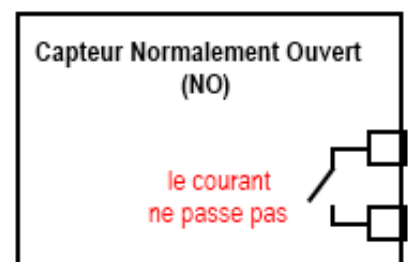
### Tout Ou Rien, un capteur qui sait ce qu'il veut

On va donc commencer par voir quelques capteurs dont la sortie est un état logique qui indique un état : **Ouvert** ou **Fermé**. Ces capteurs sont appelés des capteur Tout Ou Rien, capteurs **logiques** ou encore capteurs à **rupture de tension**.

#### Deux états

Lorsque le capteur est dit ouvert (ou *open* en anglais), le courant ne passe pas entre ses bornes. S'il s'agit de la **position de repos**, c'est à dire lorsque le capteur ne capte pas la grandeur qu'il doit capter, on dit alors que le capteur a un contact de type **Normalement Ouvert** (ou *Normally Open* en anglais). Tandis que si le capteur est dit fermé (ou *close* en anglais), le courant peut passer entre ses bornes. S'il s'agit cette fois de sa position de repos, alors on dit qu'il possède un contact **Normalement Fermé** (ou *Normally Closed*).

J'ai schématisé les contacts par des interrupteurs reliés à deux bornes (les carrés à droite) du capteur. C'est le principe d'un capteur TOR (Tout Ou Rien), mais ce n'est pas forcément des interrupteurs qu'il y a dedans, on va le voir bientôt.





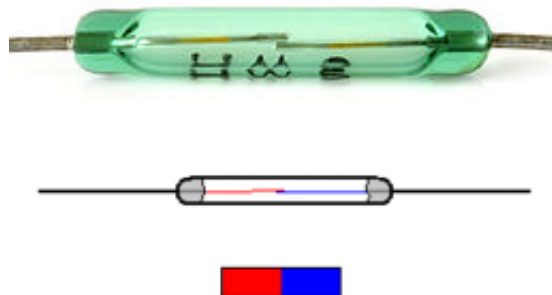
# Le capteur ILS ou Interrupteur à Lame Souple

## Le principe du TOR

Une question qui vous est peut-être passée par la tête : mais comment ce capteur peut mesurer une grandeur physique avec seulement deux états en sortie ? C'est assez facile à comprendre. Imaginons, dans le cadre d'un de vos projets personnels, que vous ayez l'intention de faire descendre les volets électriques lorsqu'il fait nuit. Vous allez alors avoir recours à un capteur de luminosité qui vous donnera une image, par une autre grandeur physique, de l'intensité lumineuse mesurée. Hors, vous ne voulez pas que ce capteur vous dise s'il fait un peu jour ou un peu nuit, ou entre les deux... non, il vous faut une réponse qui soit OUI ou NON il fait nuit. Vous aurez donc besoin d'un capteur TOR. Alors, il en existe qui sont capables de donner une réponse TOR, mais lorsque l'on utilisera un transducteur, on devra gérer ça nous même avec Arduino et un peu d'électronique.

## Champ magnétique

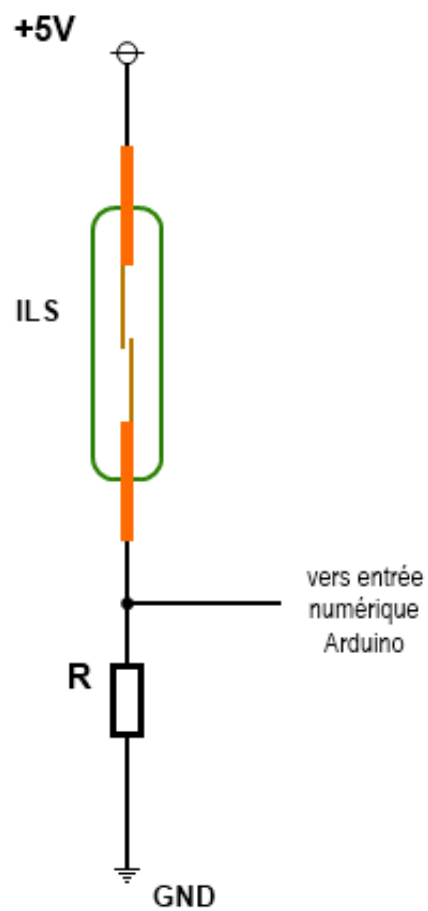
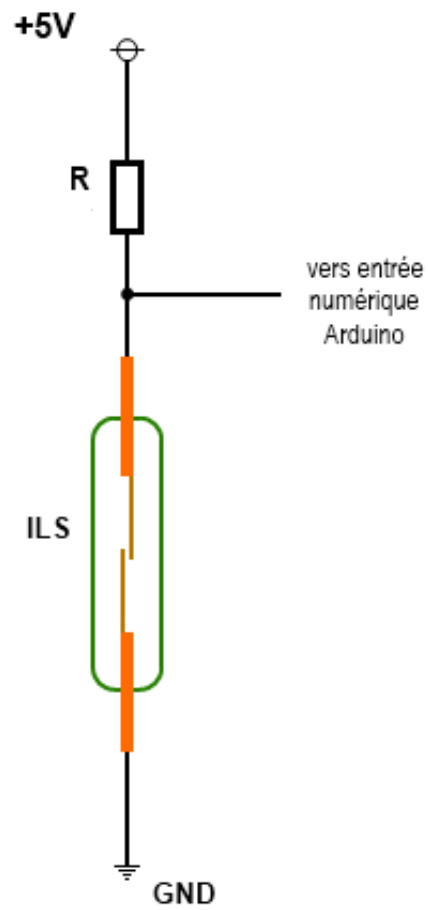
Ne prenez pas peur, je vais simplement vous présenter le capteur ILS qui utilise le champ magnétique pour fonctionner. 😊 En effet, ce capteur, est un capteur TOR qui détecte la présence de champ magnétique. Il est composé de deux lames métalliques souples et sensibles au champ magnétique. Lorsqu'un champ magnétique est proche du capteur, par exemple un aimant, eh bien les deux lames se mettent en contact et laissent alors passer le courant électrique. D'une façon beaucoup plus simple, c'est relativement semblable à un interrupteur mais qui est actionné par un champ magnétique. Photo d'un interrupteur ILS et image, extraites du site [Wikipédia](https://fr.wikipedia.org/wiki/Interrupteur_%C3%A0_lame_souple) :



Dès que l'on approche un aimant, à partir d'un certain seuil de champ magnétique, le capteur agit. Il devient alors un contact fermé et reprend sa position de repos, contact NO, dès que l'on retire le champ magnétique. Ce type de capteur est très utilisé en sécurité dans les alarmes de maison. On les trouve essentiellement au niveau des portes et fenêtres pour détecter leur ouverture.

## Câblage

Le câblage de ce capteur peut être procédé de différentes manières. On peut en effet l'utiliser de façon à ce que le courant ne passe pas lorsque rien ne se passe, ou bien qu'il ne passe pas lorsqu'il est actionné.



1. Dans le premier cas (image de gauche), la sortie vaut HIGH quand l'ILS est au

repos et LOW lorsqu'il est actionné par un champ magnétique.

2. Sur la deuxième image, le câblage est différent et fait en sorte que la sortie soit à LOW lorsque le contact ILS est au repos et passe à HIGH dès qu'il est actionné par un champ magnétique.

Un petit programme tout simple qui permet de voir si l'ILS est actionné ou non, selon le schéma utilisé :

```
1  const char entree_ils = 2; //utilisation de la broche numérique numéro 2 comme entrée
2
3  const char led_indication = 13; //utilisation de la LED de la carte pour indiquer si
4
5  unsigned char configuration_ils = 0; // ou 1, dépend du câblage de l'ILS selon les s
6  /*
7  0 pour le premier schéma (gauche)
8  1 pour le deuxième schéma (droite)
9  */
10
11 void setup()
12 {
13     //définition des broches utilisées
14     pinMode(entree_ils, INPUT);
15     pinMode(led_indication, OUTPUT);
16 }
17
18 void loop()
19 {
20     if(configuration_ils) //si c'est le deuxième schéma
21     {
22         digitalWrite(led_indication, digitalRead(entree_ils)); //la LED est éteinte
23     }
24     else //si c'est le premier schéma
25     {
26         digitalWrite(led_indication, !digitalRead(entree_ils)); //la LED est allumée
27     }
28 }
```

## Capteur logique prêt à l'emploi

Bon, là ça va être très très court, puisque vous savez déjà faire ! Les capteurs tout prêts que l'on peut acheter dans le commerce et qui fournissent un état de sortie logique (LOW ou HIGH) sont utilisables tels quels. Il suffit de connecter la sortie du capteur sur une entrée numérique de la carte Arduino et de lire dans le programme l'état sur cette broche. Vous saurez donc en un rien de temps ce que le capteur indique. On peut citer pour exemple les capteurs de mouvements.

Le programme précédent, utilisé avec l'ILS, est aussi utilisable avec un capteur logique quelconque. Après, à vous de voir ce que vous voudrez faire avec vos capteurs.

---

## Capteurs à résistance de sortie variable

### La photo-résistance

Nous y voilà, on va enfin voir le transducteur dont j'arrête pas de vous parler depuis tout à l'heure : la photo-résistance ! Je vois que vous commencez à être impatients. 🤖

### *Petit aperçu*

La photo-résistance est un composant électronique qui est de type transducteur. Il est donc capable de donner une image de la grandeur physique mesurée, la lumière ou précisément la luminosité, grâce à une autre grandeur physique, la résistance.



*Photo d'une photo-résistance – Wikipédia*

On trouve généralement ce composant en utilisation domotique, pour... devinez quoi ?! ... faire monter ou descendre les volets électriques d'une habitation ! 😊 Mais on peut également le retrouver en robotique, par exemple pour créer un robot suiveur de ligne noire. Enfin on le trouve aussi dans beaucoup d'autres applications, vous saurez trouver vous-mêmes où est-ce que vous l'utiliserez, je vous fais confiance de ce point de vue là. 😊

### *Propriété*

La photo-résistance suit une relation toute simple entre sa résistance et la luminosité :

$$R = f(E)$$

Avec :

- R la résistance en Ohm ( $\Omega$ )
- E l'intensité lumineuse en [lux](#) (lx)

Plus l'intensité lumineuse est élevée, plus la résistance diminue. À l'inverse, plus il fait sombre, plus la résistance augmente. Malheureusement, les photo-résistances ne sont pas des transducteurs très précis. Ils ont notamment des problèmes de linéarité, un temps de réponse qui peut être élevé et une grande tolérance au niveau de la résistance. Nous les utiliserons donc pour des applications qui ne demandent que peu de rigueur. Ce qui ira bien pour ce qu'on veut en faire.



*Symbole de la photo-résistance*

Une photo-résistance est une résistance qui possède une valeur de base en Ohm. C'est à dire qu'elle est calibrée pour avoir une valeur, par exemple 47 kOhm, à un certain seuil de luminosité. À ce seuil on peut donc mesurer cette valeur, suivant la

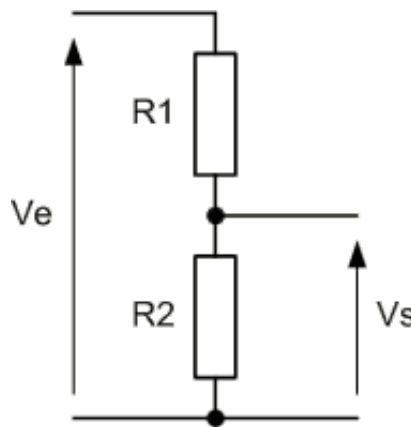
tolérance qu'affiche la photo-résistance. Si la luminosité augmente, la résistance de base n'est plus vraie et chute. En revanche, dans le noir, la résistance augmente bien au delà de la résistance de base.

Génial !! J'en veux, j'en veux ! Comment on l'utilise ? 😊

La photorésistance est principalement utilisée dans un montage en pont diviseur de tension. Vous le connaissez ce montage, c'est exactement le même principe de fonctionnement que le potentiomètre. Sauf que ce ne sera pas vous qui allez modifier le curseur mais la photorésistance qui, selon la luminosité, va donner une valeur ohmique différente. Ce qui aura pour effet d'avoir une influence sur la tension en sortie du pont diviseur.

