

Partie 3

Donnez du mouvement à vos montages avec un servo-moteur (... et la fonction Switch)

Je dois avouer que nous abordons maintenant une partie des possibilités qu'offre l'Arduino qui me fait toujours l'effet de mon premier cadeau de Noël motorisé (livré avec les piles bien sûr...), car en effet, nous allons donner du mouvement à nos montages !

Dans ce chapitre vous allez découvrir comment utiliser un servo-moteur et le programmer avec l'Arduino.

J'en profiterai pour vous apporter un nouveau mot de vocabulaire pour effectuer des conditions : le switch.

En route pour de nouveaux montages !

Le servo-moteur

Le nom lui-même est intrigant. Notez bien qu'il ne s'agit pas de cerveau (oui Lukas, ce qui se trouve entre vos deux oreilles, enfin, j'espère...) mais bien de servo. Ce mot vient du latin "servus", qui signifiait esclave. Un servo moteur est donc un moteur esclave... mais de quoi ?

Tout d'abord, une petite image de présentation :



Quelques servo-moteurs (source : francrobotique.com)

Vous remarquez un bloc, un axe et une roue trouée (ou une sorte de barre trouée aussi) sur l'axe.

Des servo-moteurs, il en existe de plusieurs tailles : des plus petits de quelques grammes, tout en plastique, au plus gros de plusieurs kilogrammes, en métal. L'énergie qui les pilote et les fait n'est pas toujours électrique, il existe des servo-moteurs hydrauliques.

Dans notre cas, je parlerai de servo-moteurs électriques de petite taille comme ceux que l'on utilise souvent en modélisme.

C'est bien beau tout ça, mais c'est quoi alors un servo-moteur ?

Alors avant d'aborder le côté servo, voyons d'abord le côté moteur...

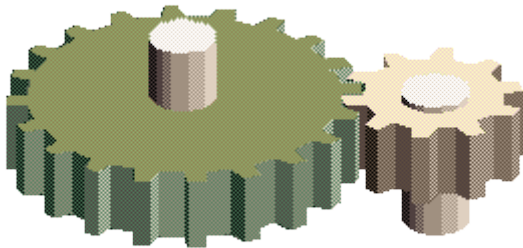
Un servo-moteur contient (dans le cas de ceux que vous utiliserez avec votre Arduino) un moteur à courant continu. Nous aborderons plus précisément ce que c'est dans le chapitre suivant, mais pour résumer, c'est un mécanisme qui tourne lorsqu'il est parcouru par l'électricité.

Le moteur du servo-moteur en revanche, n'est pas un simple moteur, il est associé à une série d'engrenages qui va lui permettre de gagner en puissance. Mais comme rien n'est gratuit, ce gain en puissance réduit sa vitesse de rotation.

Aparté sur les engrenages

Un engrenage est une roue dentée qui en tournant entraîne (engrène) une autre roue, dentée aussi. Ça a l'air évident mais deux principes fondamentaux s'ajoutent pour bien maîtriser la notion :

1. Si la première roue dentée tourne dans un sens, la seconde tournera dans l'autre sens. On peut donc dire que le sens de rotation s'inverse d'une roue à l'autre dans un engrenage.
2. Si les deux roues dentées ne font pas la même taille, la vitesse de rotation de chaque roue sera différente. En effet si la première roue a 6 dents, et la seconde 24 dents, la première effectuera 4 tours alors que la seconde n'en fera qu'un ($6 \text{ dents} \times 4 \text{ tours} = 24 \text{ dents}$). N'oublions pas qu'une roue entraîne l'autre dent à dent.



Un engrenage de deux roues : 10 et 20 dents. (source : futura-sciences.com)

Dans l'image ci-dessus, la grande roue effectue 1 tour pendant que la petite en effectue 2.



La conséquence de la différence de taille entre deux roues (la petite est appelée pignon) est que la transmission du mouvement gagne en puissance (couple) de la petite vers la grande roue. Je ne vais pas entrer ici dans des explications physiques mais retenez que :

- Si vous voulez accélérer le mouvement, le moteur doit entraîner la grande roue qui entraînera la petite. Dans ce cas la puissance de rotation (le couple) de l'axe de la petite roue sera plus faible.
- Si vous souhaitez gagner en couple, vous devrez entraîner la petite roue par le moteur, qui entraînera la plus grande. Dans ce cas, vous perdrez de la vitesse.

Je donne ces informations car elles servent au-delà de l'explication du servo-moteur. En effet, si vous fabriquez des robots mobiles, vous aurez à vous poser la question entre vitesse et puissance. Par exemple, l'hélice d'un hélicoptère doit tourner très rapidement, mais les roues d'un tracteur doivent fournir un couple important...

Dernier point, lorsque plusieurs roues dentées s'entraînent à la suite les unes des autres, on parle de train d'engrenages.

Revenons à nos servo-moteurs.

Le moteur du servo entraîne un train d'engrenage qui lui permet de gagner en puissance de rotation. L'axe qui sort du servo-moteur (celui qu'on utilise) tourne donc bien moins vite que celui du moteur qui est à l'intérieur, en revanche son couple est plus important.



