Partie 4

Les capteurs électroniques

Vous avez vu jusqu'à présent que l'on pouvait récupérer des informations depuis l'environnement de l'Arduino grâce à deux commandes : `digitalRead(pinDigital)` et `analogRead(pinAnalogique)`.
Ce ne sont pas les deux seules commandes mais elles nous permettent de récupérer des valeurs provenant de plusieurs sortes de capteurs qui vont transformer votre Arduino en une machine sensible à son environnement.

Nous allons donc voir dans ce chapitre les types de capteurs existants (enfin une partie seulement ) ainsi que leurs connexions vers l'Arduino.

Nous verrons aussi comment récupérer et utiliser les différentes valeurs lues par la carte.

Bien sûr, afin de pouvoir utiliser et programmer ces capteurs, il est nécessaire de se les procurer. Si vous n'avez pas encore le capteur sous la main, rassurez-vous, le cours est compréhensible tout de même et il vous permettra peut-être de choisir le type de capteur dont vous avez besoin pour vos projets...

Le matériel requis

Voici donc la liste des capteurs que je vais utiliser pour illustrer ce chapitre.

- Capteur de contact
- Tilt sensor
- Photorésistance
- Photodiode infrarouge (et LED infrarouge)
- Récepteur infrarouge 38 KHz (TSOP38238)
- Capteur de température (TMP36)
- Capteur à ultrasons (SRF05)

Les types de capteurs

Il existe deux grandes familles de capteurs :

Les capteurs tout-ou-rien

Comme leur nom l'indique, ils ne fournissent que deux données (souvent HIGH et LOW) en fonction de leur état de connexion. On utilise donc pour lire ces données les ports numériques de l'Arduino. On stocke souvent le résultat dans une variable de type booléenne qui sert directement pour les tests.

Les montages utilisés feront souvent intervenir une résistance pull-up ou pull-down pour éviter d'obtenir des valeurs erratiques. L'utilisation de la résistance pull-up interne de l'Arduino est aussi une bonne méthode pour simplifier le circuit, il suffit juste de penser à modifier les tests (voir le chapitre sur le bouton poussoir).

Les capteurs de variation

Ils font varier la tension de sortie soit en résistant, soit en produisant un courant. Ces capteurs seront donc lus avec les ports analogiques de l'Arduino. On transforme alors la valeur de la tension récupérée (souvent entre 0V et 5V) en un nombre entier entre 0 et 1024.

Pour le montage on utilisera souvent une résistance comme pont diviseur de tension. Je n'en expliquerai pas le principe pour le moment, les schémas que je vous proposerai seront suffisants pour correctement placer les composants.

Il vous sera utile, pour ce type de capteur, de réaliser une sorte d'étalonnage pour évaluer les limites de variation de la tension liée au capteur. Je vous fournis un programme dans ce chapitre qui peut vous y aider, mais vous concevrez vite le vôtre, j'en suis sûr !

Allez, voyons ce qui s'offre à nous !

Le capteur de contact

C'est une sorte de bouton poussoir qui s'actionne grâce à la force exercée sur un petit levier. Il en existe de plusieurs formes. En voici quelques uns :

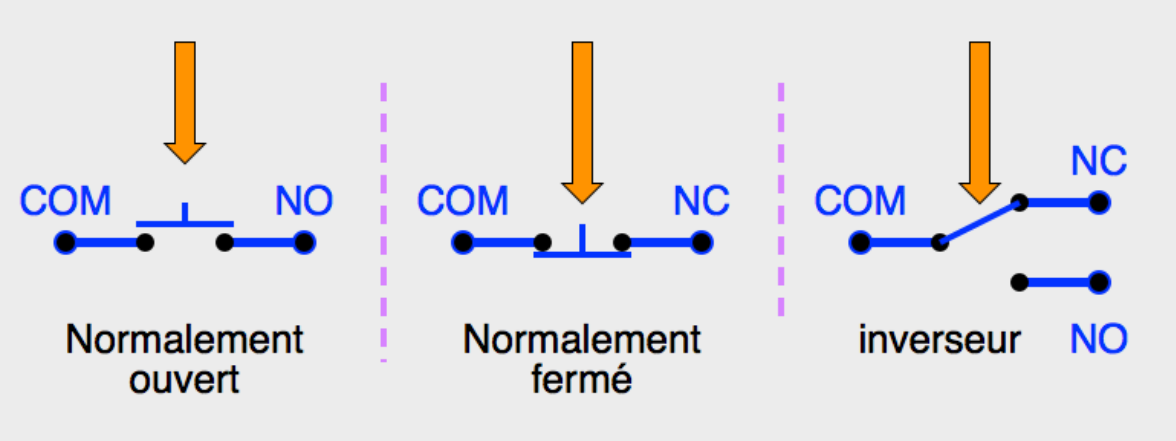
Les différents types de microrupteurs (robotastuces.free.fr)

Ils font partie de la famille des capteurs tout-ou-rien. On les appelle aussi capteurs de collision, ou microrupteurs (car ils sont petits et permettent de laisser passer ou non le courant). Ils sont assez faciles à fixer mais il faut prévoir une soudure pour relier les pattes au montage.

Ce capteur peut avoir 2 ou 3 pattes :

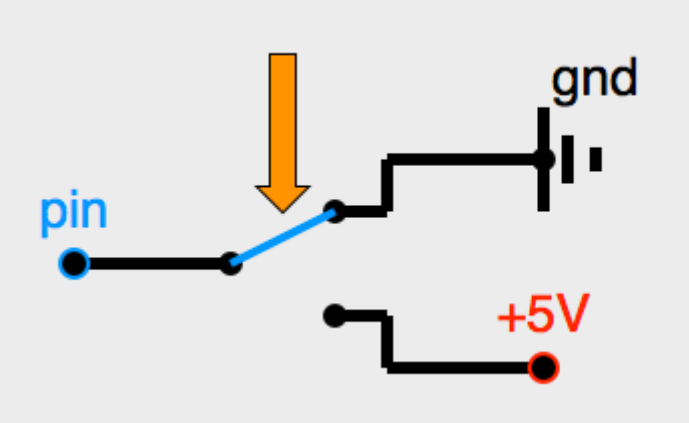
- **S'il a 2 pattes, c'est un interrupteur simple** : soit il est ouvert et il se ferme au contact (type "NO" pour *Normally Open*, c'est-à-dire normalement ouvert) ; soit il est fermé et il s'ouvre au contact (type "NC" pour *Normally Closed,* c'est-à-dire normalement fermé)
- **S'il a 3 pattes, c'est un inverseur** : il laisse passer le courant vers l'une ou l'autre des pattes. Il peut bien sûr être utilisé comme un 2 pattes.

Voici les principes des trois types de microrupteurs :



Les capteurs de contacts
Pour les connecter à l'Arduino, ce n'est pas compliqué.

- Pour le microrupteur à 2 pattes, vous faites comme pour un bouton poussoir avec une résistance en mode pull-up ou pull-down ou en utilisant le mode INPUT_PULLUP du pin concerné (voir chapitre sur le bouton poussoir).
- Pour le microrupteur à 3 pattes (c'est-à-dire l'inverseur), c'est presque plus simple : la patte COM sur le pin de lecture, la patte NC sur le ground et la patte NO sur le +5V :



Connexion d'un microrupteur inverseur à l'Arduino

Pour le reste, il se programme comme un bouton poussoir. On récupère sur le pin une valeur haute (HIGH ou 1) ou basse (LOW ou 0) en fonction d'un contact ou non.