### **Rappel sur le pont diviseur de tension**

Je vous rappelle le montage d'un pont diviseur de tension :

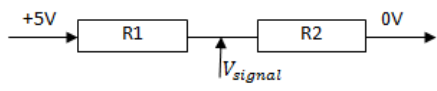


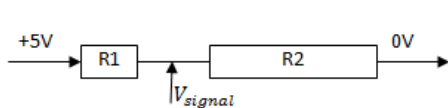
La formule associée est la suivante :

$$V_s = V_e \frac{R_2}{R_1 + R_2}$$

Cette formule s'applique **uniquement** dans le cas où la sortie  $V_s$  ne délivre pas de courant (cas des entrées numériques ou analogiques d'un microcontrôleur par exemple). Dans le cas où il y a justement un courant qui sort de ce pont, cela modifie la valeur de la tension de sortie. C'est comme si vous rajoutiez une résistance en parallèle de la sortie. Cela a donc pour effet de donner une résistance  $R_2$  équivalente plus faible et donc de changer la tension (en appliquant la formule).

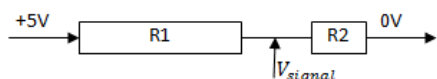
Reprenons le tableau présentant quelques exemples :

Schéma équivalent	Position du curseur	Tension sur la broche C
	Curseur à la moitié	$V_{signal} = (1 - \frac{50}{100}) \times 5 = 2.5V$
	Curseur à 25%	



du départ

$$V_{signal} = \left(1 - \frac{25}{100}\right) \times 5 = 3.75V$$



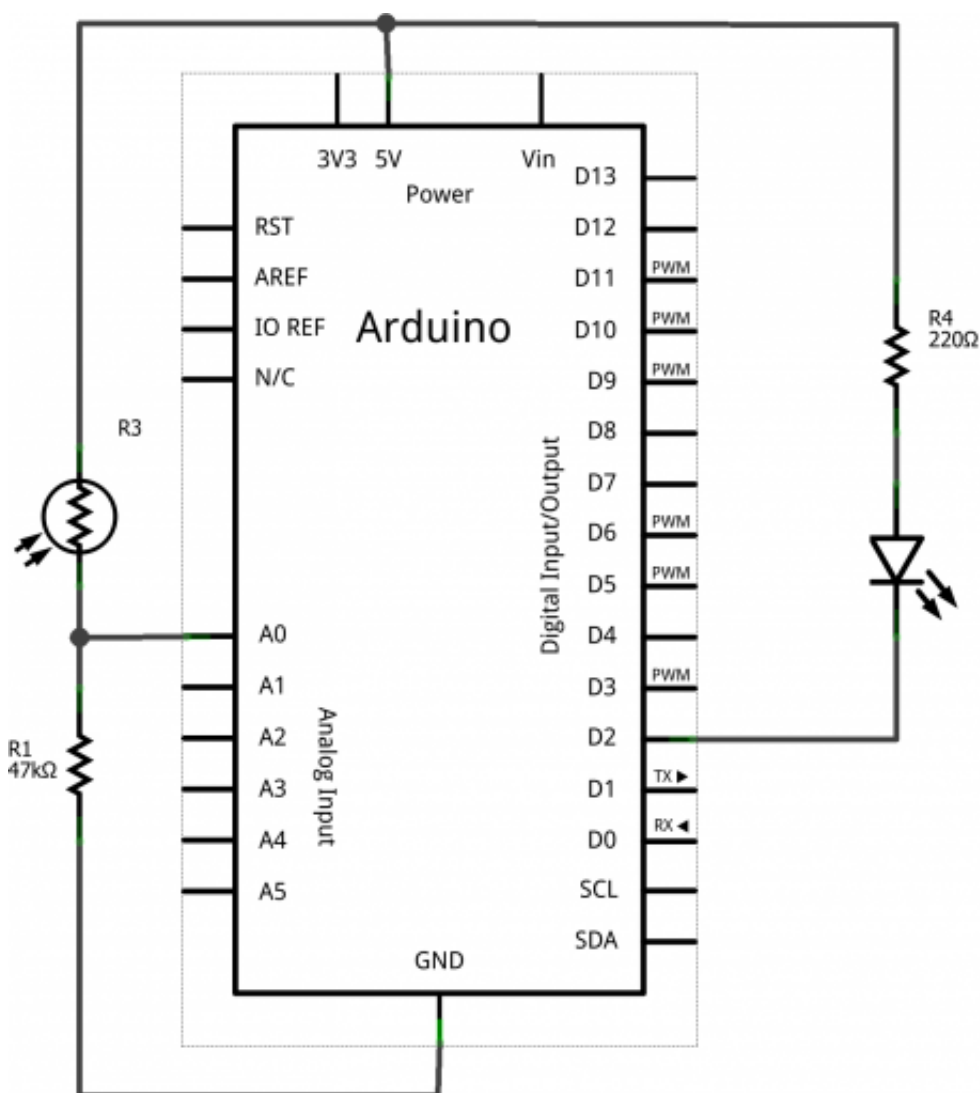
Curseur à **75%**  
du départ

$$V_{signal} = \left(1 - \frac{75}{100}\right) \times 5 = 1.25V$$

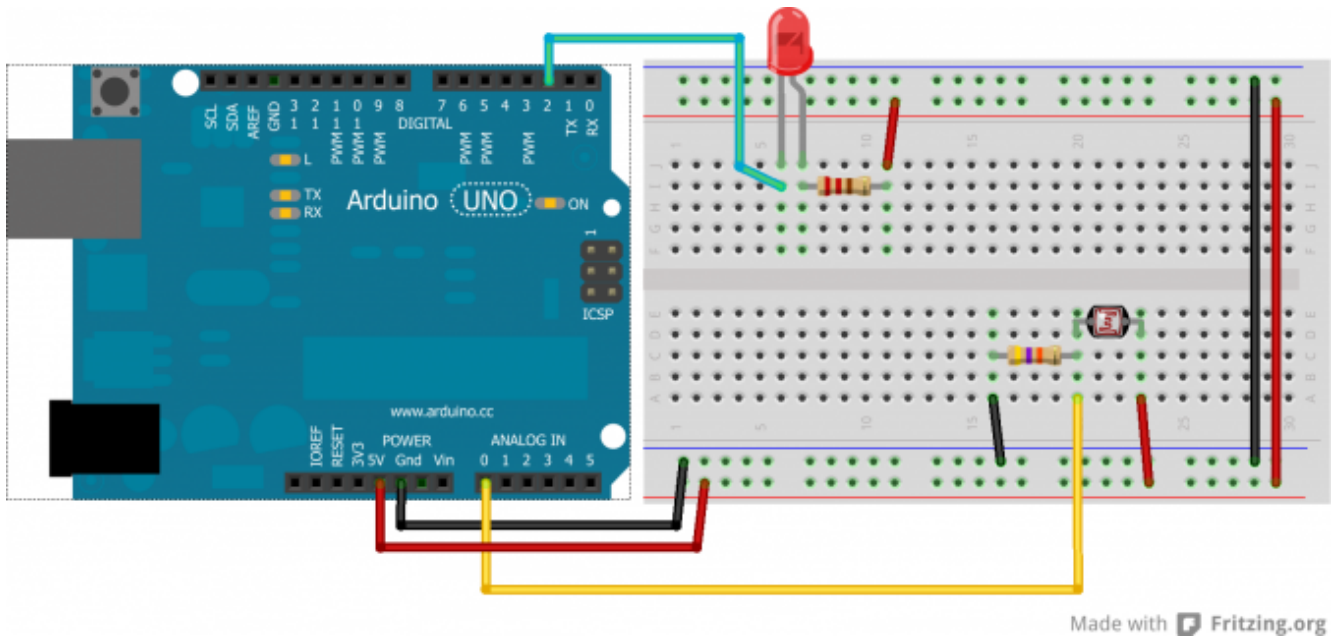
Vous allez donc maintenant comprendre pourquoi je vais vous donner deux montages pour une utilisation différente de la photorésistance...

## Utilisation n°1

Ce premier montage, va être le premier capteur que vous allez créer ! Facile, puisque je vous fais tout le travail. 😊 Le principe de ce montage réside sur l'utilisation que l'on va faire de la photo-résistance. Comme je vous le disais à l'instant, on va l'utiliser dans un pont diviseur de tension. Exactement comme lorsque l'on utilise un potentiomètre. Sauf que dans ce cas, c'est l'intensité lumineuse qui va faire varier la tension en sortie. Voyez plutôt :



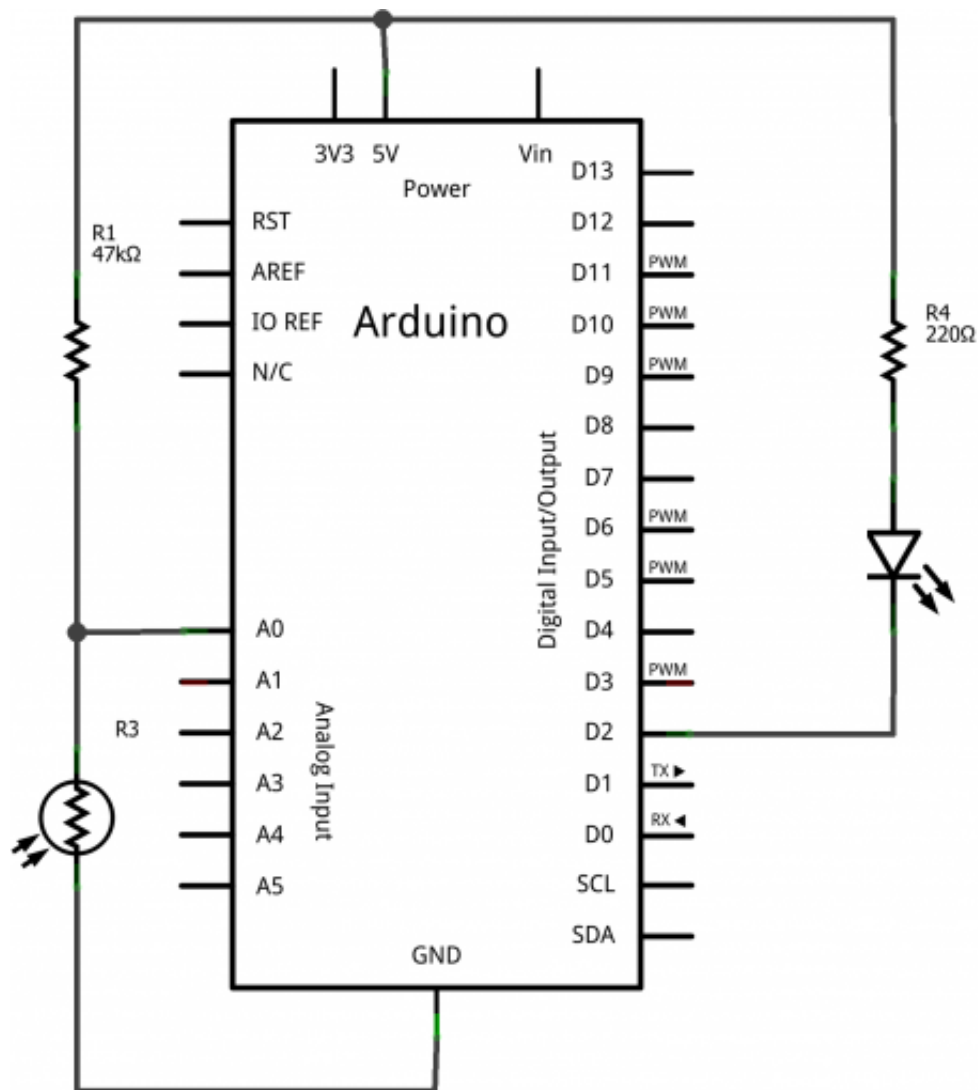




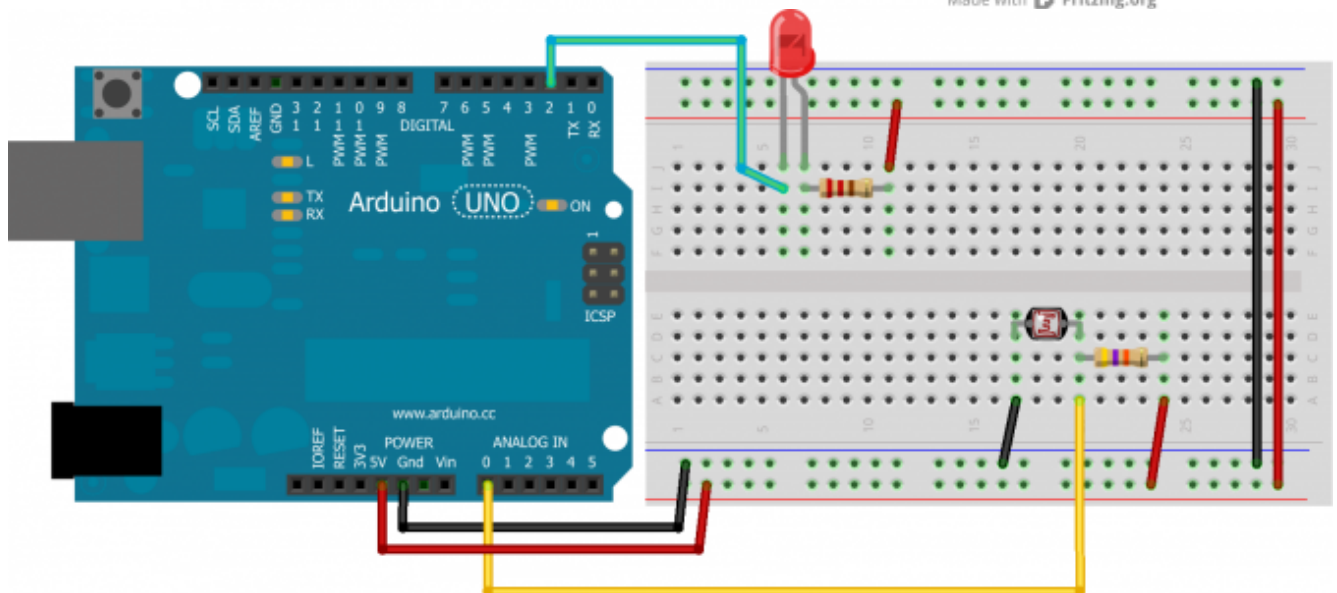
On calibre le pont diviseur de tension de manière à ce qu'il soit "équitable" et divise la tension d'alimentation par 2 en sa sortie. Ainsi, lorsque la luminosité fluctuera, on pourra mesurer ces variations avec la carte Arduino. Avec ce montage, **plus l'intensité lumineuse est élevée, plus la tension en sortie du pont sera élevée** à son tour. Et inversement, plus il fait sombre, moins la tension est élevée.