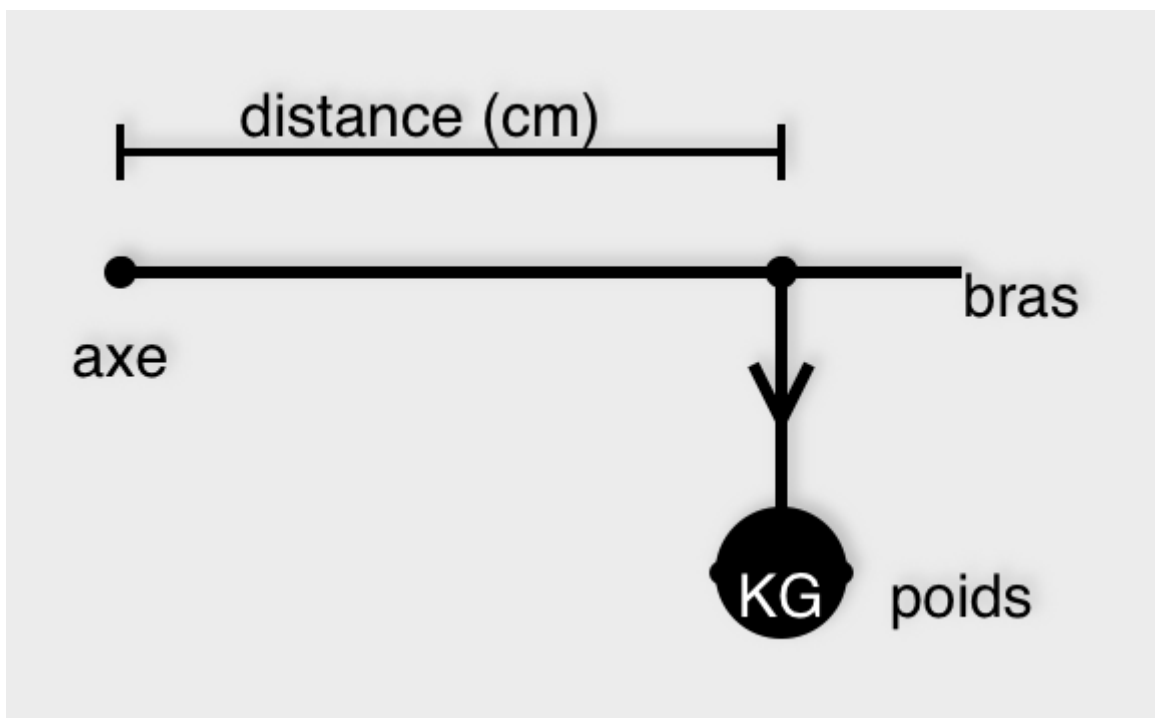
Servomoteur Dagu RS001A (source : gotronic.fr)

Vous pouvez voir sur l'image, grâce au boîtier transparent, les trains d'engrenages à l'intérieur du servo-moteur.

Le couple de ce servo est de 1,5 kg.cm.

Mais c'est quoi au juste cette histoire de couple ?

Le couple est l'effort de rotation que l'on peut appliquer sur un axe. C'est-à-dire, quelle force est nécessaire pour que l'axe tourne. Voici un petit schéma :



L'axe est celui du servo-moteur. Le bras représente la partie solide sur laquelle on va fixer quelque chose. Le poids représente la force qui va être appliquée sur le bras (soit l'axe tourne et le bras entraîne le poids, soit l'axe est fixe et le poids tire sur le bras et provoque sa rotation). La distance qui sépare l'axe de la position du poids est une donnée importante. En

effet, plus le poids est éloigné de l'axe, plus l'effort nécessaire pour le soulever sera important. C'est cette capacité d'effort qui est appelée couple.

La mesure du couple est en kg.cm (kilogramme-centimètre), et pour comprendre, 1,5 kg.cm signifie que le servo pourra soulever un poids de 1,5 kg placé à 1 cm de son axe.

Si l'on déplace ce poids à 10 cm de l'axe, on multiplie par 10 le couple nécessaire pour le soulever. Il faudra donc un servo-moteur de couple 15 kg.cm ! De la même façon, un servo-moteur de couple 1,5 kg.cm ne pourra soulever qu'un poids de 150g s'il est placé à 10 cm. Si cela vous paraît obscur, ne vous découragez pas car c'est un point primordial dans l'utilisation des servo-moteurs. En effet, si vous demandez à un servo-moteur une rotation au-dessus de ses forces (si le couple nécessaire à la rotation est supérieur au couple qu'il fournit) vous risquerez d'endommager votre servo-moteur, et surtout, votre projet ne pourra pas fonctionner.

Donc retenez bien : un servo-moteur a un axe qui fournit une rotation d'une certaine force appelée couple, à ne pas dépasser pour ne pas endommager le moteur.

Bien, nous avons vu le côté moteur, voyons maintenant le côté servo...

On l'a dit au-dessus, servo veut dire esclave. En effet, non seulement un servo-moteur peut tourner, mais surtout, il peut maintenir sa position (tant que son couple le permet) ! C'est d'ailleurs très souvent pour cette utilisation qu'il est apprécié. On lui commande de se mettre à une position précise, **il se positionne puis ne bouge plus.**

Le moteur ne tourne plus ?

Attention, c'est l'axe du servo-moteur qui ne bouge plus. Le moteur, lui est toujours en action, mais il tourne très peu dans un sens puis dans l'autre pour justement garder cet équilibre. C'est ce qu'on appelle l'asservissement. Ceci est rendu possible grâce à un peu d'électronique et aussi à un potentiomètre (voir le [chapitre précédent](#)).

En effet, nous allons demander, grâce à un code que nous verrons plus loin, au servo-moteur de se placer à une certaine position. Le moteur va tourner, entraîner son train d'engrenages qui va faire tourner l'axe du servo-moteur. À cet axe est lié un potentiomètre qui tourne en même temps ! C'est rusé, car il récupère de manière électrique (comme nous l'avons fait précédemment) la position de rotation du servo-moteur. Il la compare à la demande et ajuste la position en envoyant des ordres au moteur (son esclave).

Bon, ce n'est pas le potentiomètre qui envoie des ordres, mais un petit circuit électronique contenu dans le servo-moteur.

Du coup, notre servo-moteur maintient sa position (si le couple est suffisant pour le faire).

Les contraintes de rotation d'un servo-moteur

Il faut savoir qu'un servo-moteur est souvent limité dans sa rotation. En effet, il ne peut tourner que d'un demi-tour, soit 180° (Non Lukas, ce n'est pas si chaud que ça !). Il peut donc prendre 180 positions et les tenir.

Si vous lui demandez de se mettre à 30°, puis 170°, il ira d'une position à l'autre le plus rapidement possible pour ses capacités. On ne peut donc pas gérer facilement la vitesse de rotation d'un servo-moteur !