Voici un petit programme simple qui affiche si le contact est réalisé ou non :

```
int pinContact=7; // pin de lecture du contact

void setup() {

  Serial.begin(9600); // initialisation de la connexion série

  pinMode(pinContact,INPUT_PULLUP); //active la résistance pull-up interne

}
```

```
void loop() {

  boolean etatContact=digitalRead(pinContact);

  if (!etatContact) //test inverse car mode INPUT_PULLUP

    Serial.println("Contact");

  else

    Serial.println("Pas de contact");

}
```

Vous remarquerez que je n'ai pas mis d'accolade après le `if` et le `else` . En fait, si il n'y a qu'**une instruction** qui suit un test ou une boucle, on peut se passer des accolades.

Quand utiliser un microrupteur ?

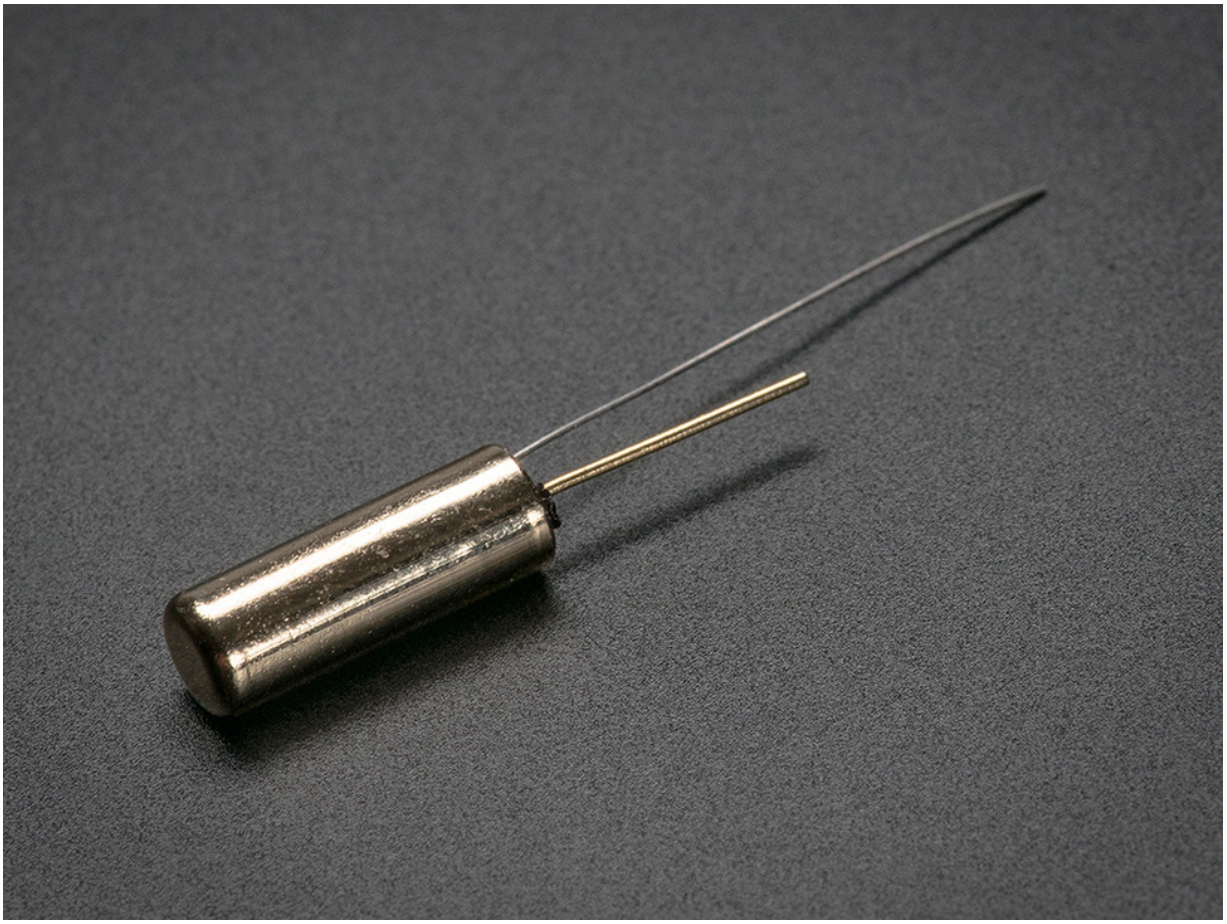
Vous pouvez l'utiliser pour vérifier si un robot touche un obstacle, si un portail est en fin de course, si un contact est réalisé entre deux objets mobiles...

Il est possible de réaliser des capteurs de contact juste avec quelques fils conducteurs. L'important est que le capteur doit avoir une position au repos stable et qu'après un contact, il reprenne cette position. On peut imaginer un capteur de type NC réalisé avec une pince à linge en bois, des punaises et quelques fils de liaison  .

Le tiltsensor

On dit que c'est l'accéléromètre du pauvre. Le principe est simple : une bille en métal vient faire contact sur des pièces métalliques si le mouvement le permet. Ce mouvement doit être brusque (genre le fameux tilt au flipper), ou un changement d'inclinaison. Le tiltsensor fait donc partie des capteurs dits tout-ou-rien.

Voici la tête qu'il a (il en existe de plusieurs formes, comme ici un tube, mais le principe reste le même) :



Exemple de tiltsensor (www.adafruit.com)
Alors pour la programmation, c'est un copié-collé du programme lié au capteur de contact. Pour les connexions, c'est la même chose.

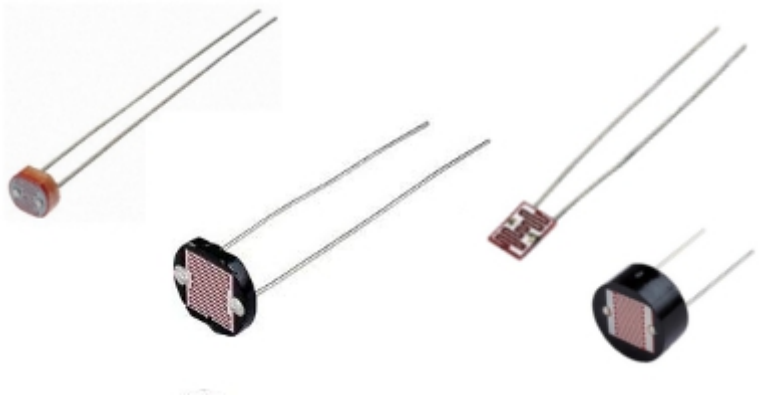
Quand utiliser un tiltsensor ?

Grâce au tiltsensor, on peut percevoir un mouvement brusque (chute ou choc) car on capte alors une absence de contact. C'est ce que l'on trouve dans les flippers. On peut aussi lire grossièrement une inclinaison avec deux tiltsensors positionnés en un V très ouvert, l'inclinaison excessive d'un côté ou de l'autre coupera le contact de l'un des deux.

Nous avons travaillé un peu avec le mouvement, voyons maintenant la lumière...