### *Utilisation n°2*

Tandis que là, c'est l'effet inverse qui va se produire : **plus il y aura de lumière, moins il y aura de tension en sortie du pont**. Et plus il fera sombre, plus la tension sera élevée.



Made with Fritzing.org



Made with Fritzing.org

Rien de bien sorcier. Il suffit de bien comprendre l'intérêt du pont diviseur de tension.

Un peu de programmation

Et si vous aviez un réveil, qui ne vous donne pas l'heure ? Fort utile, n'est-ce pas ! 😊 Non, sérieusement, il va vous réveiller dès que le jour se lève... ce qui fait que vous dormirez plus longtemps en hiver. C'est vos profs qui vont pas être contents ! 🤖 Vous n'aurez qu'à dire que c'est de ma faute. 😊 Bon allez, un peu de tenue quand même, je ne voudrais pas être la cause de votre échec scolaire. Cette fois, vraiment sérieusement, nous allons faire un tout petit programme qui va simplement détecter la présence ou l'absence de lumière. Lorsque la tension en sortie du pont diviseur de tension créée avec la photorésistance et la résistance fixe chute, c'est que la luminosité augmente. À vous de choisir le schéma correspondant suivant les deux présentés précédemment. Pour commencer, on va initialiser les variables et tout le tralala qui va avec.

```
1  const char led = 2;           //Une LED pour indiquer s'il fait jour
2  const char capteur = 0; //broche A0 sur laquelle va être connecté le pont diviseur de
3
4  float tension = 0;           //variable qui va enregistrer la tension lue en sortie
5  float seuilObscurite = 1.5;  // valeur en V, seuil qui détermine le niveau auquel
6
7  void setup()
8  {
9      //définition des broches utilisées
10     pinMode(led, OUTPUT);
11
12     Serial.begin(9600); //la voie série pour monitorer
13 }
```

Qu'allons-nous retrouver dans la fonction loop() ? Eh bien avant tout il va falloir lire la valeur présente en entrée de la broche analogique A0. Puis, on va calculer la tension correspondante à la valeur lue. Enfin, on va la comparer au seuil préalablement défini qui indique le niveau pour lequel l'absence de lumière fait loi.

```
1  void loop()
2  {
3      tension = (analogRead(capteur) * 5.0) / 1024; // conversion de cette valeur en
4
5      if(tension >= seuilObscurite)
6      {
7          digitalWrite(led, LOW); //On allume la LED
8      }
9      else
10     {
11         digitalWrite(led, HIGH); //On éteint la LED
12     }
13     // envoi vers l'ordinateur, via la liaison série, la valeur de la tension lue
14     Serial.print("Tension = ");
15     Serial.print(tension);
16     Serial.println(" V");
17
18     delay(500); // délai pour ne prendre des mesures que toutes les demi-secondes
19 }
```

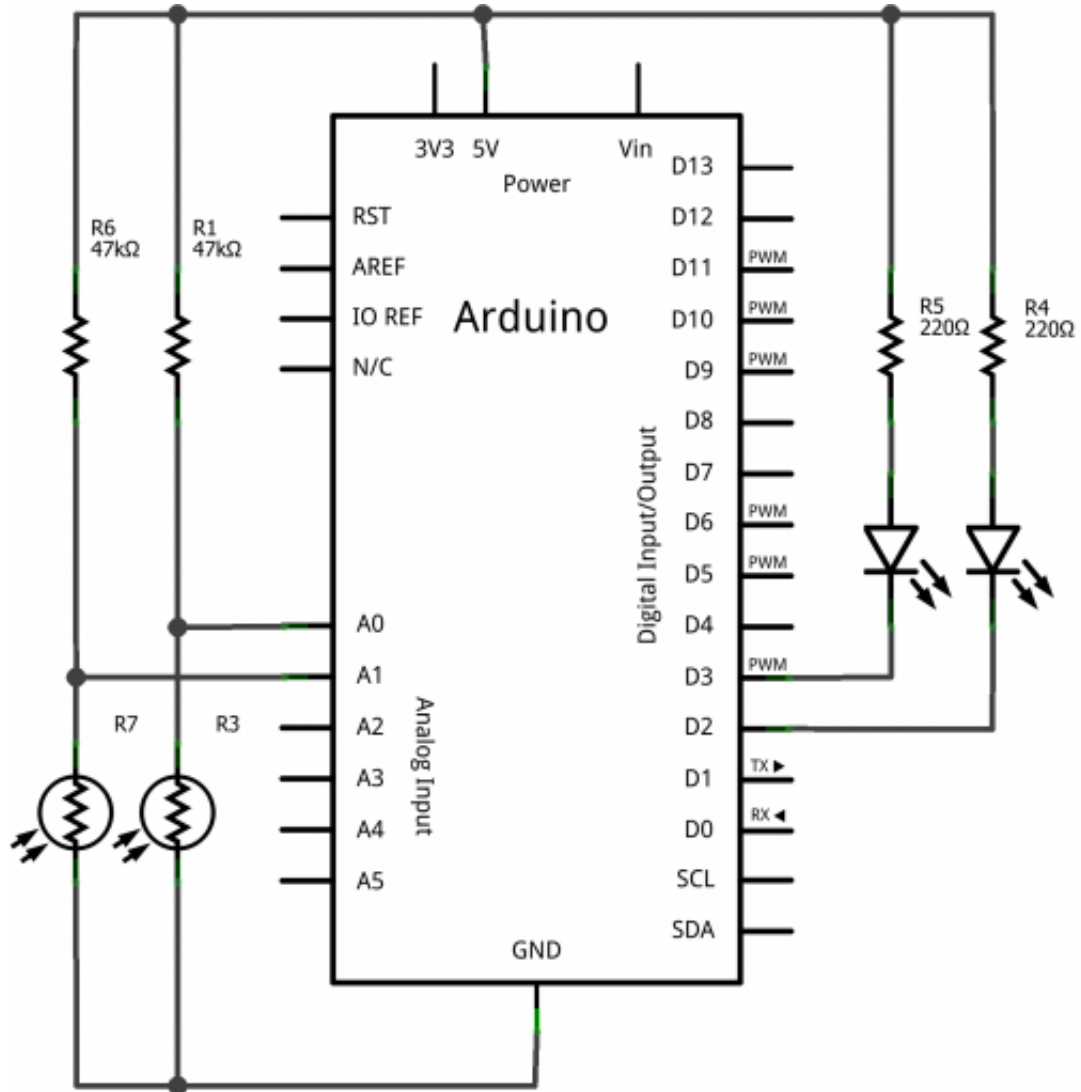
## Un programme plus évolué

Après tant de difficulté (🤖), voici un nouveau programme qui vous sera peut-être plus intéressant à faire. En fait ça le deviendra dès que je vous aurais dit l'application qui en

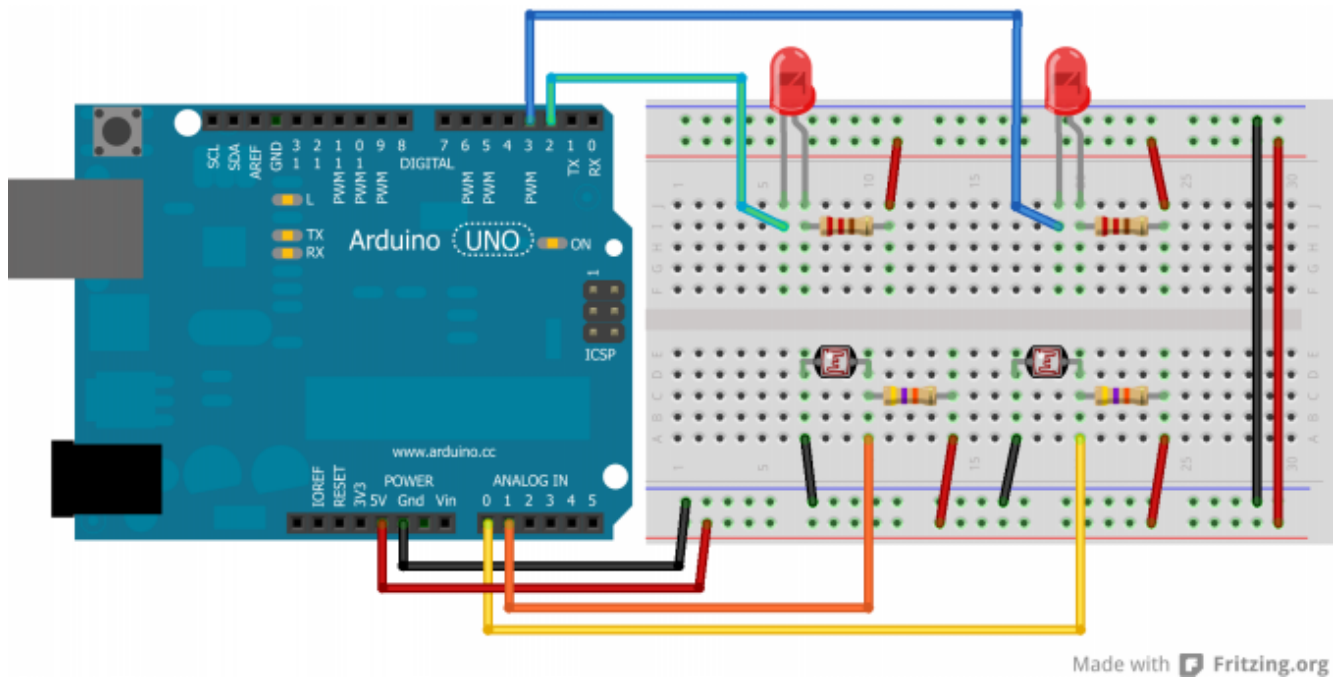
est prévue...

## Préparation

Cette fois, je vais vous demander d'avoir deux photorésistances identiques. Le principe est simple on va faire une comparaison entre les deux valeurs retournées par les deux capteurs (deux fois le montage précédent).



Made with  Fritzing.org



Si la valeur à droite est plus forte, on allumera une LED en broche 2. Sinon, on allume en broche 3. Si la différence est faible, on allume les deux. Dans tous les cas, il n'y a pas de cas intermédiaire. C'est soit à gauche, soit à droite (selon la disposition des photorésistances). Ce principe pourrait être appliqué à un petit robot mobile avec un comportement de papillon de nuit. Il cherche la source de lumière la plus intense à proximité.