Certains servo-moteurs sont dits à rotation continue (c'est-à-dire qu'ils ne sont pas bloqués dans un sens ou dans l'autre).

L'utilisation et la programmation des servo-moteurs de ce chapitre concernent des servo-moteurs d'amplitude 180°.

Bien, je pense que nous avons suffisamment survolé le principe du servo-moteur. Voyons maintenant comment on le connecte à notre carte Arduino pour le mettre en mouvement...

Connexion d'un servo-moteur à une carte Arduino

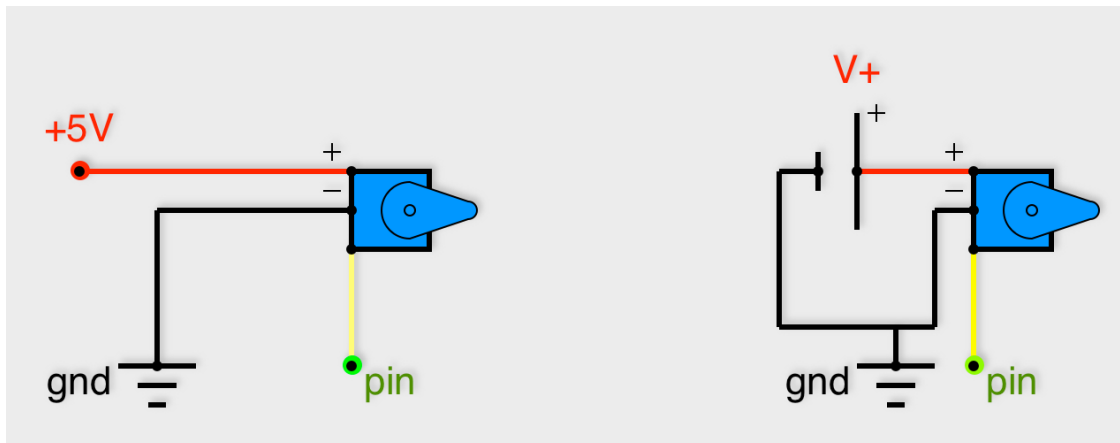
Les servo-moteurs que vous utiliserez fonctionnent à l'électricité. On peut donc s'attendre à voir au moins deux fils, mais en fait, il y en a trois !

- Le fil rouge qui se connecte à l'alimentation (voir avec le servo utilisé, souvent aux alentours de 5V),
- Le fil noir (parfois marron) qui se connecte au ground,
- Le fil jaune (parfois orange ou blanc) qui va servir de commande. Il se connecte à n'importe quelle sortie numérique de l'Arduino (de 0 à 13).

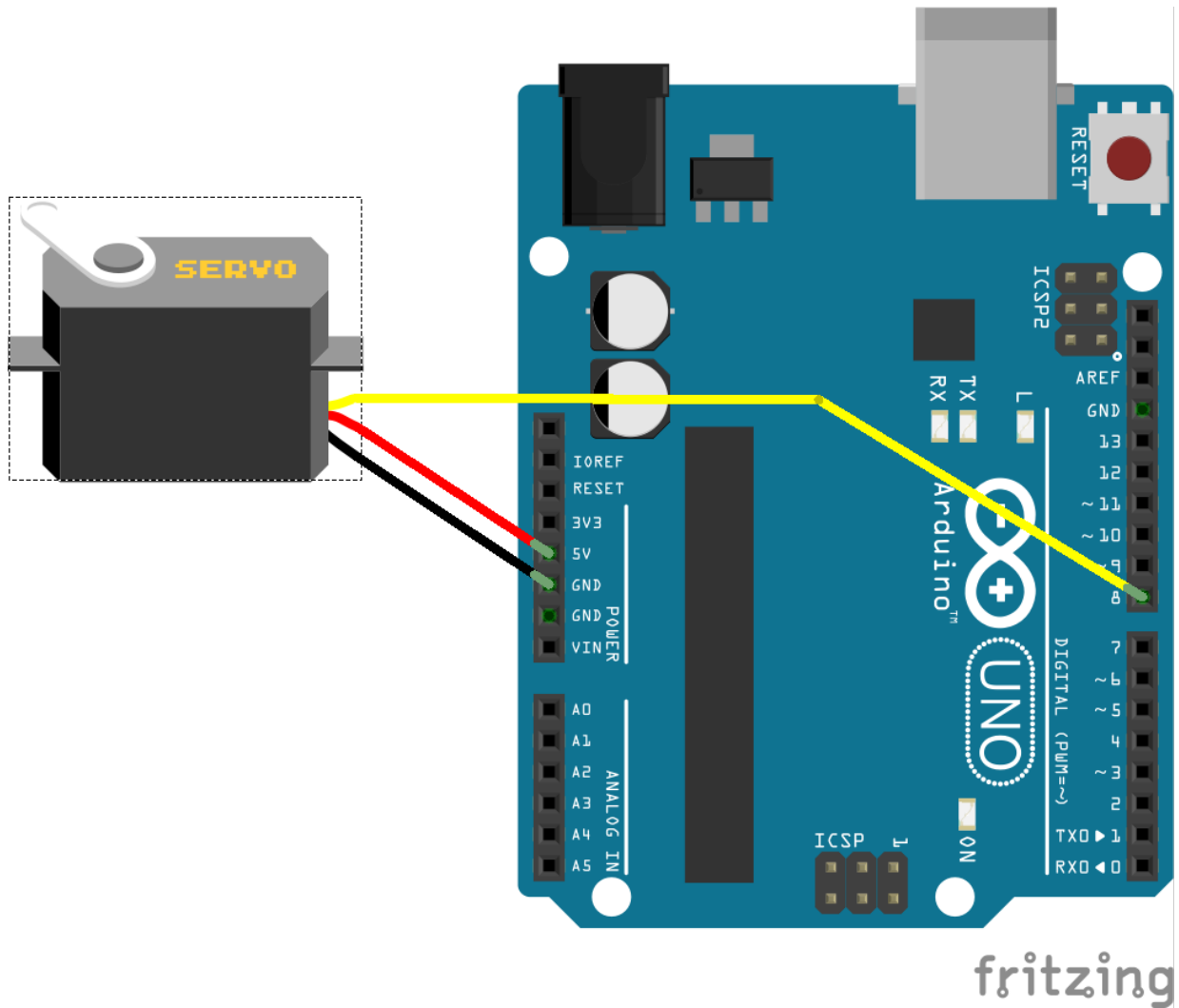
Ne vous trompez pas dans les connexions. Vérifiez à deux fois avant de faire passer le courant. En effet, si vous inversez les fils, vous risquez de griller le servo-moteur.