La photorésistance

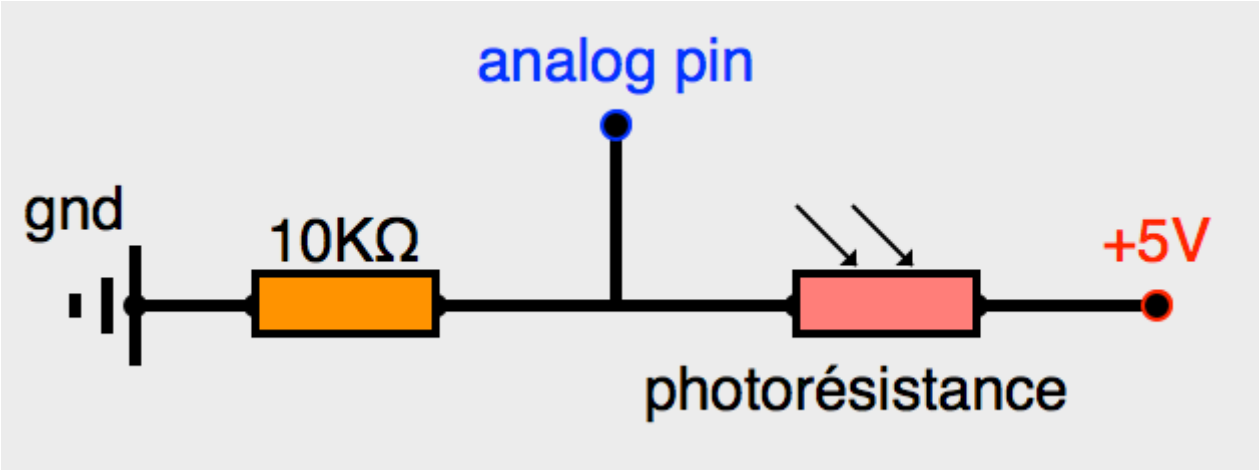
La photorésistance est une forme de capteur de lumière. C'est un composant très simple à mettre en œuvre et qui permet une interaction intéressante avec l'environnement. Voici à quoi elle ressemble :



Quelques photorésistances (robotastuces.free.fr)
Le principe est assez simple : plus il y a de lumière, plus la résistance est basse. L'obscurité provoque une résistance importante.

Il s'agit donc d'un capteur de variation qu'il faudra connecter à l'Arduino avec un pin analogique. Rappelez-vous le pin analogique transforme une tension reçue entre OV et 5V reçue en valeur entre 0 et 1 024 (rafraîchissez-vous la mémoire dans le chapitre sur le potentiomètre si besoin).

Voici le schéma de montage à réaliser pour que votre carte Arduino puisse lire correctement votre photorésistance :



Connexion d'une photo résistance à un pin analogique
Voici le petit programme qui va avec. Plus la photorésistance reçoit de la lumière, plus la résistance est faible, donc plus la valeur affichée par l'Arduino est élevée.

```
int pinPR=A0; //pin de connexion pour la photorésistance

void setup() {

  Serial.begin(9600);

}
```

```
void loop() {

  int valeur=analogRead(pinPR); // on lit la valeur transmise par la photorésistance

  Serial.println(valeur); // on l'affiche

}
```

Il peut être intéressant, sans calcul de tension liés au montage, de récupérer les valeurs extrêmes lues par l'Arduino. Voici un autre programme qui affiche ces valeurs et permet d'ajuster le pourcentage entre 0% et 100%. Je vous laisse l'observer, il est intéressant pour étalonner vos mesures dans un lieu donné.

```
int pinPR=A0; //pin de connexion pour la photorésistance

int valMin=1024; // on initialise la valeur minimale au plus haut

int valMax=0; // et la valeur maximale au plus bas

void setup() {

  Serial.begin(9600);

}



void loop() {

  int valeur=analogRead(pinPR); // on lit la valeur transmise par la photorésistance

  if (valeur>valMax) //on compare avec valMax

    valMax=valeur;  // on modifie valMax

  if (valeur<valMin) // on compare avec valMin

    valMin=valeur;  // on modifie valMin

  int pourcentage=map(valeur,valMin,valMax,0,100); //pourcentage entre les bornes

  //Séquence d'affichage

  Serial.print("Valeur : ");

  Serial.print(valMin);

  Serial.print(" < ");
```

```
  Serial.print(valeur);

  Serial.print(" < ");

  Serial.print(valMax);

  Serial.print("  soit : ");

  Serial.print(pourcentage);

  Serial.println(" %");

}
```

Orientez la photorésistance vers différents endroits de la pièce plus ou moins lumineux, approchez votre main... Vous obtenez les limites des mesures et le pourcentage de luminosité lu entre ces bornes.

Quand utiliser une photorésistance ?

On peut grâce à ce capteur, déclencher un évènement en fonction de la luminosité : pièce qui s'éclaire lorsqu'on entre, inclinaison des stores pour gérer l'entrée de lumière, guidage d'un robot grâce à une source de lumière, allumage d'une lampe si trop d'obscurité...

C'est notre premier capteur sans contact, ça se fête ! (Lukas, ça se fête aussi sans contact, merci...)

Si la zone où est placée votre photorésistance est trop éclairée, il faut réduire la résistance de 10KΩ à 1KΩ afin d'obtenir des mesures plus fines (mais moins précises dans l'ombre du coup). Une bonne méthode est d'utiliser un potentiomètre à la place de cette résistance, ce qui permet d'affiner les réglages.
Bon, vous avez vu qu'il n'y a rien de bien compliqué dans ce montage et dans ce programme. À vous ensuite de l'adapter à vos besoins.

Ne connectez surtout pas de matériel fonctionnant en 220V avec l'Arduino ! C'est dangereux ! Dans le cours de perfectionnement, vous verrez comment réaliser des montages domestiques sécurisés.
Continuons avec la lumière, mais celle qu'on ne voit pas !

La photodiode infrarouge

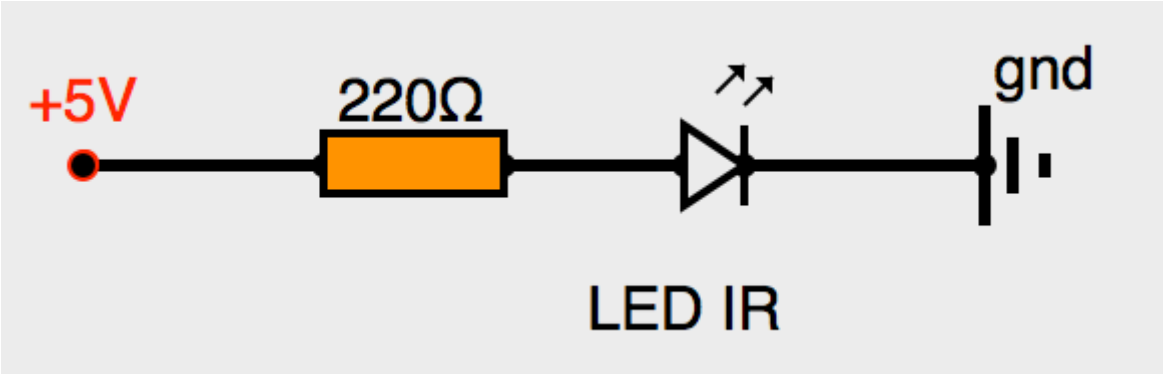
Alors pour faire simple, l'être humain de base (pas les super-héros) ne voit pas tout de la lumière. En fait, la lumière est une onde électromagnétique. Le spectre (c'est son nom) de la lumière visible, donc que notre œil peut voir, va du violet (390 nm) au rouge (780nm). L'unité "nm" veut dire nanomètre, et mesure une longueur d'onde. Le spectre électromagnétique complet comprend dans l'ordre : les rayons gamma, les rayons X, les ultra-violets, la lumière visible, les infrarouges, les micro-ondes, les ondes radio.

La lumière infrarouge est donc invisible pour l'oeil humain. En revanche, nous savons l'émettre et la capter. Elle est utilisée dans vos télécommandes par exemple pour envoyer un message à votre télé ou votre chaîne hi-fi (nous verrons d'ailleurs dans le cours de perfectionnement comment faire).