## Le programme

Il parle de lui même, pas besoin d'en dire plus.

```

1 //déclaration des broches utilisées
2 const char ledDroite = 2;
3 const char ledGauche = 3;
4 const char capteurDroit = 0;
5 const char capteurGauche = 1;
6
7 /* deux variables par capteur qui une stockera la valeur lue sur la broche analogique
8 et l'autre stockera le résultat de la conversion de la précédente valeur en tension
9 float lectureDroite = 0;
10 float lectureGauche = 0;
11 float tensionDroite = 0;
12 float tensionGauche = 0;
13
14 void setup()
15 {
16     pinMode(ledDroite, OUTPUT);
17     pinMode(ledGauche, OUTPUT);
18     Serial.begin(9600);
19 }
20
21 void loop()
22 {
23     lectureDroite = analogRead(capteurDroit); // lecture de la valeur en sortie du
24     lectureGauche = analogRead(capteurGauche);
25     tensionDroite = (lectureDroite * 5.0) / 1024; // conversion en tension de la va
26     tensionGauche = (lectureGauche * 5.0) / 1024;

```

```

27
28     if(tensionDroite > tensionGauche) // si la tension lue en sortie du capteur 1 es
29     {
30         digitalWrite(ledDroite, LOW); //allumée
31         digitalWrite(ledGauche, HIGH); //éteinte
32     }
33     else
34     {
35         digitalWrite(ledDroite, HIGH); //éteinte
36         digitalWrite(ledGauche, LOW); //allumée
37     }
38     //envoi des données lues vers l'ordinateur
39     Serial.print("Tension Droite = ");
40     Serial.print(tensionDroite);
41     Serial.println(" V");
42     Serial.print("Tension Gauche = ");
43     Serial.print(tensionGauche);
44     Serial.println(" V");
45
46     delay(100); // délai pour ne prendre une mesure que toutes les 100ms
47 }

```

J'en parlais donc brièvement, ce petit programme peut servir de cerveau à un petit robot mobile qui cherchera alors la source de lumière la plus intense à ses “yeux”. Vous n’aurez plus qu’à remplacer les LED par une commande de moteur (que l’on verra dans la prochaine partie sur les moteurs) et alimenter le tout sur batterie pour voir votre robot circuler entre vos pattes. 🤖 Bien entendu ce programme pourrait largement être amélioré !

## Un autre petit robot mobile

On peut renverser la situation pour faire en sorte que le robot suive une ligne noire tracée au sol. C’est un robot suiveur de ligne. Le principe est de “coller” les deux “yeux” du robot au sol. L’état initial va être d’avoir un œil de chaque côté de la ligne noire. Le robot avance tant qu’il voit du blanc (car la ligne noire est sur une surface claire, blanche en général). Dès qu’il va voir du noir (lorsque la luminosité aura diminué), il va alors arrêter de faire tourner le moteur opposé à l’œil qui a vu la ligne noire. Ainsi, le robot va modifier sa trajectoire et va continuer en suivant la ligne. À partir de cela, je vous laisse réfléchir à tout ce que vous pouvez faire. Non pas de programme donné tout frais, je viens de définir un cahier des charges, somme toute, assez simple. Vous n’avez donc plus qu’à le suivre pour arriver à vos fins.

---

## Capteurs à tension de sortie variable

Passons à un capteur un petit peu plus drôle et plus étonnant, dont les applications sont très variées !

### L’élément piézoélectrique

Sous ce nom peu commun se cache un phénomène physique très intéressant. L’élément piézoélectrique, que l’on retrouve dans divers objets du quotidien (montres, certains briquets, raquettes de tennis, ...) présente en effet toute une panoplie de



caractéristiques utilisées dans des dizaines voire centaines de domaines. Nous allons voir tout ça en détail. Nous, ce qui va nous intéresser pour le moment, c'est sa propriété à capter des sons.



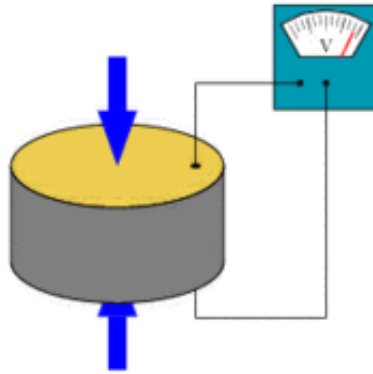
Éléments piézoélectriques de montre à gauche et allumeur de briquet à droite – source Wikipédia

## Constitution

Avant de parler de son fonctionnement, voyons un peu sa constitution. Prenons les éléments piézoélectriques de la première image, à gauche. On observe qu'ils se trouvent sous une forme de pastille composée de plusieurs couches. Généralement c'est une pastille de céramique qui est montée sur une pastille métallique. La fabrication de ces éléments étant très complexe, nous en resterons à ce niveau d'approche.

## Propriété

J'ai trouvé amusant de voir sur internet que l'on parlait de sa propriété principale comme étant analogue à celle d'une éponge. Je ne vous épargnerais donc pas cet exemple. 🍴 Dès qu'on met une éponge en contact avec de l'eau, elle l'absorbe. Tandis que lorsqu'on la presse, elle se vide de l'eau qu'elle a absorbée. Le rapport avec l'élément piézoélectrique ? Eh bien il agit un peu de la même manière. Un élément piézoélectrique, lui, subit un phénomène semblable : dès qu'on lui admet une contrainte mécanique, il génère une tension électrique. En revanche, dès qu'on lui administre une tension électrique, il génère alors une contrainte mécanique, restituée par exemple sous forme sonore.



Génération d'une tension électrique par un élément piézoélectrique sous l'action d'une contrainte mécanique – source Wikipédia

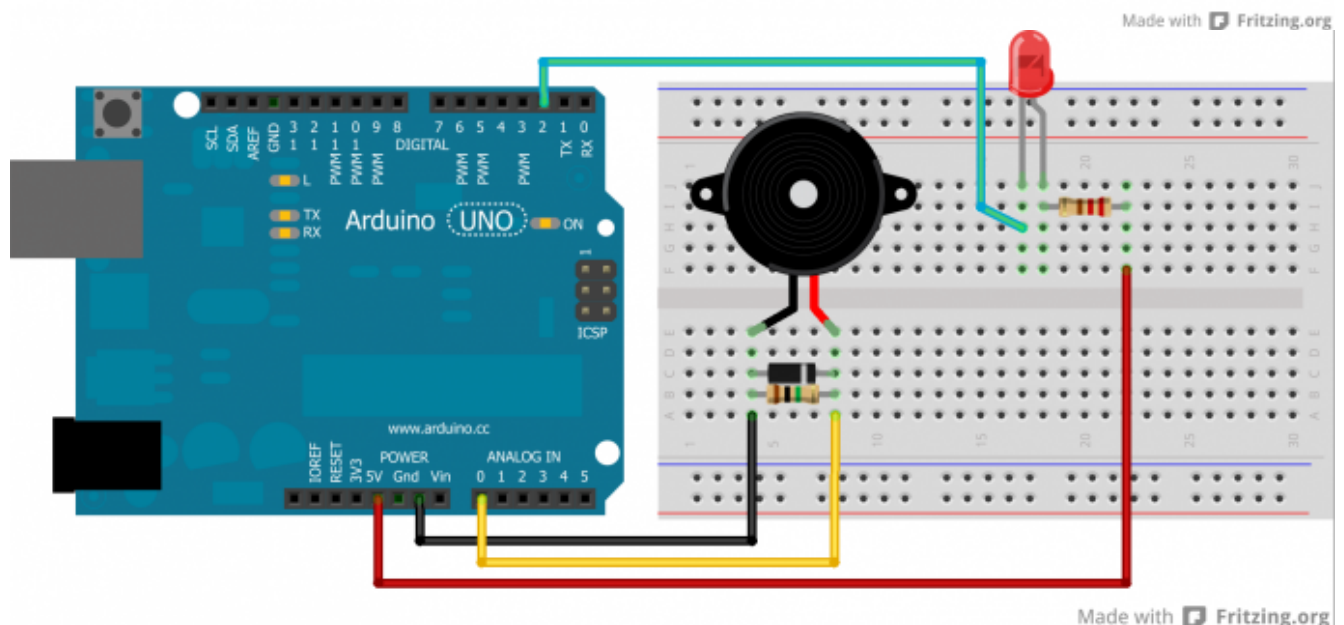
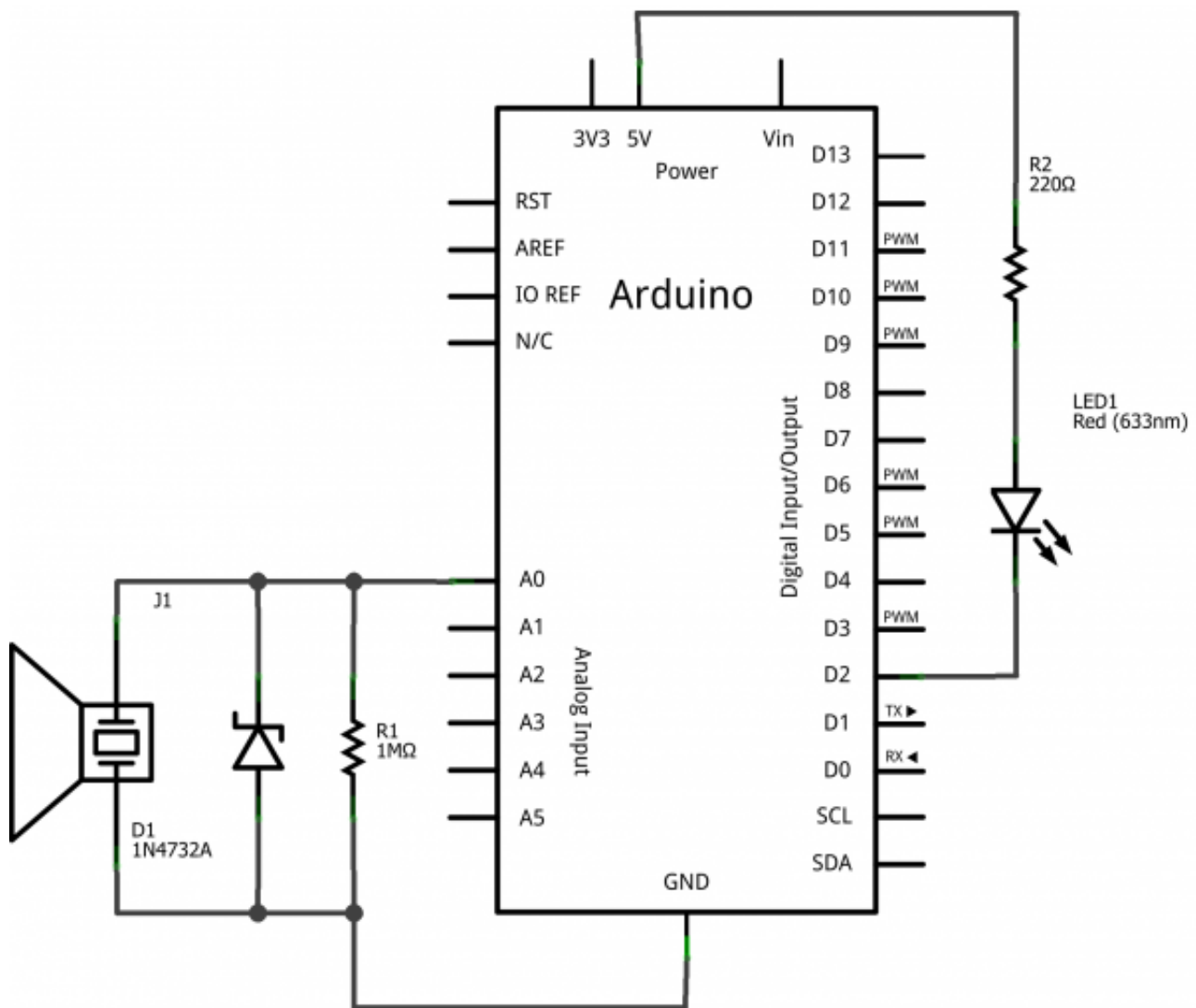
Un exemple d'utilisation dont vous ne vous doutez certainement pas, c'est l'utilisation de cette propriété dans certaines raquettes de tennis. L'élément piézoélectrique se trouve dans le manche de la raquette. Lorsqu'une balle frappe la raquette, elle génère une contrainte mécanique sur l'élément piézoélectrique qui en retour génère une tension électrique. Cette tension est récupérée et injectée à nouveau dans l'élément piézoélectrique qui génère alors une contrainte mécanique opposée à celle générée par la balle. Vous me suivez ? L'intérêt ? Réduire les vibrations causées par le choc et ainsi améliorer la stabilité de la raquette (et de la frappe) tout en réduisant le "stress" provoqué sur le poignet du joueur. Dingue non ?

### *Utilisation*

L'utilisation que nous allons faire de cet élément va nous permettre de capter un choc. Cela peut être un "toc" sur une porte, une déformation de surface, voire même une onde sonore un peu puissante. Ce capteur délivre directement une tension proportionnelle à la contrainte mécanique qu'on lui applique. Il s'agit donc d'un capteur actif. Nous pouvons donc l'utiliser sans rien en le connectant directement avec Arduino.