Du fait que le servo-moteur utilise un moteur, il peut être préférable de l'alimenter par une source différente que celle fournie par l'Arduino. Dans ce cas, il est nécessaire que les ground de l'Arduino, de l'alimentation et du servo-moteur soient connectés ensemble.

Voici le schéma du montage électrique :



Connexion d'un servo-moteur à une carte Arduino
Et voici la représentation Fritzing du montage :



Connexion d'un servo-moteur au pin 8 de l'Arduino

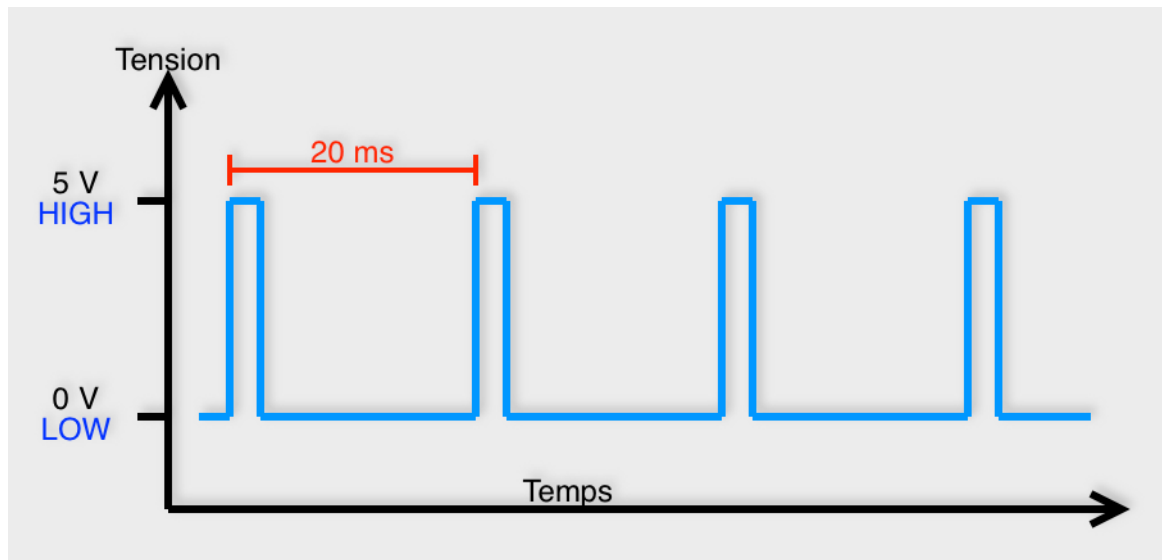
Et là j'aurais tendance à dire : c'est fini ! Votre servo-moteur est connecté et prêt à l'emploi !

Voyons maintenant ce que nous pouvons lui demander...

Envoyez des ordres à un servo-moteur

Pour commander un servo-moteur, il faut lui envoyer ce qu'on appelle un train d'impulsions électriques (oui Lukas, si vous voulez, un train électrique...), on peut traduire par : des envois de courant électrique qui se suivent à intervalle et durée précis. L'intervalle de temps entre chaque impulsion envoyée est aussi appelé **période**.

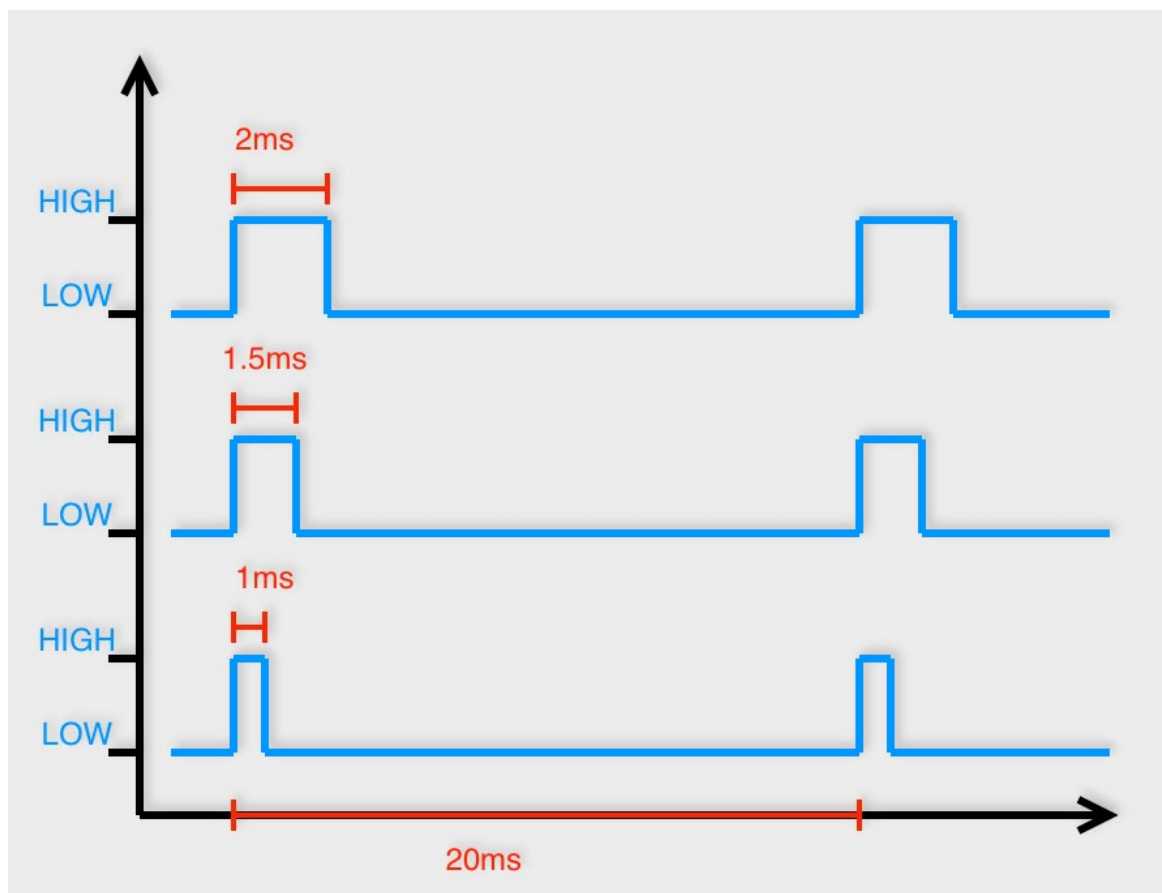
Voici à quoi ressemble un train d'impulsions :