La photodiode infrarouge va réagir à la lumière infrarouge en produisant un signal électrique que l'Arduino va pouvoir récupérer.

Avant d'utiliser une photodiode, il nous faut produire de l'infrarouge. Et bien rien de plus simple ! Il suffit d'utiliser une LED infrarouge (ben oui, ça existe !). Ça ressemble à ça :

Une LED infrarouge
Il y en a des blanches et des grises.Comme toutes les LED, la LED infrarouge doit être connectée dans le bon sens et comme toutes les LED, il faut la protéger avec une résistance.
Petit rappel du schéma :



Connexion d'une LED infrarouge avec résistance de protection.
Bien, si vous avez le matériel pour (longueur de fil, mini bread-board) je vous conseille de la réaliser sur un socle mobile (pour pouvoir l'éloigner et l'approcher de la photodiode), car vous pourrez ainsi tester plus facilement les valeurs perçues par la photodiode (qui reçoit la lumière infrarouge).

Alors maintenant c'est quoi une photodiode, et bien voici à quoi elle ressemble :

Une photodiode

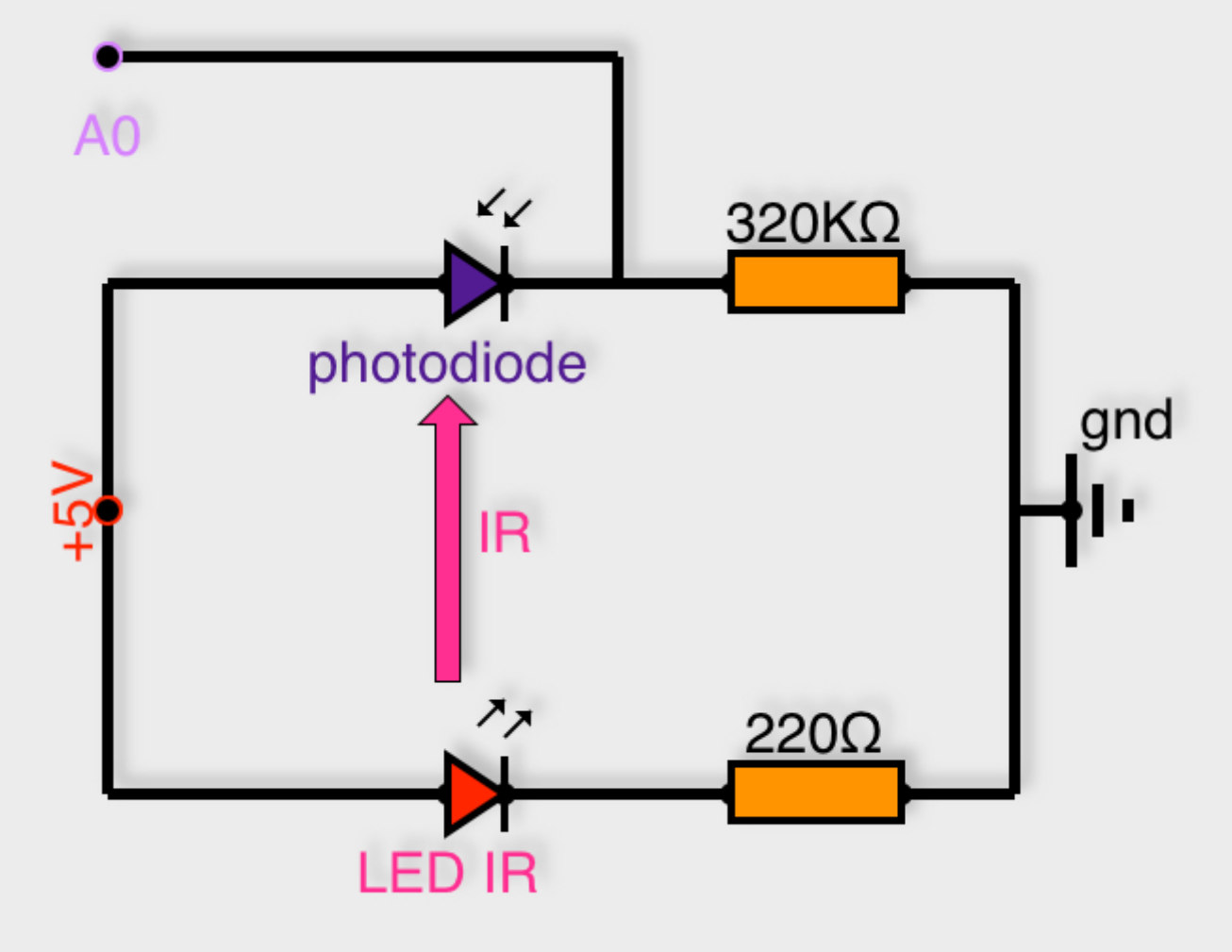
Heu... ça ressemble à une LED non ?

Si ! Et en plus il en existe des blanches et des grises (comme la LED IR) ! Alors je vous conseille de bien noter où vous rangez les unes et les autres, car elles ne fonctionnent pas pareil du tout ! Vous trouverez aussi des photodiodes rectangulaires avec un angle coupé qui représente la borne -.

Les photodiodes sont souvent avec une coque de plastique noir. Cela permet de filtrer la lumière visible et de ne capter que la lumière infrarouge. Il est possible de différencier une photodiode d'une LED infrarouge (si elles sont transparentes toutes les deux) en les regardant de face : on voit un carré noir au fond de la photodiode qui n'apparaît pas dans la LED. Mais ça reste assez peu fiable. Le mieux c'est de bien ranger.
Voyons comment on peut connecter notre photodiode à l'Arduino...

Montage dans le sens "normal"

L'idée est de respecter le sens de connexion de la photodiode, c'est-à-dire la patte courte vers le ground et la patte longue vers le +5V. La lecture se fait du côté de la patte courte. Pour que ce montage permette une lecture efficace du passage de courant dans la diode, il faut ajouter une résistance importante vers le ground (ici 320KΩ). La LED IR est elle connectée comme une LED normale et son faisceau dirigé vers la photodiode.



Montage d'une photodiode en mode normal.
Et voici un programme on ne peut plus simple pour lire avec l'Arduino la valeur de la lumière infrarouge perçue par la photodiode :