### Montage

Vous allez procéder au montage suivant en respectant le schéma de câblage :



La résistance de  $1M\Omega$  en parallèle de l'élément piézoélectrique permet d'éviter les courants trop forts qui peuvent être générés par l'élément piézoélectrique. Il est accompagné par une diode un peu particulière que l'on appelle une diode zener. Cette dernière sert à éviter les surtensions. Si jamais la tension générée par le piezo dépasse son seuil (4.7V en l'occurrence), elle deviendra passante et le courant ira donc vers la

masse plutôt que dans le microcontrôleur (évitant ainsi de griller l'entrée analogique). Cette dernière n'est pas indispensable mais conseillée cependant.

## Programme

Le programme que nous allons associer à ce montage, et qui va être contenu dans la carte Arduino, va exploiter la tension générée par l'élément piézoélectrique, lorsqu'on lui administrera une contrainte mécanique, pour allumer ou éteindre une LED présente en broche 2 de la carte Arduino. On pourra s'en servir pour détecter un événement sonore tel que le toc sur une porte. La condition pour que l'élément piézoélectrique capte correctement le toc d'une porte est qu'il doit être positionné sur la porte de façon à ce que sa surface soit bien plaquée contre elle. Aussi nous utiliserons la liaison série pour indiquer la tension produite par l'élément piézoélectrique. C'est un petit plus, par forcément utile mais qui vous donnera une idée de la force qu'il faut pour générer une tension particulière. Vous en trouverez certainement une application utile. 😊 Allez, un peu de programmation !

### Fonction setup()

Au début du programme nous déclarons quelques variables que nous utiliserons par la suite. Aussi nous amorçons l'utilisation de la liaison série et des broches utilisées de la carte Arduino.

```
1 const char led = 2;           // utilisation de la LED en broche 13 de la carte
2 const char piezo = 0;        // l'élément piézoélectrique est connecté en broche analogique
3 const int seuil_detection = 100;
4 /* on peut définir le seuil de détection qui va simplement
5 permettre de confirmer que c'est bien un événement sonore suffisant et non parasite */
```

Petite parenthèse par rapport au seuil. Ici il est configuré de façon à être comparé à la lecture directe de la valeur en broche analogique (comprise entre 0 et 1023). Mais on peut aussi le définir pour qu'il soit comparé au calcul de la tension en sortie du capteur (par exemple le mettre à 1, pour 1V).

```
1 float lecture_capteur = 0;    // variable qui va contenir la valeur lue en broche analogique
2 float tension = 0;           // variable qui va contenir le résultat du calcul de la tension
3 int etat_led = LOW;          // variable utilisée pour allumer ou éteindre la LED à chaque événement
4
5 void setup()
6 {
7     pinMode(led, OUTPUT);      // déclaration de la broche 13 en sortie
8     Serial.begin(9600);        // utilisation de la liaison série
9 }
```

### Fonction principale

Étant donné que le code est plutôt court et simple, nous le laisserons dans la seule fonction loop() plutôt que de le découper en plusieurs petites fonctions. Cependant libre à vous de l'agencer autrement selon vos besoins.

```
1 void loop()
2 {
3     lecture_capteur = analogRead(piezo);    // lecture de la valeur en sortie du capteur
```

```

4   tension = (lecture_capteur * 5.0) / 1024; // conversion de cette valeur en tens
5
6   if (lecture_capteur >= seuil_detection)
7   {
8       etat_led = !etat_led;           // on modifie l'état de la LED pour le pass
9       digitalWrite(led, etat_led); // application du nouvel état en broche 13
10
11      // envoi vers l'ordinateur, via la liaison série, des données correspondant
12      Serial.println("Toc !");
13      Serial.print("Tension = ");
14      Serial.print(tension);
15      Serial.println(" V");
16  }
17 }

```

Ici pas de délai à la fin de la boucle. En effet, si vous mettez un délai (qui est bloquant) vous risqueriez de rater des “toc” puisque cet événement est bref et imprévisible. Si jamais vous tapiez sur votre élément piézoélectrique au moment où le programme est dans la fonction `delay()`, vous ne pourriez pas l’intercepter.

## Seuil de tension

Comme je le disais, il est aussi possible et non pas idiot de changer le seuil pour qu’il soit comparé en tant que tension et non valeur “abstraite” comprise entre 0 et 1023. Cela relève de la simplicité extrême, voyez plutôt :

```

1  const char led = 2;           // utilisation de la LED en broche 13 de la carte
2  const char piezo = 0;        // l'élément piézoélectrique est connecté en broche anal
3  const float seuil_detection = 1.36; // seuil de détection en tension et non plus en
4
5  float lecture_capteur = 0;    // variable qui va contenir la valeur lue en broche anal
6  float tension = 0;           // variable qui va contenir le résultat du calcul de la
7  int etat_led = LOW;          // variable utilisée pour allumer ou éteindre la LED à c
8
9  void setup()
10 {
11     pinMode(led, OUTPUT);      // déclaration de la broche 13 en sortie
12     Serial.begin(9600);        // utilisation de la liaison série
13 }
14
15 void loop()
16 {
17     lecture_capteur = analogRead(piezo); // lecture de la valeur en sortie du ca
18     tension = (lecture_capteur * 5.0) / 1024; // convection de cette valeur en tens
19
20     if (tension >= seuil_detection) //comparaison de deux tensions
21     {
22         etat_led = !etat_led; // on modifie l'état de la LED pour le passer
23         digitalWrite(led, etat_led); // application du nouvel état en broche 13
24
25         // envoi vers l'ordinateur, via la liaison série, des données correspondant
26         Serial.println("Toc !");
27         Serial.print("Tension = ");
28         Serial.print(tension);
29         Serial.println(" V");
30     }
31 }

```

Je n'ai modifié que le type de la variable `seuil_detection`.

## La réversibilité de l'élément piézoélectrique

Tout à l'heure je vous disais que l'élément piézoélectrique était capable de transformer une contrainte mécanique en une tension électrique. Je vous ai également parlé du "phénomène éponge" en vous disant que l'on pouvait aussi bien absorber que restituer non pas de l'eau comme l'éponge mais une contrainte mécanique à partir d'une tension électrique. On va donc s'amuser à créer du son avec l'élément piézoélectrique ! 😊

### *Faire vibrer l'élément piézoélectrique !*

Attention tout de même, bien que vous l'aurez très certainement compris, il n'est plus question d'utiliser l'élément piézoélectrique en entrée comme un capteur, mais bien en sortie comme un actionneur.

Tout d'abord, il vous faudra brancher l'élément piézoélectrique. Pour cela, mettez son fil noir à la masse et son fil rouge à une broche numérique, n'importe laquelle. Pas besoin de résistance cette fois-ci. Et voilà les branchements sont faits ! Il ne reste plus qu'à générer un signal pour faire vibrer l'élément piézoélectrique. Selon la fréquence du signal, la vibration générée par l'élément piézoélectrique sera plus ou moins grave ou aiguë. Essayons simplement avec ce petit programme de rien du tout (je ne vous donne que la fonction `loop()`, vous savez déjà tout faire 😊) :

```
1 void loop()
2 {
3     digitalWrite(piezo, HIGH);
4     delay(5);
5     digitalWrite(piezo, LOW);
6     delay(5);
7 }
```

Ce code va générer un signal carré d'une période de 10ms, soit une fréquence de 100Hz. C'est un son plutôt grave. Vous pouvez aisément changer la valeur contenue dans les délais pour écouter les différents sons que vous allez produire. Essayez de générer un signal avec la PWM et soyez attentif au résultat en changeant la valeur de la PWM avec un potentiomètre par exemple.

### *Une fonction encore toute prête*

Maintenant, si vous souhaitez générer ce signal et en même temps faire d'autres traitements, cela va devenir plus compliqué, car le temps sera plus difficile à maîtriser (et les délais ne sont pas toujours les bienvenus 😊). Pour contrer cela, nous allons confier la génération du signal à une fonction d'Arduino qui s'appelle `tone()`. Cette fonction prend en argument la broche sur laquelle vous voulez appliquer le signal ainsi que la fréquence dudit signal à réaliser. Si par exemple je veux émettre un signal de 440Hz (qui correspond au "la" des téléphones) je ferais : `tone(piezo, 440);`. Le son va alors devenir permanent, c'est pourquoi, si vous voulez l'arrêter, il vous suffit d'appeler la fonction `noTone()` qui va alors arrêter la génération du son sur la broche spécifiée en argument.



La fonction `tone()` peut prendre un troisième argument qui spécifie en millisecondes la durée pendant laquelle vous désirez jouer le son, vous évitant ainsi d'appeler `noTone()` ensuite.

Pour les plus motivés d'entre vous, vous pouvez essayer de jouer une petite mélodie avec l'élément piézoélectrique. 😊 *Ah les joies nostalgiques de l'époque des sonneries monophoniques.* 😊

---

## Étalonner son capteur

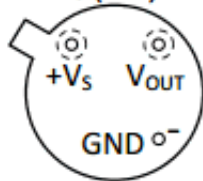
Faisons une petite pause dans notre découverte des capteurs pour parler d'un problème qui peut arriver à tout le monde... Comment faites-vous si vous possédez un capteur mais ne possédez pas sa caractéristique exacte ? Comment feriez-vous pour faire correspondre une valeur analogique lue avec une donnée physique réelle ? Par exemple, si je vous donne un composant en vous disant "hey, te voilà un capteur de température, je sais qu'il s'alimente en 5V sur telle et telle broches, je sais que le signal de sortie est une tension en fonction de la température mais je suis incapable de te dire quelle est la caractéristique (la courbe tension en fonction de la température)". Nous allons maintenant voir comment résoudre ce problème en voyant une méthode pour étalonner son capteur. Nous allons ainsi nous même déterminer la courbe caractéristique du capteur et déterminer son coefficient liant la température et la tension. À la fin, je vous donnerais la vraie courbe constructeur et nous pourrions comparer nos résultats pratiques avec ceux de référence 😊 .

### *Le capteur utilisé*

Pour étudier la méthode que je vous propose ici, nous allons utiliser un capteur de température assez répandu qui se nomme "LM35". Il existe dans différents boîtiers que voici :

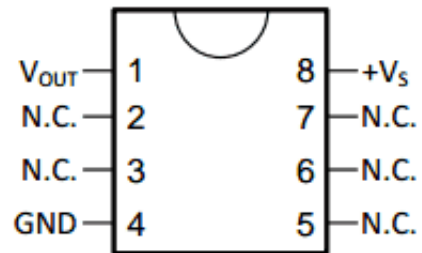
## CONNECTION DIAGRAMS

METAL CAN PACKAGE  
TO (NDV)



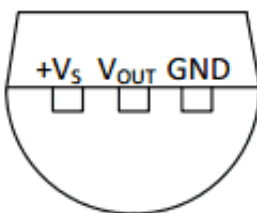
Case is connected to negative pin (GND)

SMALL-OUTLINE MOLDED PACKAGE  
SOIC-8 (D)  
TOP VIEW

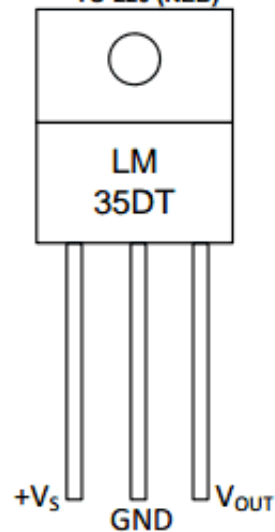


N.C. = No connection

PLASTIC PACKAGE  
TO-92 (LP)  
BOTTOM VIEW



PLASTIC PACKAGE  
TO-220 (NEB)



Tab is connected to the negative pin (GND).

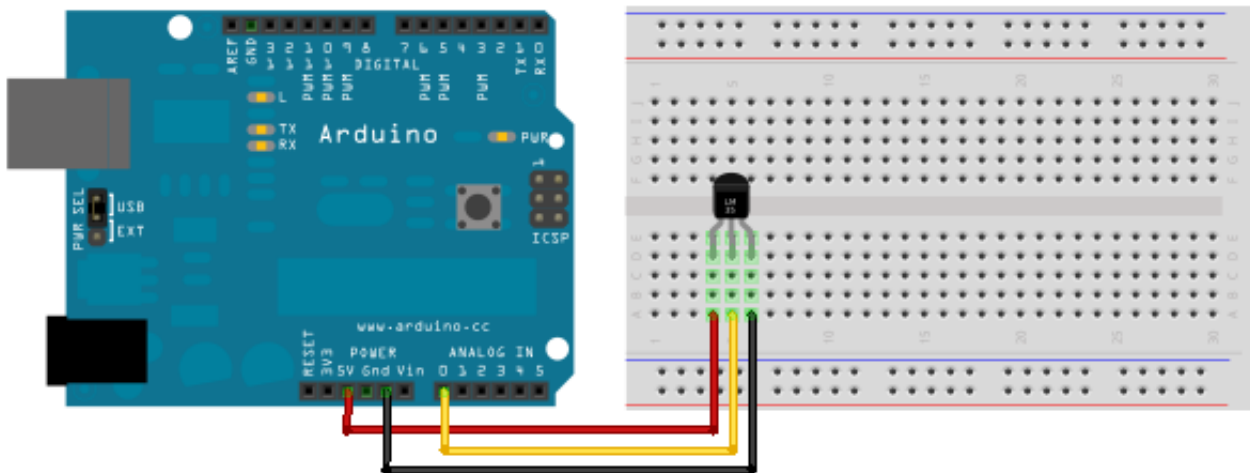
Vous aurez deviné le branchement, il est assez simple. Il suffit de relier +VS au 5V et GND à la masse. Le signal sera ensuite lu sur la broche Vout.