Il faut observer plusieurs choses :

- La valeur de la tension est soit +5V (HIGH de l'Arduino) soit 0V (LOW de l'Arduino),
- Les impulsions sont envoyées régulièrement, ici toutes les 20 millisecondes. C'est la durée généralement utilisée pour piloter un servo-moteur.

La durée de l'impulsion peut varier. C'est d'ailleurs grâce à elle que nous allons piloter notre servo-moteur.



Et voici comment le servo-moteur va interpréter ces impulsions :

- Lors d'une impulsion à 1ms : le servo-moteur se met à la position 0°,
- Lors d'une impulsion à 1.5 ms : le servo-moteur se met à la position 90°,
- Lors d'une impulsion à 2ms : le servo-moteur se met à la position 180°.

On peut donc facilement faire un mappage des valeurs pour obtenir la durée de l'impulsion en fonction de l'angle :

```
int dureeImpulsion=map(angle,0,179,1000,2000);
```

La valeur de l'angle est entre 0° et 179°, le résultat sera entre 1 000 et 2 000 microsecondes (donc entre 1 et 2 millisecondes). Grâce à cette formule, on peut obtenir toutes les valeurs d'impulsions pour des positions entre 0° et 179°.

Mais comment envoyer une impulsion aussi précisément ?

Et bien nous allons utiliser l'horloge interne de l'Arduino ! Nous l'avons d'ailleurs déjà fait grâce à la fonction `delay()` mais pour calculer en microsecondes, nous avons une autre fonction :

```
delayMicroseconds(valeur);
```

Il suffit ensuite d'envoyer du courant (HIGH) ou non (LOW) dans le PIN de commande du servo, en respectant la durée de l'impulsion et le temps entre chaque impulsion.

Je vous livre un code qui va nous permettre d'incliner le servo-moteur à différents angles entre 0° et 179° à partir de l'Arduino :

```
/*
Comande de servo-moteur par impulsion

Nanomaitre 2015

Le servo est connecté au pin 8 de l'arduino
*/

int periode=20000;// période entre chaque début d'impulsion en microsecondes

int pinServo=8; // variable pour le pin connecté à la commande du servo

void setup() {

  pinMode(pinServo,OUTPUT);// on prépare le pin en mode OUTPUT

  digitalWrite(pinServo,LOW); // on l'initialise à l'état bas
}
```



```

//boucle principale

void loop() {

    for (int angle=0;angle<=180;angle+=20){//on fait varier l'angle de 0 à 180° par tranche
de 20°

        setAngle(angle);// on appelle la fonction setAngle définie plus bas

    }

}

//fonction setAngle pour envoyer les impulsions

void setAngle(int a){

    int duree=map(a,0,179,1000,2000);// on transforme l'angle en microsecondes et on stocke
dans la variable duree

    digitalWrite(pinServo,LOW);//on met le pin à l'état bas

    // la boucle qui suit est nécessaire

    // pour laisser le temps au servo d'aller à sa position

    for (int t=0;t<300;t++){

        digitalWrite(pinServo,HIGH);// on envoie l'impulsion

        delayMicroseconds(duree); // pendant la bonne durée

        digitalWrite(pinServo,LOW); // on stoppe l'impulsion

        delayMicroseconds(periode-duree); // on attend le temps restant pour atteindre la
période

    }

}

```

Envoyez et testez ! Si tout se passe bien, le servo prend une position extrême (0°) puis se déplace par tranche de 20°, puis lorsqu'il atteint les 180°, il revient à la position de départ et recommence.

Plusieurs cas de figure peuvent se présenter et empêcher le programme de fonctionner correctement :

- **Le servo-moteur ne revient pas à sa position de 0°** : il faut alors augmenter le nombre de boucles effectuées par le compteur `t` dans la fonction `setAngle()`. En effet, si la vitesse de rotation de votre servo ne lui permet pas d'atteindre le zéro en 300 boucles, il lui faut plus de temps.
- **Le servo-moteur ne parcourt pas un demi-tour complet** : il faut alors changer les valeurs dans le mappage. En effet, tous les servos ne sont pas réglés de manière identique. Par exemple, pour celui que j'utilise sur ce test, j'ai remplacé les valeurs 1 000 et 2 000 par 500 et 2 500 respectivement.

Vous serez souvent amenés à chercher les valeurs à utiliser pour les limites d'impulsion par vous-mêmes. L'idée est de trouver la valeur limite (haute et basse) afin de programmer votre Arduino pour ne jamais la dépasser ! (La fonction de mappage est tout à fait adaptée dans ce cas.)

Que se passe-t-il si l'on envoie pas assez longtemps le train d'impulsions ?