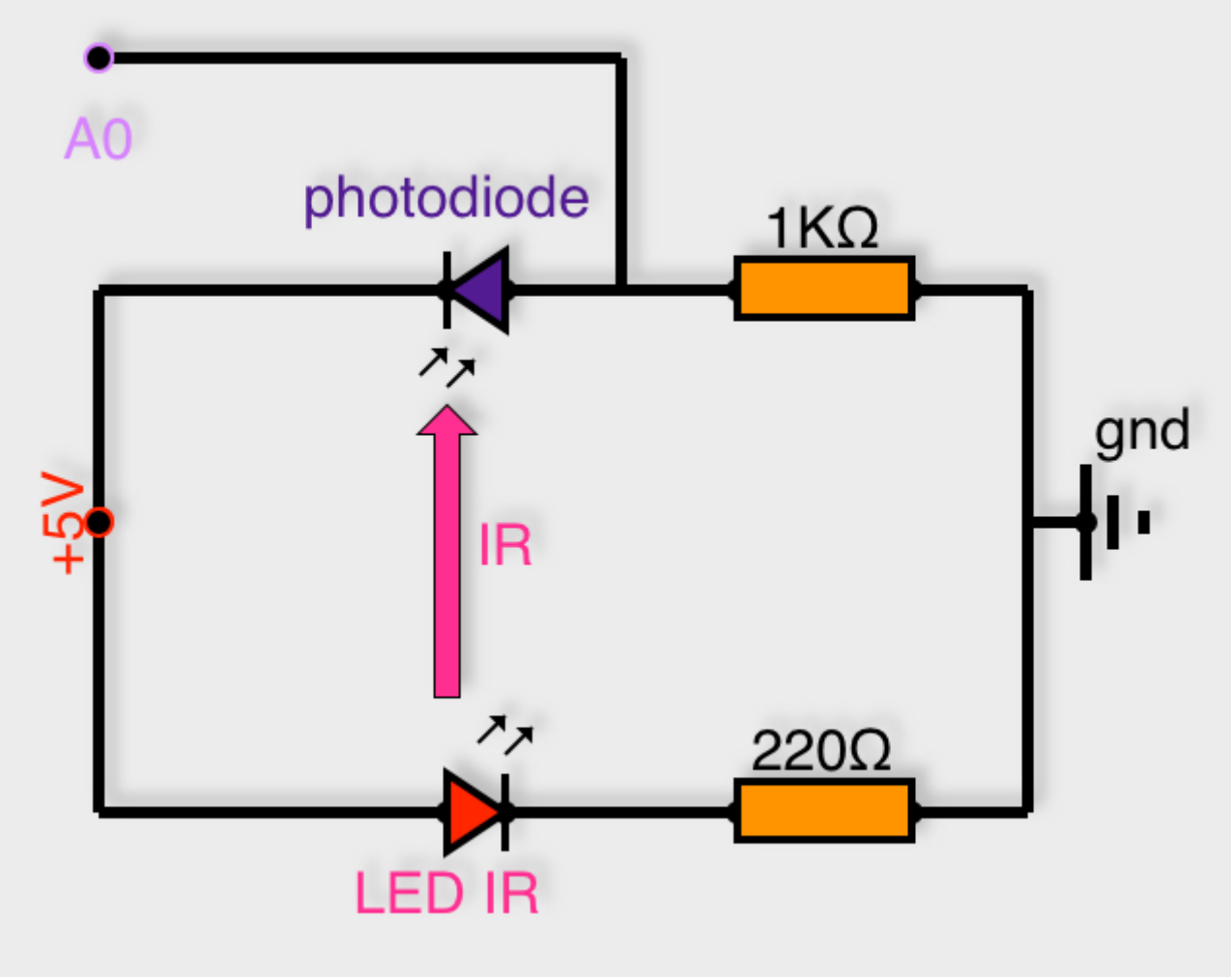
```
void setup() {

  Serial.begin(9600);

}

void loop() {

  Serial.println(analogRead(A0)); //lecture du CAN A0 connecté à la patte - de la photodiode

}
```

En passant un objet opaque (votre main par exemple) entre la LED IR et la photodiode, vous verrez le résultat varier entre 30 et 800 environ. Ce genre d'information est donc complètement exploitable ensuite dans un programme.

Montage "à l'envers"

Il s'agit ici d'utiliser le fait que la photodiode émet un courant lorsqu'elle reçoit un faisceau d'ondes infrarouges. Nous allons donc monter la diode dans le mauvais sens : la petite patte vers le +5V et la grande vers le ground. La résistance pour ce montage doit être plus faible (ici 1KΩ).

Ces deux façons de monter une photodiode sont efficaces l'une comme l'autre. C'est donc bon de connaître les deux. Je n'ai pas encore trouvé un argument choc qui justifie d'utiliser un montage plutôt que l'autre...
De mon côté j'utilise l'une comme l'autre en fonction du résultat le plus efficace (et ça varie d'une diode à l'autre !)
Le programme est le même.



Montage d'une photodiode en mode inversé
De même, vous devriez trouver une plage de lecture entre 30 et 880 en lançant le même programme que pour le montage dans le sens “normal”.

En fait la longueur d'onde d'une diode est fixe. En revanche ce qui varie c'est la tension (résistance plus ou moins importante) qui montre la quantité d'infrarouge captée (en valeur lue par le CAN dans notre exemple entre 30 et 880). Elle est directement liée à la distance entre la diode et la photodiode. C'est donc une donnée qui indique une notion de distance. mais elle n'est pas transformable en cm car trop variable en fonction du montage. Vous trouverez des explications plus complètes dans le cours de perfectionnement.
Quand utiliser une photodiode infrarouge ?

La première application qui vient à l'esprit est la barrière infrarouge. En effet vous pouvez très bien associer un montage d'alarme (nous verrons les sons dans mon cours de perfectionnement Arduino) qui réagit lorsque la valeur lue est inférieure à un seuil. Vous pouvez décider de diriger votre robot mobile vers une source infrarouge située dans la pièce. Plus la réception du faisceau est forte, plus la direction de votre robot est bonne.

Un montage de la LED IR et de la photodiode côte-à-côte et dirigées dans le même sens permet de réaliser un petit détecteur de distance (la lumière IR envoyée par la diode se reflète sur l'obstacle et est perçue de façon plus ou moins importante par la photodiode : plus la lumière est perceptible, plus l'obstacle est proche).

Enfin ce même montage peut permettre de faire déplacer un robot en suivant une ligne noire. Du fait que la couleur noire absorbe les radiations et que le blanc les rejette (oui c'est simplifié, mais bon), la lecture montrera facilement si la LED émet sur du noir ou du blanc (sortie ou non de la ligne) et permettra de gérer le robot en fonction.

Le récepteur infrarouge

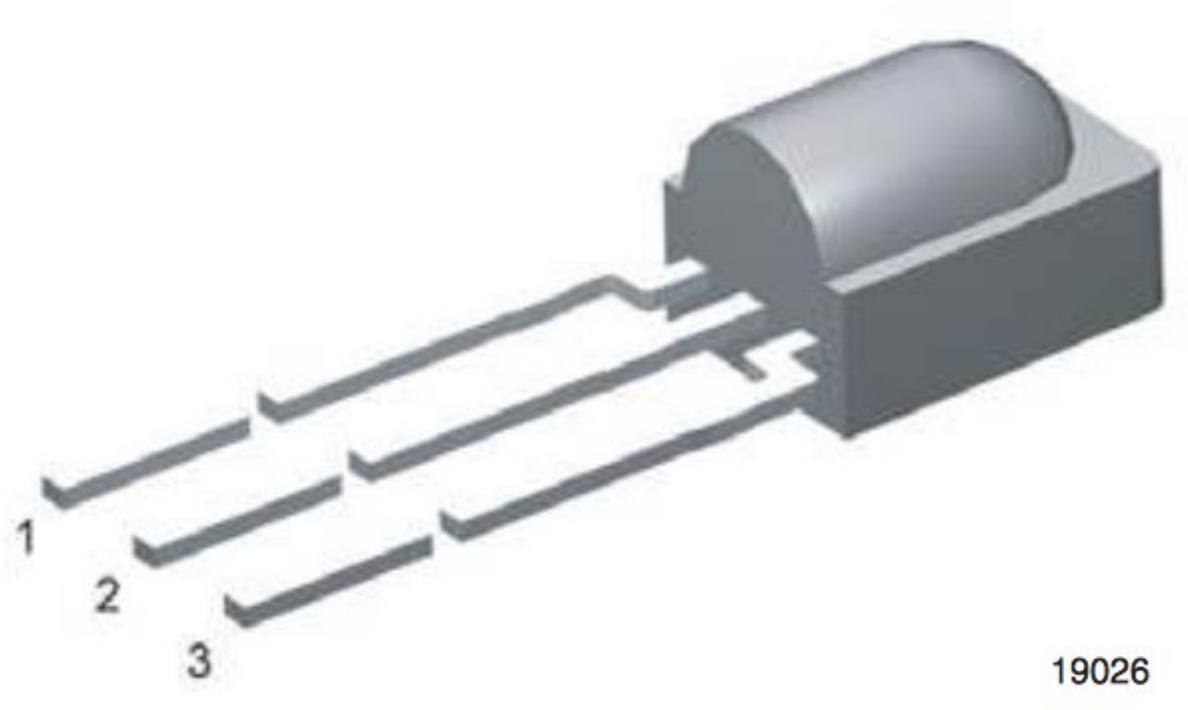
La différence entre la photodiode et le récepteur infrarouge est importante. En effet, non seulement ce dernier est de type tout-ou-rien, mais en plus il ne fonctionne que s’il reçoit la lumière infrarouge sous forme de signal (état haut, état bas) cadencé à 38KHz environ (soit toutes les 26 millisecondes environ). Il sert surtout de capteur de télécommande infrarouge (ou de tout ce qui émet un code infrarouge à la bonne fréquence, donc un autre Arduino par exemple).