## La méthode

La méthode pour caractériser le capteur est assez simple. À l'aide d'une multitude de mesures et d'un appareil témoin, nous allons pouvoir créer un tableau qui nous servira à calculer la courbe (à l'aide d'un logiciel comme Excel par exemple). Pour cela, en plus de votre capteur vous aurez besoin d'un appareil de mesure "témoin" qui vous servira de référence. Par exemple le bon vieux thermomètre qui traîne accroché à votre fenêtre fera parfaitement l'affaire 😊.

### Prise de mesures

Vous êtes prêts, alors allons-y, commençons à travailler. Reliez le capteur à l'Arduino et l'Arduino à l'ordinateur, de la manière la plus simple possible, comme ceci par exemple :



Ensuite, nous devons récupérer les données envoyées par le capteur de manière régulière (ou rajoutez un bouton et faite des envois lors de l'appui 😊). Pour cela, voici un petit programme sans difficulté qui vous enverra les valeurs brutes ou converties en volts toutes les demi-secondes.

```

1  const int capteur = 0; //capteur branché sur la pin analogique 0
2  float tension = 0.0;
3  int valeur = 0;
4
5  void setup()
6  {
7      Serial.begin(9600);
8  }
9
10 void loop()
11 {
12     valeur = analogRead(capteur);
13     tension = (valeur*5.0)/1024;
14
15     Serial.print("Tension : ");
16     Serial.print(tension);
17     Serial.println(" V");
18     Serial.print("Valeur : ");
19     Serial.println(valeur);
20     Serial.println("-----");
21
22     delay(500);
23 }

```

Maintenant que tout est prêt, il nous faut un banc de test. Pour cela, préparez une casserole avec de l'eau contenant plein de glaçons (l'eau doit être la plus froide possible). Faites une première mesure avec votre capteur plongé dedans (attention, les broches doivent être isolées électriquement ou alors mettez l'ensemble dans un petit sac plastique pour éviter que l'eau n'aille faire un court-circuit). Faites en même temps une mesure de la température réelle observée à l'aide du thermomètre. Une fois cela fait, relevez ces mesures dans un tableau qui possèdera les colonnes suivantes :

- Température réelle (en °C)
- Tension selon Arduino (en V)
- Valeur brute selon Arduino

Quand la première mesure est faite, commencez à faire réchauffer l'eau (en la plaçant

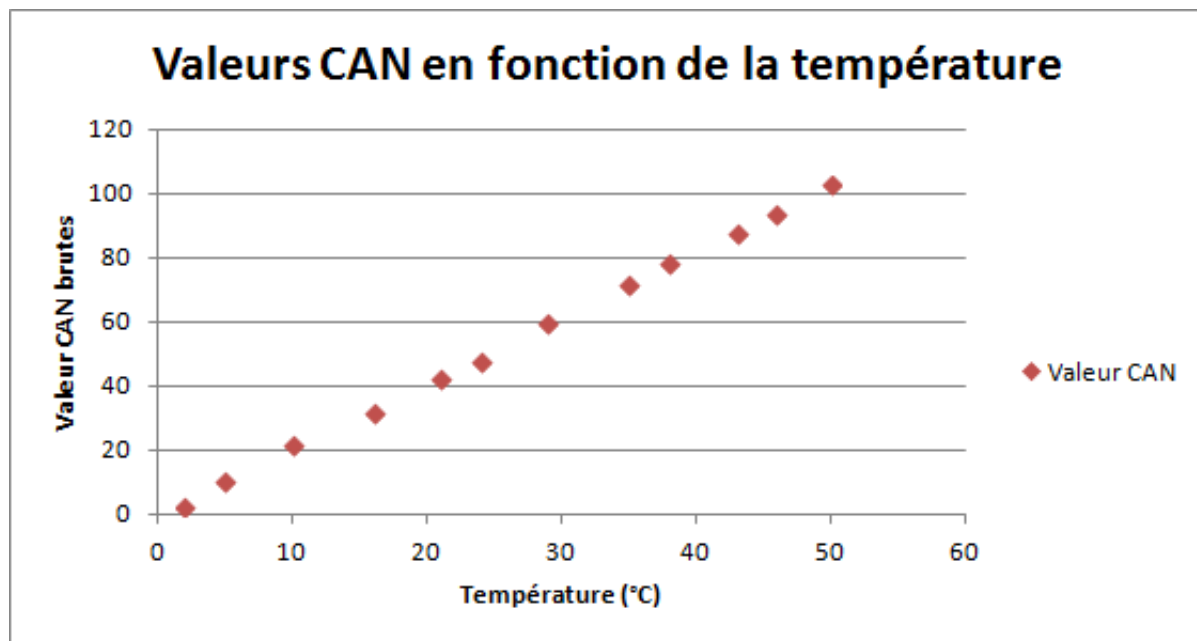
sur une plaque de cuisson par exemple). Continuez à faire des mesures à intervalle régulier (tous les 5 degrés voire moins par exemple). Plus vous faites de mesure, plus l'élaboration de la courbe finale sera précise. Voici à titre d'exemple le tableau que j'ai obtenu :

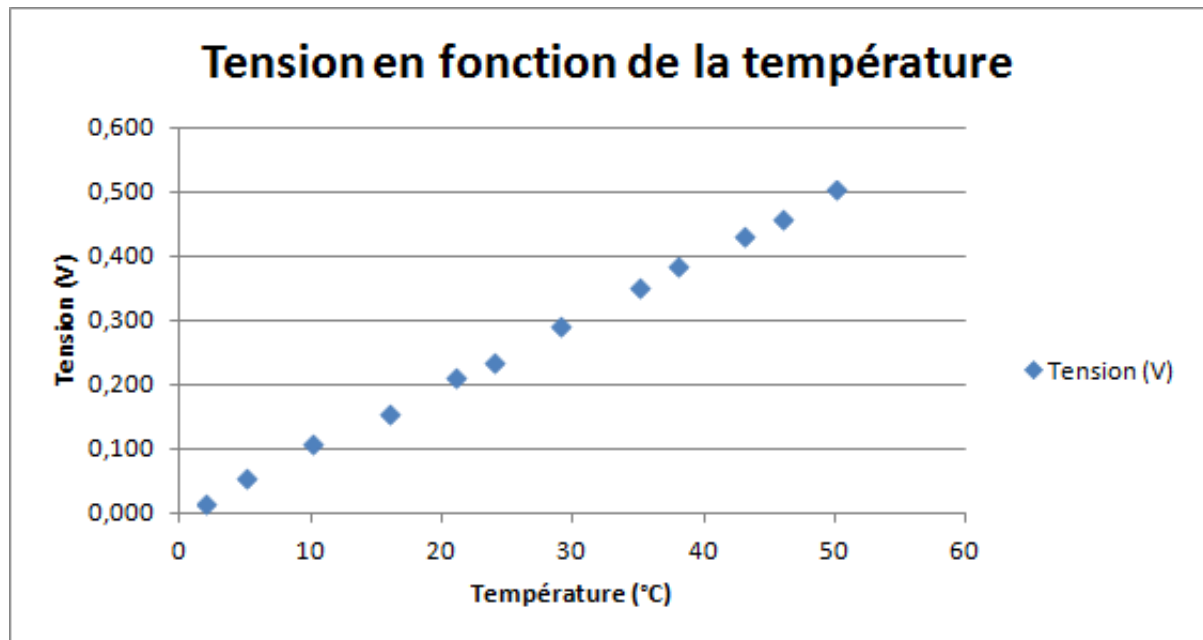
#### Température (°C) Tension (V) Valeur CAN

2	0,015	3
5	0,054	11
10	0,107	22
16	0,156	32
21	0,210	43
24	0,234	48
29	0,293	60
35	0,352	72
38	0,386	79
43	0,430	88
46	0,459	94
50	0,503	103

#### Réalisation de la caractéristique

Lorsque vous avez fini de prendre toutes vos valeurs, vous allez pouvoir passer à l'étape suivante qui est : Calculer la caractéristique de votre courbe !! Sortez vos cahiers, votre calculatrice et en avant ! ... Non j'blague (encore que ça ferait un super TP), on va continuer à utiliser notre logiciel tableur pour faire le travail pour nous ! On va donc commencer par regarder un peu l'allure de la courbe. Je vais en faire deux, une symbolisant les valeurs brutes de la conversion du CAN (entre 0 et 1023) en rouge et l'autre qui sera l'image de la tension en fonction de la température en bleu. Nous pourrons alors déterminer deux caractéristiques, selon ce qui vous arrange le plus.





Une fois cela fait, il ne reste plus qu'à demander gentiment au logiciel de graphique de nous donner la **courbe de tendance** réalisée par ces points. Sous Excel, il suffit de cliquer sur un des points du graphique et choisir ensuite l'option "Ajouter une courbe de tendance..." . Vous aurez alors le choix entre différents types de courbe (linéaire, exponentielle...). Ici, on voit que les points sont alignés, il s'agit donc d'une équation de courbe linéaire, de type  $y=ax+b$ . Cochez la case "Afficher l'équation sur le graphique" pour pouvoir voir et exploiter cette dernière ensuite.

**Format de courbe de tendance**

**Options de courbe de tendance**

Couleur du trait

Style de trait

Ombre

**Options de courbe de tendance**

Type de régression/de courbe de tendance

☐ Exponentielle

☒ Linéaire

☐ Logarithmique

☐ Polynomiale    Ordre : 2

☐ Puissance

☐ Moyenne mobile    Période : 2

Nom de la courbe de tendance

☒ Automatique : Linéaire (Tension (V))

☐ Personnalisé :