Et bien le servo ne rejoindra pas sa position. De plus, même s'il l'atteint et que le train d'impulsions s'arrête, il ne maintiendra pas sa position angulaire. Ce qui est fâcheux...

Prenez l'exemple d'un servo qui est utilisé pour faire fonctionner un bras robotique, s'il ne maintient pas la position demandée, le bras se pliera si le poids relié au servo est assez lourd (souvent le poids du bras lui-même suffit !).



La course du servo-moteur est souvent arrêtée par une cale qui stoppe le mouvement d'un côté comme de l'autre. L'obliger à aller plus loin le fait forcer sur cette cale. Certains bricoleurs habiles font sauter ces cales (en démontant le servo) et obtiennent un servo à rotation continue. Pour le commander d'un côté, il suffit de laisser le pin sur HIGH et de l'autre sur LOW.

Bien, vous avez compris et testé comment se pilote un servo-moteur par un train d'impulsions électriques. Et bien je vous annonce que l'équipe d'Arduino a conçu une bibliothèque spéciale qui gère tout ceci bien plus facilement : la bibliothèque Servo !

La bibliothèque Servo

Je vous ai déjà parlé des bibliothèques. Ce sont des programmes et des constantes stockées dans des fichiers que l'on peut inclure dans nos programmes.

Elles sont réalisées par les ingénieurs de chez Arduino, ou par toute personne qui se sent capable d'en créer une qui répond à un besoin particulier. Nous verrons comment créer une bibliothèque bien plus loin dans un autre cours. Pour le moment, il suffit de savoir que votre programme peut s'enrichir du travail des autres, gratuitement, et souvent avec performance.

Vous avez déjà utilisé la bibliothèque Serial

(`Serial.begin(9600)` , `Serial.print()` , `Serial.println()`). Ces commandes sont des fonctions incluses dans la bibliothèque Serial. On le repère d'ailleurs par la présence du mot-clé `Serial` devant chaque nom de fonction.

La bibliothèque Serial est incluse par défaut dans chaque programme que vous démarrez. Il vous suffit d'en appeler les fonctions. En revanche, la bibliothèque Servo, doit être appelée dans le programme, afin qu'elle soit ajoutée au programme. On inclue une bibliothèque avec un mot-clé spécial. Voici l'exemple pour la bibliothèque Serial :

```
#include <Servo.h>
```

Ceci est une directive pré-compilation. C'est-à-dire que l'IDE, avant même de chercher à transformer votre programme en langage machine, va aller chercher la bibliothèque sur votre ordinateur, ajouter ses lignes de codes aux vôtres et créer un programme complet. Ensuite seulement, il va compiler.

Observez bien la structure : `#include` puis `<nomDeLaBibliotheque>` et pas de point-virgule à la fin.

Je vous propose une petite expérience :

1. Ouvrez un programme vide et nommez-le "test",
2. Cliquez sur le V de vérifier,
3. Lisez le message de l'IDE (en bas) qui vous informe de la taille de votre programme et des variables,
4. Essayez de retenir en gros ce qui est dit,
5. Ajoutez au tout début de votre programme la ligne de code que l'on vient de voir pour inclure la bibliothèque Servo (il faut la mettre avant le `setup()`),
6. Cliquez sur vérifier et observez les changements.

Votre programme prend 3 fois plus de place en mémoire juste avec cette ligne. Tout simplement parce que cette ligne en a ajouté bien d'autres !



Tous deux représentent la librairie Servo, réalisée en C++. Elle utilise les timers de l'Arduino (qui gèrent le temps qui passe) et permet de proposer des fonctions qui maintiennent le servo en position tout en exécutant une autre tâche !

Les bibliothèques utilisent des classes que l'on peut simplifier en les imaginant comme des objets programmables, c'est-à-dire des objets qui ont plusieurs fonctions spécifiques.

Souvent pour utiliser une bibliothèque, il faut créer un objet lié à cette bibliothèque. C'est le cas pour la bibliothèque Servo. Nous allons créer un objet de type Servo et lui donner un nom :

```
Servo monServo;
```

Maintenant, à chaque fois que nous ferons référence à "monServo" il s'agira de cet objet que nous venons de créer. Pour qu'il soit accessible partout (pensez au scope des variables) on le crée généralement avant le `setup()` .

Bien, ensuite il faut relier cet objet au pin de commande que l'on a choisi (par exemple le pin 8). On va utiliser une autre fonction qui est assez transparente :

```
monServo.attach(8);
```

On remarque que l'on commence par le nom de l'objet puis on appelle une fonction qu'il peut comprendre.

Enfin pour mettre le servo à l'angle désiré, on appelle une fonction de l'objet monServo qui là encore est claire comme de l'eau de roche, voyez vous-même :

```
monServo.write(angle);
```

Où "angle" est la valeur de l'angle que l'on désire.

Voici un code qui place un servo en position 0° puis 180° d'une seconde à la suivante :

```
#include <Servo.h> //on importe la bibliothèque Servo

int pinServo=8; // variable pour stocker le pin pour la commande

Servo leServo; // on définit un objet Servo nommé leServo

void setup() {

    leServo.attach(pinServo); // on relie l'objet au pin de commande

}

void loop() {

    leServo.write(0); // on dit à l'objet de mettre le servo à 0°

    delay(1000); // ce délai est nécessaire pour que le servo atteigne sa position

    leServo.write(179); // position à 179, 180 est à éviter car cela forcerait le servo à dépasser ses limites

    delay(1000); // attente à nouveau

}
```

Les commentaires devraient suffire à comprendre.



```
#include <Servo.h> //on importe la bibliothèque Servo

int pinServo=8; // variable pour stocker le pin pour la commande

Servo leServo; // on définit un objet Servo nommé leServo

void setup() {
```

```

leServo.attach(pinServo); // on relie l'objet au pin de commande

pinMode(13,OUTPUT); //pin 13 en mode OUTPUT
}

void loop() {

    leServo.write(0); // on dit à l'objet de mettre le servo à 0°

    diode13(); // appel de la fonction diode13() définie plus bas

    leServo.write(179); // position à 179°, 180° est à éviter

    diode13(); // appel de la fonction

}

void diode13(){

    // on fait clignoter 30 fois la LED 13

    for (int t=0;t<30;t++){

        digitalWrite(13,HIGH);

        delay(100);

        digitalWrite(13,LOW);

        delay(100);

    }

}

```

Vous remarquerez que la LED attachée au pin 13 clignote alors que le servo-moteur se déplace. De plus, si vous essayez de faire tourner le servo à la main sans trop forcer, vous sentirez une résistance à la rotation qui montre que le servo maintient sa position (Lukas, j'avais dit doucement ! Vous êtes bon pour casser votre tirelire maintenant...).