Voici un exemple de récepteur infrarouge : le TSOP38238.

Récepteur infrarouge : le TSOP38238
Et voici un lien vers la datasheet du TSOP38238.

Il fait l'objet d'un chapitre complet dans le cours de perfectionnement avec des exemples de lectures de codes de télécommande et de simulation d'une télécommande par l'Arduino.

Voici tout de même une explication sur la façon dont on le connecte :



Les pattes du TSOP38238
La patte 1 (OUT) se connecte sur un pin numérique de l'Arduino (en effet c'est un capteur tout-ou-rien), la patte 2 sur le ground et la patte 3 sur le 5V. Voici un mini-programme qui montre les réactions de TSOP38238 :

```
void setup() {

  Serial.begin(9600);

}

void loop() {

  Serial.println(digitalRead(7));

}
```

Ce programme ne donnera pas de réaction si vous mettez une LED IR devant le TSOP et que vous passez la main entre les deux. Ce capteur ne peut pas servir de capteur pour barrière IR. En revanche, si vous dirigez une télécommande infrarouge (celle de votre télévision, de votre chaîne hifi ou de votre lit à bulles) vers le capteur et que vous appuyez sur l'un des boutons, vous verrez des 0 et des 1 se succéder sur la console. Le capteur reçoit bien le code.

Voici un autre code qui vous montre que les valeurs ne sont pas liées au hasard. Il permet d'afficher des "temps fictifs" d'états haut et bas en fonction du signal reçu par le récepteur infrarouge.

```
void setup() {

  Serial.begin(9600);

  pinMode(7, INPUT);

}

void loop() {

  int p = 0; //variable de comptage pour tableau

  int tLow,tHigh; //variables de stockage de temps fictif

  int duree[32][2]; //tableau de stockage des temps de 32 sur 2

  Serial.println("Go"); //lancement de la procédure

  while (digitalRead(7)) {} // attente de changement d'état



  while (p < 32) { // boucle de stockage des données

    tLow=0; //initialisation du temps fictif pour état BAS

    tHigh=0;//initialisation du temps fictif pour état HAUT

    while (!digitalRead(7)) { //tant que le récepteur reçoit un signal LOW

      tLow++; //on incrémente le temps fictif pour état bas

    }

    while (digitalRead(7)) { //tant que le récepteur reçoit un signal HIGH

      tHigh++; //on incrémente le temps fictif pour état haut

    }

    duree[p][0] = tLow; //stockage dans tableau

    duree[p][1] = tHigh; //stockage dans tableau

    p++;

  }
```

```
  //affichage du tableau
  Serial.print("LOW\t");
  for (int p = 0; p < 32; p++) {
    Serial.print(duree[p][0]);
    Serial.print("\t");
  }
  Serial.println();
  Serial.print("HIGH\t");
  for (int p = 0; p < 32; p++) {
    Serial.print(duree[p][1]);
    Serial.print("\t");
  }
  Serial.println();


  delay(3000);// attente pour nouvelle procédure
}
```

Ce code ne donne qu'une une approximation des temps où le signal est aux états haut et bas et ne peut pas servir des données utilisables. En revanche il montre la procédure utilisée pour décrypter un code. Pour avoir des temps plus exacts, il faut utiliser les registres de l'Arduino, c'est à dire lui parler directement sans passer par des fonctions. En effet, la fonction `digitalRead()` est bien trop lente pour lire correctement le pin 7. Nous y reviendrons dans dans le cours de perfectionnement.
À noter tout de même : l'utilisation de la boucle

```
while(condition){
    //code à exécuter
}
```

qui permet de réaliser un bloc de code tant que la condition est vérifiée et qui sort de la boucle lorsqu'elle ne l'est plus. C'est une alternative au

```
for (variable;condition;incrément){

    //code à exécuter

}
```
qui permet de réaliser une boucle lorsqu'on connaît la valeur finale.

Avec l'utilisation de `while`, il est important de vérifier que la boucle sera quittée à un moment ou un autre, sinon vous serez en boucle infinie... dans les limbes numériques de l'Arduino.

Quand utiliser un récepteur infrarouge ?

Ce récepteur fait souvent partie des kits que l'on trouve dans le commerce, c'est pour ça que je vous en parle ici. En revanche, son utilisation (envoi de code IR ou réception de code IR) n'est pas accessible au débutant (à moins de copier bêtement du code). Je donnerai des exemples d'utilisation et de programmes dans le cours de perfectionnement, vous pourrez y faire un tour si ça vous intéresse !

Capteur de température

Il existe de multiples capteurs de température. Je vous propose ici de voir comment utiliser le TMP36, qui est un capteur qui permet de mesurer des températures entre -50°C et 125°C, ce qui devrait correspondre à la majorité des climats terrestres et aux données que vos robots pourront récolter dans un environnement peu hostile.

Alors comme d'habitude, une petite photo pour qu'on ait une idée de sa tête...

TMP36

Et là vous me direz encore, mais ça ressemble beaucoup à un transistor !!! Et bien oui : trois pattes, un côté plat, un côté rond... Mais il y a des inscriptions dessus qui permettent de distinguer son nom ! (Oui Lukas, un peu comme votre gourmette !), du coup, on peut aller voir la datasheet qui va bien ! Mais ce n'est pas très lisible pour les connexions, alors je vous aide. Prenez le composant les pattes en bas et le côté plat face à vous :

- patte de gauche : connectée au +5V ;
- patte de droite : connectée au ground ;
- patte du centre : connectée à un CAN de l'Arduino.

Ne vous trompez pas dans les connexions de cette puce, car elle peut se mettre à chauffer et ça brûle les doigts si on veux l'enlever, ce qu'il faut faire, sinon elle grille .