Prévision

Transférer : 0,0 périodes

Reculer : 0,0 périodes

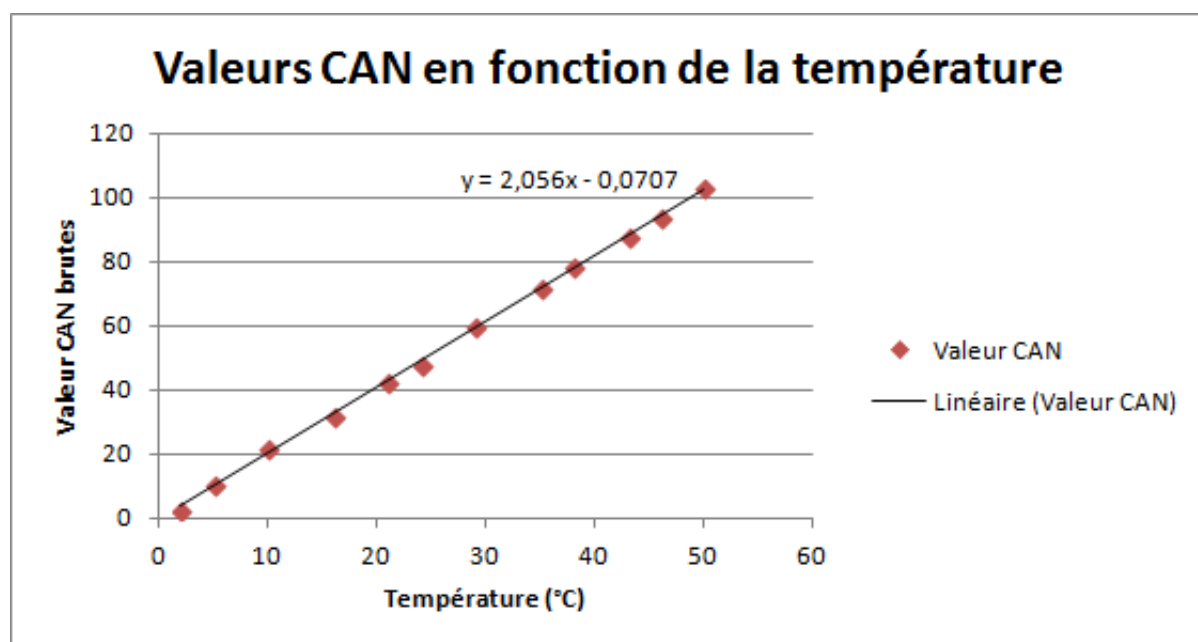
☐ Définir l'interception = 0,0

☒ Afficher l'équation sur le graphique

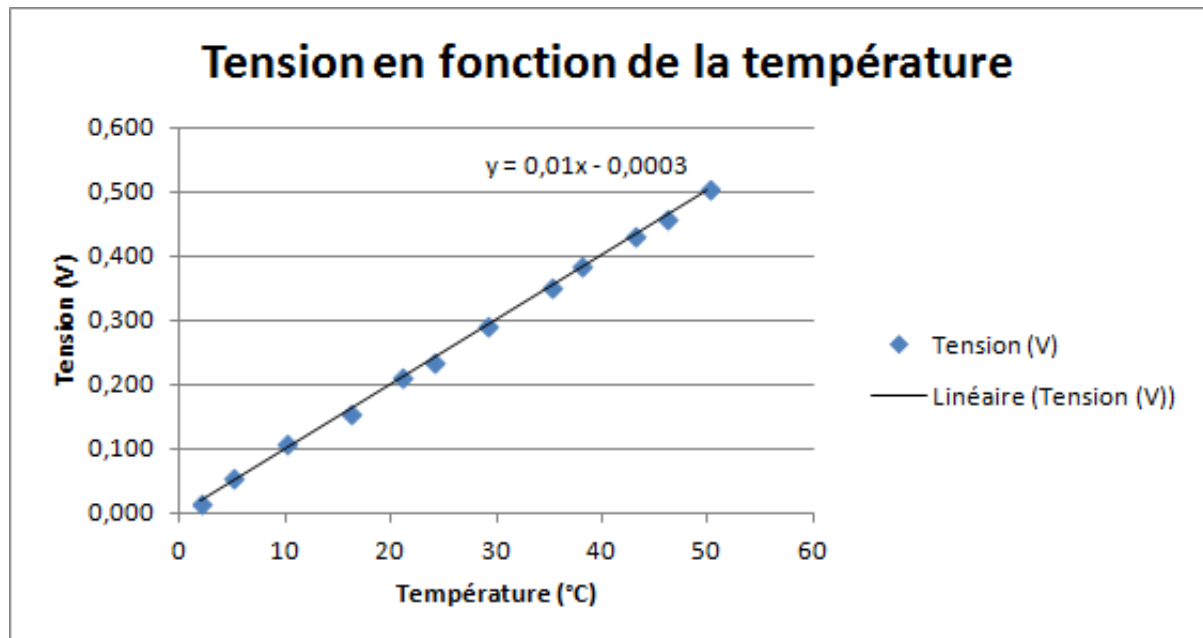
☐ Afficher le coefficient de détermination ( $R^2$ ) sur le graphique

Fermer

Voici alors ce que l'on obtient lorsque l'on rajoute notre équation :







Grâce à l'équation, nous pouvons déterminer la relation liant la température et la tension (ou les valeurs du CAN). Ici nous obtenons :

- $y = 0.01x - 0.0003$  (pour la tension)
- $y = 2.056x - 0.0707$  (pour les valeurs du CAN)

Le coefficient constant (-0.003 ou -0.0707) peut ici être ignoré. En effet, il est faible (on dit **négligeable**) comparé aux valeurs étudiées. Dans les équations, x représente la température et y représente la tension ou les valeurs du CAN. On lit donc l'équation de la manière suivante : Tension en Volt égale 0,01 fois la température en degrés celsius. Ce qui signifie que dorénavant, en ayant une mesure du CAN ou une mesure de tension, on est capable de déterminer la température en degrés celsius 😊 Super non ? Par exemple, si nous avons une tension de 300mV, avec la formule trouvée précédemment on déterminera que l'on a  $0.3 = 0.01 \times \text{Temperature}$ , ce qui équivaut à  $\text{Temperature} = 0.3/0.01 = 30^\circ\text{C}$ . On peut aisément le confirmer via le graphique 😊 Maintenant j'ai trois nouvelles, deux bonnes et une mauvaise... La bonne c'est que vous êtes capable de déterminer la caractéristique d'un capteur. La deuxième bonne nouvelle, c'est que l'équation que l'on a trouvé est correcte... parce qu'elle est marquée dans [la documentation technique](#) qui est super facile à trouver 😊 (ça c'était la mauvaise nouvelle, on a travaillé pour rien !! ) Mais comme c'est pas toujours le cas, c'est toujours bien de savoir comment faire 😊

## Adaptation dans le code

Puisque nous savons mesurer les valeurs de notre capteur et que nous avons une équation caractéristique, nous pouvons faire le lien en temps réel dans notre application pour faire une utilisation de la grandeur *physique* de notre mesure. Par exemple, s'il fait 50°C nous allumons le ventilateur. En effet, souvenez-vous, avant nous n'avions qu'une valeur entre 0 et 1023 qui ne signifiait physiquement pas grand chose. Maintenant nous sommes en mesure (oh oh oh 😊 ) de faire la conversion. Il faudra pour commencer récupérer la valeur du signal. Prenons l'exemple de la lecture d'une tension analogique du capteur précédent :

```
1 int valeur = analogRead(monCapteur); //lit la valeur
```

Nous avons ensuite deux choix, soit nous le transformons en tension puis ensuite en valeur physique grâce à la caractéristique du graphique bleu ci-dessus, soit nous transformons directement en valeur physique avec la caractéristique rouge. Comme je suis feignant, je vais chercher à économiser une instruction en prenant la dernière solution. Pour rappel, la formule obtenue était :  $y = 2.056x - 0.0707$ . Nous avons aussi dit que le facteur constant était négligable, on a donc de manière simplifiée  $y = 2.056x$  soit "la température est égale à la valeur lue divisé par 2.056 ( $x = \frac{y}{2.056}$  ). Nous n'avons plus qu'à faire la conversion dans notre programme !

```
1 float temperature = valeur/2.056;
```

. Et voilà ! Si l'on voulait écrire un programme plus complet, on aurait :

```
1 int monCapteur = 0; //Capteur sur la broche A0;
2 int valeur = 0;
3 float temperature = 0.0;
4
5 void setup()
6 {
7     Serial.begin(9600);
8 }
9
10 void loop()
11 {
12     valeur = analogRead(monCapteur);
13     temperature = valeur/2.056;
14
15     Serial.println(temperature);
16
17     delay(500);
18 }
```

Et si jamais notre coefficient constant n'est pas négligeable ?

Eh bien prenons un exemple ! Admettons qu'on obtienne la caractéristique suivante :  $y = 10x + 22$ . On pourrait lire ça comme "ma valeur lue par le CAN est égale à 10 fois la valeur physique plus 22". Si on manipule l'équation pour avoir x en fonction de y, on aurait :

$$\begin{aligned} y &= 10x + 22 \\ y - 22 &= 10x \\ x &= \frac{y-22}{10} \end{aligned}$$

Dans le code, cela nous donnerait :

```
1 void loop()
2 {
3     valeur = analogRead(monCapteur);
4     temperature = (valeur-22)/10;
5
6     Serial.println(temperature);
7
8     delay(500);
9 }
```

---

## [Arduino 503] Des capteurs plus évolués

Comme nous avons pu le voir plus tôt, certains capteurs transmettent l'information sous forme d'une donnée électrique qui varie : la résistance ou la tension.

Cependant, certains capteurs envoient l'information de manière "codée", afin qu'elle soit plus résistante au bruit (perturbations) et garantir le signal transmis. Parmi ces méthodes de transmission, on retrouve l'utilisation d'une PWM, de la fréquence ou d'un protocole de communication.

---

### Capteur à sortie en modulation de largeur d'impulsion (PWM)

#### Principe

Vous vous souvenez de la PWM ? Nous l'avons utilisée dans le chapitre sur [les sorties analogiques](#). Dans ce type de signal, l'information est présente dans la durée de l'état haut par rapport à l'état bas. Ici, notre capteur va donc de la même façon coder l'information via une durée d'état (mais elle ne sera pas forcément relative à son état antagoniste). Il est donc nécessaire de connaître les caractéristiques du capteur pour pouvoir interpréter le signal correctement.

En effet, si on prend le signal sans rien savoir du capteur, comment déterminer ce que 20ms d'état haut signifie par exemple ?

Pour cela, ce type de composant doit toujours être utilisé avec sa documentation technique, afin de déterminer des paramètres comme ses bornes inférieure et supérieure de mesure. Mais c'est aussi vrai pour les autres (si vous tombez sur une résistance variable sans rien en connaître, vous devrez vous farcir une belle séance d'étalonnage pour l'identifier !).

#### Utilisation

Prenons un cas simple. Imaginons que nous avons un capteur de température qui nous renvoie l'information suivante : *"La température en °C Celsius est proportionnelle à la durée d'état haut. La plage de mesure va de 0°C à 75°C pour une durée d'état haut de 0ms à 20ms"*.

Nous avons donc une relation proportionnelle entre une température et une durée. Nous pouvons alors déduire une règle mathématique pour faire la conversion degrés/durée.

En effet, on a les équivalences suivantes :

- $0^{\circ}\text{C} \Leftrightarrow 0\text{ms} \Leftrightarrow 0\%$
- $75^{\circ}\text{C} \Leftrightarrow 20\text{ms} \Leftrightarrow 100\%$

Une simple règle de trois nous donne :  $x = \frac{75}{20} = 3.75^{\circ}\text{C/ms}$  ce qui signifie que pour chaque milliseconde, on a  $3.75^{\circ}\text{C}$ .  
Voyons maintenant comment l'utiliser...

## Dans la pratique avec Arduino

Bon, c'est pas mal on a le côté théorique de la chose, mais ça ne nous dit toujours pas comment on va l'exploiter avec notre Arduino. En effet, générer une PWM on sait faire, mais mesurer une durée d'état haut ça on ne sait pas !

Et bien rassurez-vous, comme d'habitude c'est assez simple. En effet, il existe une fonction dans le framework Arduino qui sert exactement à cela, mesurer une durée d'état haut ou bas.

Cette fonction s'appelle `pulseIn()`. Elle prend simplement en paramètres la broche sur laquelle vous voulez faire la mesure et l'état que vous voulez mesurer (HIGH ou LOW). En option, un troisième paramètre permettra de spécifier un "timeout", un temps maximal à attendre avant de décider que la mesure n'est pas possible. Si le timeout est de 2 secondes et que l'état à mesurer n'a pas commencé 2 secondes après l'appel de la fonction, alors cette dernière retournera 0. Dernier détail, l'intervalle de mesure de la fonction est de 10µs à 3 minutes et renvoie un **unsigned long** représentant la durée de l'état en **microsecondes**.

Voilà, vous savez tout pour utiliser cette fonction ! Il ne faut pas oublier cependant que la broche sur laquelle nous allons faire la mesure doit être placée en INPUT lors du `setup()` 😊.

Reprenons maintenant l'exemple commencé ci-dessus.

Pour mesurer la température, nous allons mesurer l'état haut en sachant que celui-ci sera proportionnel à la température. Un code simple serait donc :

```
1  const char capteur = 2; //en admettant que le capteur de température soit sur la bro
2
3  void setup()
4  {
5      pinMode(capteur, INPUT);
6      Serial.begin(9600); //pour afficher la température
7  }
8
9  void loop()
10 {
11     unsigned long duree = pulseIn(capteur, HIGH, 25000);
12     //dans notre exemple la valeur est dans l'intervalle [0, 20000]
13     float temperature = duree*0.00375; // 3.75 °C par ms donc 0.00375 °C par µs
14
15     /* Dans notre cas, on n'utilise pas "map()" car la fonction fait des arrondis, ce
16
17     Serial.print("Duree lue : ");
18     Serial.println(duree, DEC);
19     Serial.print("Temperature : ");
20     Serial.println(temperature);
21
22     delay(200); //pour ne pas spammer la voie série
23 }
```

## Simulation de l'exemple

Si comme moi vous n'avez pas de capteur retournant une PWM, voici un petit montage tout simple permettant de tester ce concept.

Pour cela, nous allons utiliser une PWM de l'Arduino ! En effet, on sait depuis le chapitre "[Sorties analogiques](#)" faire varier un rapport cyclique dans une PWM, on va donc l'appliquer ici pour tester `pulseIn()`.

Pour cela, reliez une broche PWM à la broche qui vous sert de capteur puis essayez de réaliser ce que l'on vient de voir.