Je vous laisse essayer plusieurs valeurs pour les angles, les nombres de boucles et les temps d'attente.

Potar et Servo

Lorsque je vous disais au début de chapitre que c'est la partie de l'Arduino qui me fait toujours un petit effet, il s'agit en fait de ce qui suit...

Vous savez maintenant utiliser un servo-moteur. Vous savez aussi lire les valeurs d'un potentiomètre. Que diriez-vous de commander votre servo-moteur grâce au potentiomètre ? L'idée est simple, le servo se positionne en fonction de la position du potentiomètre.

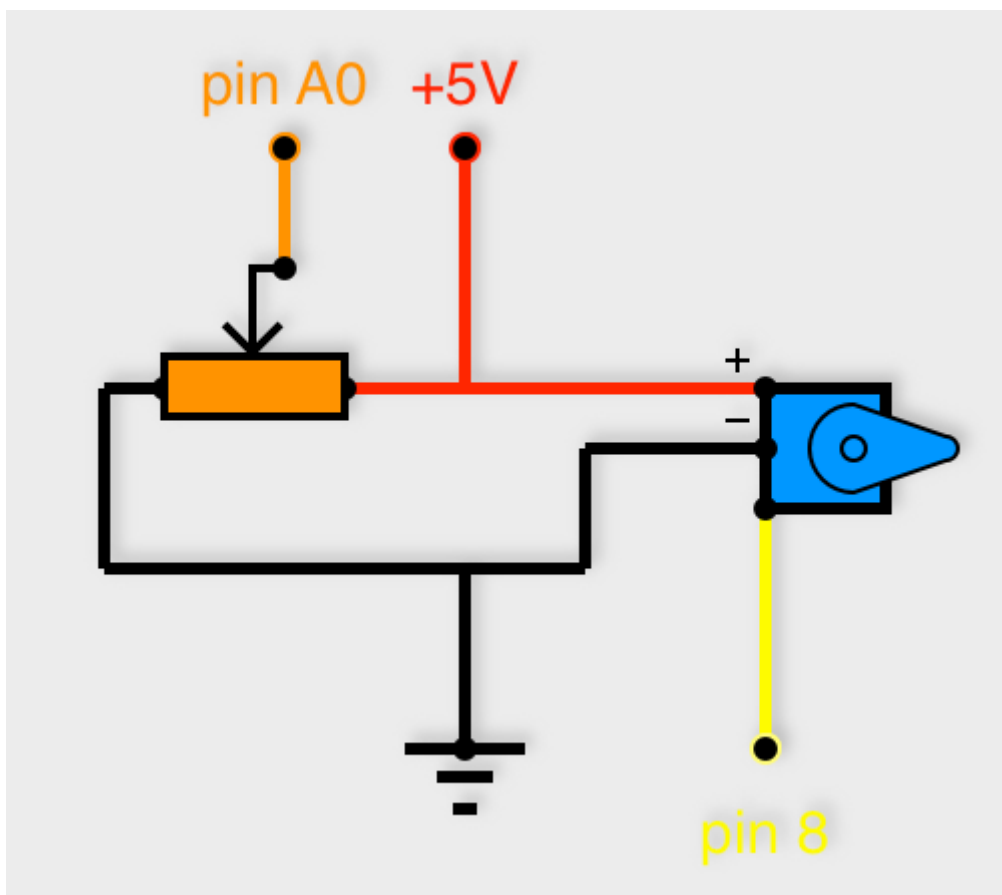
Techniquement, vous avez tout ce qu'il vous faut pour réaliser ce programme sans aide, et je vous conseille d'essayer !

Quelques indices :

- Branchez correctement le potentiomètre et le servo-moteur (on ne sait jamais...).
- Prévoyez les variables de lecture du potentiomètre, de position du servo, la bibliothèque, l'objet Servo.
- Avec un mappage habile des valeurs, transformez la position du potentiomètre en angle.
- Positionnez le servo.



Voici le schéma du montage électrique :



Connexion d'un servo-moteur et d'un potentiomètre
Et voici le code qui va bien...

```
#include <Servo.h> // on importe la bibliothèque
```

```

Servo servo; // on crée l'objet Servo

int pinServo=8; // on définit le pin lié à la commande du servo

int pinPotar=A0; // on définit le pin lié à la lecture du potentiomètre

void setup() {

    servo.attach(pinServo); // on relie l'objet servo au pin de commande

}

void loop() {

    int valeurPotar=analogRead(pinPotar); // lecture de la valeur du potentiomètre

    int angle=map(valeurPotar,0,1023,0,179); // tranformation en angle

    servo.write(angle); //mise en position du servo

}

```

Un tout petit bout pour le code, un grand mouvement pour le servo !

Pour les plus curieux, vous vous êtes sûrement demandé si l'on peut récupérer la position d'un servo-moteur avec une commande du genre `servo.read()` ?

En effet, la commande `read()` existe dans la bibliothèque Servo. Elle renvoie la dernière valeur de position envoyée au servo-moteur, mais pas la position réelle du servo ! Et c'est malheureusement impossible de la connaître avec un servo de base. Il existe des servo-moteurs qui proposent cette possibilité avec un fil supplémentaire pour la lecture de la position. Vous ne pouvez donc pas tester l'arrivée du servo à une position précise pour déclencher un évènement. Seulement l'angle que vous lui avez demandé.