Bien, une fois connectée, il faut pouvoir s'en servir ! Plus bas dans la datasheet, on voit ce graphique :

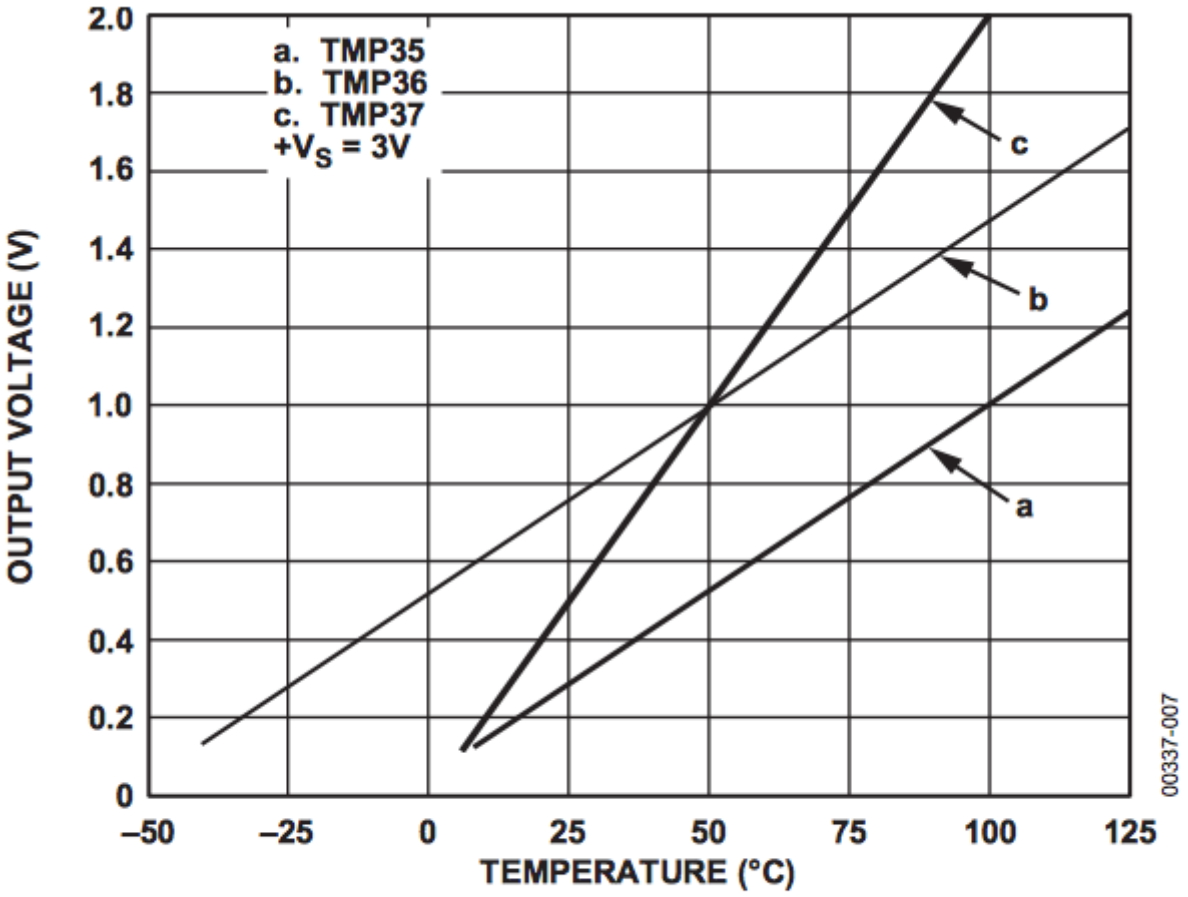


Figure 6. Output Voltage vs. Temperature

courbe de relation entre température et tension
Ce graphique présente le rapport entre la tension lue à la sortie (patte du centre) et la température. Je vous rappelle qu'on parle bien de tension car les convertisseur analogique-numériques (CAN) de l'Arduino convertissent bien une tension lue entre 0V et 5V en nombre entre 0 et 1023.

On lit sur le graphique que le TMP36 correspond à la droite b qui va de 0°C (pour 0V) à 125°C (pour 1,75V). Il est aussi indiqué que la tension d'entrée Vs (S pour "supply") vaut 3V, mais sachez que le TMP36 fonctionne aussi à 5V (pour les mêmes valeurs).

Alors comment faire ? Tout simplement en mappant correctement les valeurs. Voici le programme qui affiche la température lue par le TMP36 :

```
void setup() {

  Serial.begin(9600);

}
```

```
void loop() {

  int valeur=analogRead(A0);//on lit la valeur au CAN 0

  int t=map(valeur,0,1023,0,5000);//on transforme la valeur lue en valeur de tension entre
0 et 5000 mV (car pas de virgule)

  int tmp=map(t,0,1750,-50,125); //on transformela tension (de 0 à 1750mV en température
(de -50°C à 125°C);

  Serial.println(tmp); //on affiche le résultat

}
```

Et bien voilà ! Vous avez un thermomètre ! (Lukas vous l'utiliserez pour ce que vous voudrez, je ne veux pas le savoir)

Quand utiliser un capteur de température ?

Et bien tout ce qui concerne la température : station météo, déclenchement de chauffage (attention, les connexions doivent respecter les normes de sécurité EDF !!!), analyse de variation de température d'un liquide (biberon, eau des nouilles, avec bien sûr un bricolage sans risque ). De quoi bien vous occuper !

Le capteur à ultrasons

De même que nous ne percevons pas toutes les fréquences de la lumière, nous ne percevons pas toutes les fréquences du son. Le spectre audible de fréquences (que l'humain peut percevoir) s'étend conventionnellement entre 20Hz et 20KHz. Comme pour la lumière, cela peut varier d'un individu à l'autre.

En revanche, le son ne se déplace pas à la même vitesse que la lumière :

- Vitesse de la **lumière** : **299 792 458 m/s** (mètre par seconde) **dans le vide** (c'est à dire sans matière) ;
- Vitesse du son : Le son se propage obligatoirement dans la matière (donc pas de son dans le vide) et sa vitesse dépend de la matière en question. La vitesse du **son dans l'air** est d'environ **340 m/s** (mètre par seconde).

Le capteur à ultrason simple permet, comme un micro, de capter des sons. Sauf qu'il ne capte que les ultrasons, c'est-à-dire les sons situés au-delà des 20KHz (précisément de 20 KHz à 10 MHz).

Le SRF05 contient à la fois un capteur et un émetteur à ultrasons.

Voici une petite photo de présentation :

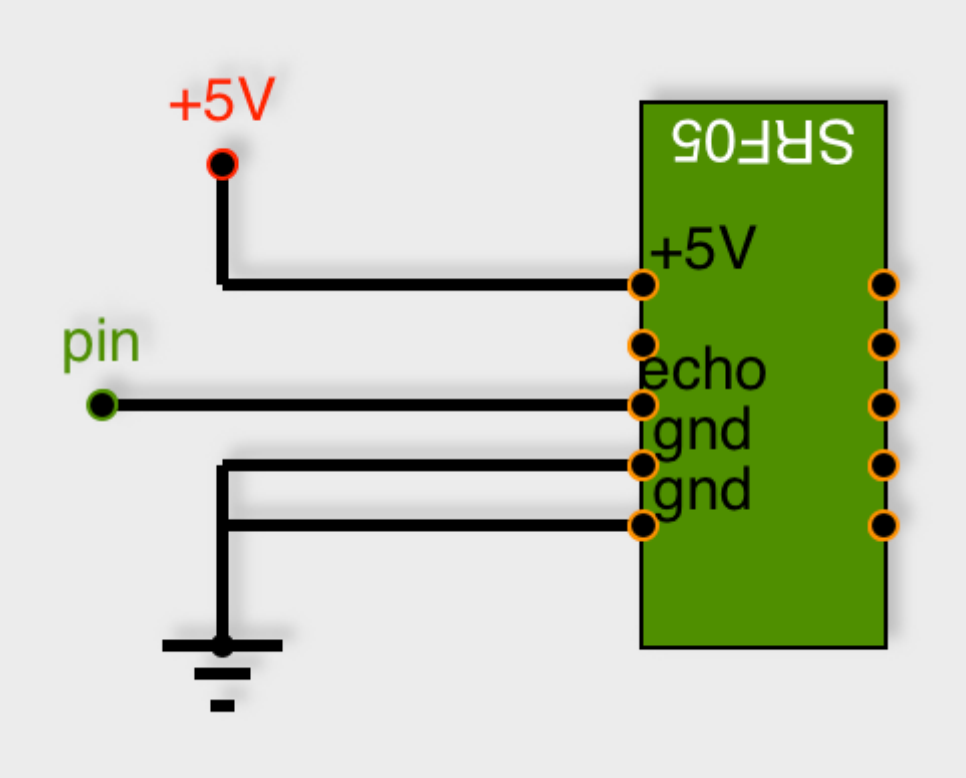
SRF05 : émetteur et récepteur à ultrasons
Vous l'aurez compris, il s'agit d'un module tout prêt qui contient de l'électronique (avec un microcontroleur PIC16F630) qui va communiquer avec l'Arduino.