La fréquence de la PWM via Arduino est d'environ 490Hz, ce qui signifie que la durée d'état haut pourra varier entre 0ms et 2,04ms

```
1  const char capteur = 2; //broche capteur
2  const char emetteur = 3; //broche PWM
3
4  void setup()
5  {
6      pinMode(capteur, INPUT);
7      pinMode(emetteur, OUTPUT);
8
9      Serial.begin(9600);
10 }
11
12 void loop()
13 {
14     analogWrite(emetteur, 127); //test avec une valeur moyenne : environ 1ms
15     unsigned long duree = pulseIn(capteur, HIGH);
16
17     Serial.print("Duree : ");
18     Serial.println(duree, DEC); //vérifie qu'on a bien la durée attendue
19
20     delay(250);
21 }
```

## Étude de cas : le capteur de distance SRF05

Prenons un exemple, le télémètre ultrason SRF05 dont la doc. technique a été retranscrite [ici](#).

Ce composant est l'exemple classique du capteur renvoyant un créneau codant l'information. En effet, ce dernier mesure ce que l'on appelle un **temps de vol**. Explications !

Le SRF05 est un télémètre ultra-son. Pour mesurer une distance, il compte le temps que met une onde pour faire un aller-retour. Un chronomètre est déclenché lors du départ de l'onde et est arrêté lorsque l'on détecte le retour de l'onde (une fois que celle-ci a "rebondi" sur un obstacle). Puisque l'on connaît la vitesse de propagation ( $V$ ) de l'onde dans l'air, on peut déterminer la distance ( $d$ ) nous séparant de l'objet. On a donc la formule :  $v = \frac{d}{t}$  soit  $d = t \times v$ .

Le temps mesuré correspond à l'aller ET au retour de l'onde, on a donc deux fois la distance. Il ne faudra pas oublier de diviser le résultat par deux pour obtenir la distance réelle qui nous sépare de l'objet.

Comme expliqué dans la documentation, pour utiliser le sonar il suffit de générer un état haut pendant 10  $\mu$ s puis ensuite mesurer l'état haut généré par le sonar.

Ce dernier représente le temps que met l'onde à faire son aller-retour. Si l'onde met plus de 30ms à faire son voyage, elle est alors considérée comme perdue et la ligne repasse à LOW.

Et voilà, vous avez maintenant toutes les informations pour faire un petit programme d'essai pour utiliser ce sonar.

Ah, une dernière information... La vitesse d'une onde sonore dans l'air à 15°C est de 340 mètres par seconde. Ça pourrait être utile !

```
1 #define VITESSE 340 //vitesse du son 340 m/s
2
3 const int declencheur = 2; // la broche servant à déclencher la mesure
4 const int capteur = 3; // la broche qui va lire la mesure
5
6 void setup()
7 {
8     pinMode(declencheur, OUTPUT);
9     pinMode(capteur, INPUT);
10
11     digitalWrite(declencheur, LOW);
12     Serial.begin(9600);
13 }
14
15 void loop()
16 {
17     digitalWrite(declencheur, HIGH);
18     delayMicroseconds(10); //on attend 10 µs
19     digitalWrite(declencheur, LOW);
20
21     //puis on récupère la mesure
22     unsigned long duree = pulseIn(capteur, HIGH);
23
24     if(duree > 30000)
25     {
26         //si la durée est supérieure à 30ms, l'onde est perdue
27         Serial.println("Onde perdue, mesure echouee !");
28     }
29     else
30     {
31         //l'onde est revenue ! on peut faire le calcul
32         duree = duree/2; //on divise par 2 pour n'avoir qu'un trajet (plutôt que l'all
33         float temps = duree/1000000.0; //on met en secondes
34         float distance = temps*VITESSE; //on multiplie par la vitesse, d=t*v
35
36         Serial.print("Duree = ");
37         Serial.println(temps); //affiche le temps de vol d'un trajet en secondes
38         Serial.print("Distance = ");
39         Serial.println(distance); //affiche la distance mesurée
40     }
41
42     delay(250);
43 }
```

## Capteur à signal de sortie de fréquence variable

Voyons maintenant un autre type de sortie très similaire à celui vu ci-dessus, la fréquence. Les capteurs de ce type vont donc vous délivrer une fréquence variable en fonction de la valeur mesurée. Je ne vais pas vous mentir, je n'ai pas d'exemple en tête !

Cependant, il est facile d'imaginer comment les utiliser en prenant en compte ce que l'on vient de voir pour le capteur renvoyant une PWM.

En effet, considérons un capteur nous envoyant un signal de type “créneau” à une fréquence  $f$ . Un créneau possède en théorie une durée d’état haut égale à la durée de l’état bas. Si on fait donc une mesure de cette durée d’état haut via `pulseIn()` vue précédemment, on peut aisément déduire la période (T) du signal (qui sera égale à deux fois la valeur lue) et ainsi la fréquence puisque  $f = 1/T$ .  
De manière programmatrice, on obtiendra donc le code suivant :

```
1  const char capteur = 2; //broche sur laquelle est branchée le capteur
2
3  void setup()
4  {
5      pinMode(capteur, INPUT);
6
7      Serial.begin(9600);
8  }
9
10 void loop()
11 {
12     unsigned long duree = pulseIn(capteur, HIGH); //ou LOW, ce serait pareil !
13     duree = duree*2; //pour avoir la période complète
14
15     float frequence = 1.0/duree; //hop! on calcule la fréquence !
16     frequence = frequence*1000000; //passe la fréquence en Hz (car la période était m
17     Serial.print("Frequence = ");
18     Serial.println(frequence);
19
20     delay(250);
21 }
```

## Exemple / Exercice

Afin de mettre tout cela en pratique, je vous propose un petit exercice pour mettre en œuvre ce dernier point. Peu de matériel est à prévoir.

### Principe

Pour cet exercice, je vous propose d’émuler le comportement d’un capteur générant une fréquence variable.

Nous allons utiliser un potentiomètre qui va nous servir de “variateur”. Ensuite nous allons utiliser une fonction propre à Arduino pour générer une fréquence particulière qui sera l’image multipliée par 10 de la valeur mesurée du potentiomètre. Enfin, nous allons “reboucler” la sortie “fréquence” sur une entrée quelconque sur laquelle nous mesurerons cette fréquence.

C’est clair ? J’espère !

#### La fonction `tone()`

Pour faire cette exercice vous connaissez déjà tout à une chose près : Comment générer une fréquence. Pour cela, je vous propose de partir à la découverte de la fonction `tone()`. Cette dernière génère une fréquence sur une broche, n’importe laquelle. Elle prend en paramètre la broche sur laquelle le signal doit être émis ainsi que la fréquence à émettre. Par exemple, pour faire une fréquence de 100Hz sur la broche 3 on fera simplement : `tone(3, 100);`.



Un troisième argument peut-être utilisé. Ce dernier sert à indiquer la durée pendant laquelle le signal doit être émis. Si on omet cet argument, la fréquence sera toujours générée jusqu'à l'appel de la fonction antagoniste `noTone()` à laquelle on passe en paramètre la broche sur laquelle le signal doit être arrêté.

L'utilisation de la fonction `tone` interfère avec le module PWM des broches 3 et 11. Gardez-le en mémoire 😊. Cette fonction ne peut pas non plus descendre en dessous de 31Hz.

Vous avez toutes les informations, maintenant à vous de jouer !

## Correction

Voici ma correction commentée. Comme il n'y a rien de réellement compliqué, je ne vais pas faire des lignes d'explications et vous laisser simplement avec le code et ses commentaires 🤖!

```
1  const char potar = 0; //potentiomètre sur la broche A0;
2  const char emetteur = 8; //fréquence émise sur la broche 8
3  const char recepneur = 2; //fréquence mesurée sur la broche 2
4
5  void setup()
6  {
7      pinMode(emetteur, OUTPUT);
8      pinMode(recepneur, INPUT);
9
10     Serial.begin(9600);
11 }
12
13 void loop()
14 {
15     unsigned int mesure = 100; //fait la lecture analogique (intervalle [0;1023] )
16
17     tone(emetteur, mesure*10); //applique la mesure comme fréquence (intervalle [0;10
18
19     unsigned long periode = pulseIn(recepneur, HIGH); //mesure la demi-période
20     periode = periode*2; //pour avoir une période complète on multiplie par 2
21     float frequence = 1.0/periode; //transforme en fréquence
22     frequence = frequence*1000000; //passe la fréquence en Hz (car la période était m
23
24     Serial.print("Mesure : ");
25     Serial.println(mesure, DEC);
26     Serial.print("Periode : ");
27     Serial.println(periode, DEC);
28     Serial.print("Frequence : ");
29     Serial.println(frequence);
30
31     delay(250);
32 }
```

## Capteur utilisant un protocole de communication

Certains capteurs ne renvoient pas l'information sous forme "physique" dans le sens où ils ne renvoient pas quelque chose de mesurable directement, comme un temps ou une tension. Non, ces derniers préfèrent envoyer l'information encapsulée bien au chaud dans une trame d'un protocole de communication.

Le gros intérêt de cette méthode est très probablement la résistance au "bruit". Je ne parle bien sûr pas des cris des enfants du voisin ou les klaxons dans la rue mais bien de bruit électronique. Ce dernier est partout et peut avoir des conséquences ennuyeuses sur vos mesures. Transmettre l'information par un protocole de communication est donc un moyen fiable de garantir que la donnée arrivera de manière intègre jusqu'au destinataire. De plus, on peut facilement coupler cette transmission avec un protocole de vérification simple ou compliqué (comme la vérification de parité ou un calcul de CRC).

Cette partie ne va pas vous enseigner comment utiliser chacun des moyens de communication que nous allons voir. En effet, il s'agit plutôt d'une introduction/ouverture sur l'existence de ces derniers. Ils seront traités de manière indépendante dans des chapitres dédiés comme le fût [la voie série](#).

## Quelques protocoles de communication

### Voie série / UART

Ce protocole vous devez déjà le connaître par cœur puisque nous l'utilisons presque dans tous les chapitres ! Il s'agit en effet de la voie série via `Serial`. Rien de réellement compliqué donc tellement vous êtes habitués à le voir !

Ceci est une liaison point-à-point, donc seuls deux composants (l'Arduino et le capteur) peuvent être reliés entre eux directement. Elle est généralement bi-directionnelle, ce qui signifie que les deux composants reliés peuvent émettre en même temps.

### I2C

Le protocole I<sup>2</sup>C (Inter-Integrated Circuit) ou TWI (Two Wire Interface) permet d'établir une liaison de type "maître/esclave". L'Arduino sera maître et le capteur l'esclave (l'Arduino peut aussi être un esclave dans certains cas). Ainsi, l'Arduino émettra les ordres pour faire les demandes de données et le capteur, lorsqu'il recevra cet ordre, la renverra.

Ce protocole utilise 3 fils. Un pour la masse et ainsi avoir un référentiel commun, un servant à émettre un signal d'horloge (SCL) et un dernier portant les données synchronisées avec l'horloge (SDA).

Chez Arduino, il existe une librairie pour utiliser l'I2C, elle s'appelle `Wire`.

On peut placer plusieurs esclaves à la suite. Un code d'adresse est alors utilisé pour décider à quel composant le maître fait une requête.

### SPI

Le SPI (Serial Peripheral Interface) est une sorte de combo entre la voie série et l'I2C. Elle prend le meilleur des deux mondes.

Comme en voie série, la liaison est bi-directionnelle et point-à-point\*. Cela signifie que l'Arduino et le capteur sont reliés directement entre eux et ne peuvent que parler entre eux. Cela signifie aussi que les deux peuvent s'envoyer des données simultanément.

Comme en I2C, la liaison est de type maître/esclave. L'un fait une demande à l'autre, le

maître transmet l'horloge à l'esclave pour transmettre les données.

Cette transmission utilise 4 fils. Une masse pour le référentiel commun, un fil d'horloge (SCLK), un fil nommé MOSI (Master Output, Slave Input; données partant de l'Arduino et allant vers le capteur) et MISO (Master Input, Slave Output, données partant du capteur et allant vers l'Arduino).

\* Afin d'améliorer cette voie série, il existe une autre broche nommée SS (Slave Select) permettant de choisir à quel composant le maître parle. Ainsi la liaison n'est plus limitée à un esclave seulement.

Chez Arduino, il existe une librairie pour utiliser le SPI, elle s'appelle ... **SPI**.

## ***Protocole propriétaire***

Ici pas de solution miracle, il faudra manger de la documentation technique (très formateur !). En effet, si le constructeur décide d'implémenter un protocole à sa sauce alors ce sera à vous de vous plier et de coder pour réussir à l'implémenter et l'utiliser.

---

Vous savez maintenant tout sur les capteurs, votre Arduino peut maintenant "sentir" le monde qui l'entoure comme promis en introduction de cette partie. Mais ce n'est pas fini, il existe un grand nombre de capteurs, beaucoup trop important pour en faire une liste exhaustive dans un tutoriel comme celui-ci.

Maintenant que vous pouvez percevoir le monde, passons à la suite en essayant d'interagir avec ce dernier grâce à [l'utilisation de moteurs](#)...