Voici un lien vers la Notice du SRF05 afin de voir comment connecter correctement le SRF05 et comprendre son fonctionnement.

On y lit plusieurs choses :

- Le SRF05 mesure des échos entre 3cm et 4m de distance.
- La mesure ne peut se faire que toutes les 50ms.
- Le meilleur cône d'écoute de l'écho est de 30° (soit -15° et +15° en face du capteur) mais il peut capter correctement jusqu'à un cône de 60°. Non Lukas, le cône d'écoute n'est pas l'appareil que votre grand oncle se met dans l'oreille pour mieux entendre ! Il s'agit de la zone en face du capteur qui peut recevoir l'écho...
- Il peut être connecté de deux façons différentes : soit avec une patte pour l'émission et une patte pour la réception, soit avec une seule patte pour l'émission et la réception.

Nous utiliserons dans l'exemple la connexion avec une seule patte (émission/réception). Voici le schéma :



Connexion du SRF05 avec une patte pour envoi et réception (attention à l'orientation)
Notez bien que dans ce cas, deux pattes seront connectées au ground.

Comment le capteur à ultrasons mesure-t-il la distance des échos de sons ?

Le principe est assez simple :

1. L'Arduino demande (par programmation) au SRF05 de procéder à la mesure. Cette demande se fait en envoyant un signal haut (HIGH) pendant 10 microsondes au SRF05. Puis on attend le retour en comptant le temps qui passe (en microsecondes).
2. Le SRF05 envoie un "beep" ultrasons (à la fréquence de 40KHz) et attend l'écho. C'est-à-dire qu'il attend que le son rebondisse sur quelque obstacle et lui revienne.
3. Dès que le son est capté en retour, il en informe l'Arduino en lui renvoyant un signal HAUT.
4. L'Arduino(par programmation) exploite ensuite les données du temps entre envoi et réception.

Mais comment peut-on savoir comment transformer du temps en distance ?

Il suffit de faire un calcul mathématique... 

Le temps parcours 340 mètres en 1 seconde, soit 34000 cm en 1 seconde, soit 34000 cm en 1000000 microsecondes.

Vous suivez ? Ce ne sont que des conversions, attendez la suite...

Avec une règle de trois on cherche le temps mis pour parcourir 1 cm :

temps=1000000*1/34000=29 et des poussières. Donc le son met environ 29µs (microsecondes) pour parcourir 1cm.

Le capteur envoie le son qui rencontre un obstacle et qui rebondit. Donc il va parcourir la même distance à l'aller qu'au retour. Si l'objet est à 1 cm, le son va donc mettre 29µs pour aller jusqu'à l'obstacle, puis 29µs pour revenir, soit 58µs.

Pour obtenir la distance par rapport au temps mesuré entre "beep" et "écho", il suffit donc de diviser le temps en microseconde par 58 et on obtient la bonne distance !

(Bon, je double la moyenne de celui ou celle qui arrive à faire comprendre ça à Lukas en moins d'une semaine, parce que je n'ai pas l'impression que mon "beep" a fait écho chez lui...)

Voici maintenant le code commenté pour mesurer une distance avec le capteur ultrasons SRF05. Vous verrez que je ne me contente pas d'une mesure, mais d'une moyenne sur 10 mesures. Cela ralentit la fréquence des mesures mais en augmente la fiabilité.

```
// SRF05 connecté en mode un seul pin (émission/réception)



int pinSRF = 7; //pin digital pour l'envoi et la réception des signaux

int vSon=59; //valeur de temps en µs d'un aller retour du son sur 1cm

void setup() {

  Serial.begin(9600); //on initialise la communication série

}

//boucle principale

void loop() {

  int distance=mesureDistance(); //on récupère la valeur de distance grâce à la fonction
créee plus bas

  Serial.println(distance); // on affiche la distance en cm

}



//fonction de mesure de distance avec SRF05

int mesureDistance() {

  unsigned long mesure = 0; // variable de mesure
```

```
  unsigned long cumul = 0; //variable pour la moyenne

  for (int t = 0; t < 10; t++) { // boucle pour effectuer 10 mesures

    pinMode (pinSRF, OUTPUT); //on prépare le pin pour envoyer le signal

    digitalWrite(pinSRF, LOW); //on commence à l'état bas

    delayMicroseconds(2); //on attend que le signal soit clair

    digitalWrite(pinSRF, HIGH);//mise à l'état haut

    delayMicroseconds(10); //pendant 10 µs

    digitalWrite(pinSRF, LOW); //mise à l'état bas

    pinMode(pinSRF, INPUT); //on prépare le pin pour recevoir un état

    mesure = pulseIn(pinSRF, HIGH); // fonction pulseIn qui attend un état haut et renvoie
le temps d'attente

    cumul+=mesure; //on cumule les mesures

    delay(50); //attente obligatoire entre deux mesures

  }

  mesure=cumul/10; //on calcule la moyenne des mesures

  mesure=mesure/vSon;//on transforme en cm

  return mesure; //on renvoie la mesure au programme principal

}
```

Le type `unsigned long` permet de stocker de entiers entre 0 et 4 294 967 295. C'est plus sûr lorsque vous utilisez des nombres qui peuvent grandir rapidement, sinon vos données seront fausses (dépassement de mémoire).
Le dépassement de mémoire ne cause rien de grave en soi. La valeur qui dépasse une limite, modifie le nombre en négatif (pour un type signé) ou revient à 0 (pour un type non signé). En revanche, cela fausse complètement les valeurs. Essayez d'ailleurs de mettre un type int à la place de unsigned long, pour voir.
La fonction `pulseIn(pin,ETAT)` est une fonction bien pratique que propose le langage Arduino. Elle compte le temps en microseconde entre le moment où on appelle cette fonction et le moment où le pin passe à l'état indiqué. On peut appeler cette fonction avec l'état d'attente d'un signal haut (HIGH) ou bas (LOW). Dans le programme nous attendons un signal haut.

La fonction `delayMicroseconds()` permet d'attendre non pas en millisecondes mais en microsecondes.

Quand utiliser un capteur à ultrasons ?

L'application la plus utilisée est la détection d'obstacle à distance par des robots mobiles. On peut adapter la vitesse du robot par rapport à la détection, l'arrêter si l'objet est trop proche, et lui faire chercher un direction "libre" avant de le faire repartir.

On peut très bien imaginer un appareil de mesure de volume (avec un stockage des données de distance sur trois dimensions). Ou un stabilisateur d'altitude (dans une mesure de 4m maximum) pour un objet volant.

Je vais m'arrêter ici pour les capteurs, non pas parce qu'on a fait le tour, mais juste parce que nous avons vu différentes façons de recevoir des données d'un capteur (tout-ou-rien, valeur de tension et signaux). Dans la grande majorité des cas, vous trouverez les informations nécessaires à l'utilisation d'un capteur sur les forums et sur les notices (datasheet ou notice du fabriquant).