Bien, pour finir, je vous propose de réaliser un programme qui utilise un servo et un bouton poussoir.

L'accélérateur

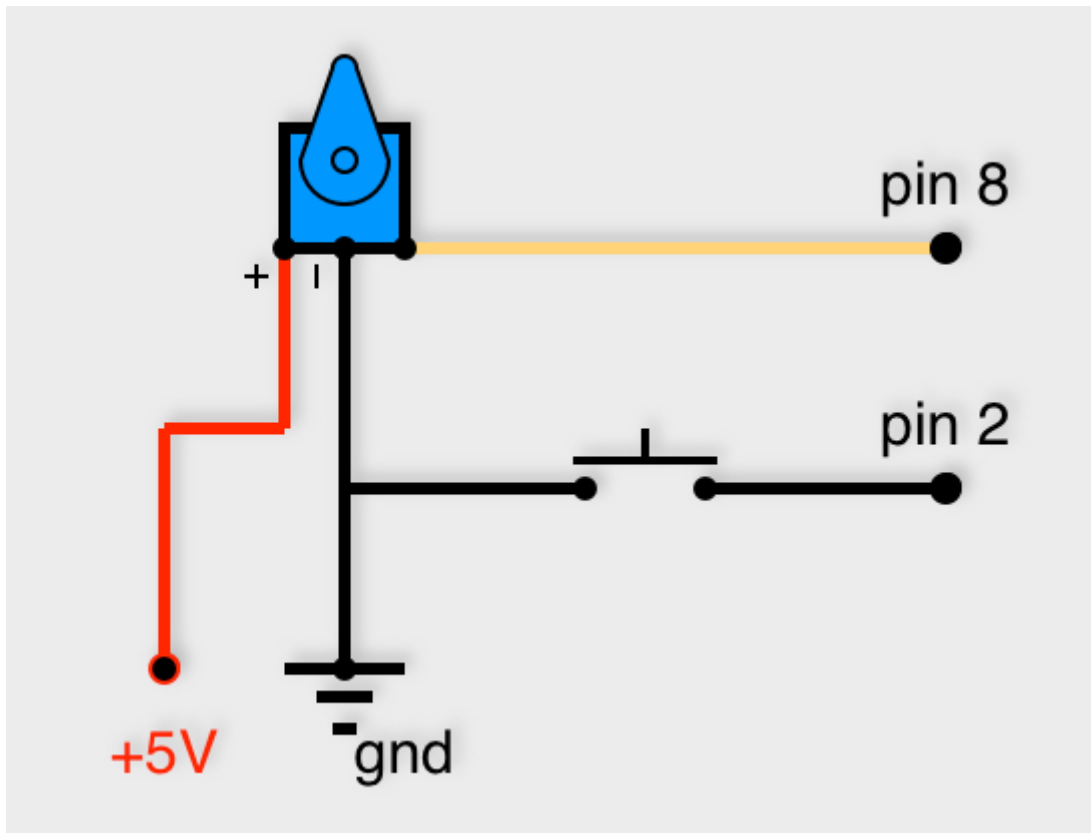
Le principe du programme est le suivant :

- Le servo est à 0° au départ.
- Lorsqu'on appuie sur le bouton poussoir (en restant appuyé) le servo passe à 45° puis 90° puis 135° et 180° en fonction du temps d'appui.
- Lorsqu'on relâche, le servo "redescend" par les positions inverses, avec les mêmes paliers de temps, jusqu'à 0°.
- Si on appuie durant la "descente" le servo "remonte".

La difficulté de ce programme réside dans la méthode que vous allez utiliser pour la durée d'appui et la reprise d'appui sur le bouton.

On connectera le bouton poussoir sur le pin 2 en mode INPUT_PULLUP, le servo sur le pin 8.

Voici le schéma électrique :



Je vous propose de réaliser ce programme en deux temps.

1. Premier programme : le servo bouge en fonction du temps d'appui et redescend, sans les paliers d'angles (0,45,90...) ;
2. Second programme : on solutionne les paliers d'angles (et on apprend du même coup un mot de programmation : `switch()`).

Accélérateur sans paliers d'angles

Le servo se se déplace vers sa position maximale tant qu'on appuie, et revient lorsqu'on relâche.

Pour réussir ce premier programme, il vous faut :

- Un objet Servo attaché au pin 8,
- Le pin 2 en mode INPUT_PULLUP et relié au bouton poussoir,
- Une variable qui stocke le temps d'appui, (je l'ai appelée `cumul`, mais faites-vous plaisir si vous avez d'autres idées de nom)
- Un mappage du temps d'appui vers un angle pour le Servo,
- Des limites pour le temps d'appui.

Je vous laisse réaliser ce programme en essayant de respecter ces points. Vous avez maintenant tout ce qu'il faut pour réussir. Il est très probable que vous ne réussissiez pas du

premier coup (bugs, oublis, test erronés...) et c'est nécessaire pour mieux programmer par la suite. Allez ! au boulot !

Comme l'objectif de ce programme reste d'apprendre à programmer, voici le code que je vous propose :

```
#include <Servo.h> //import de la bibliothèque Servo

Servo accel; //création de l'objet Servo "accel"

int pinServo=8; //pin de commande du servo

int pinBouton=2; //pin de lecture du bouton poussoir

int cumul=0; //variable d'appui

void setup() {

    pinMode(pinBouton,INPUT_PULLUP); //mode INPUT_PULLUP pour le poussoir

    accel.attach(pinServo); //liaison de l'objet Servo au pin de commande

    Serial.begin(9600); //pour lecture sur la console (Optionnel)

}

void loop() {

    boolean etatBouton=digitalRead(pinBouton); //lecture de l'état du bouton

    //si le bouton est appuyé

    if (!etatBouton){ // en mode INPUT_PULLUP on obtien 0 quand on appuie !

        cumul++; // on fait augmenter la valeur de la variable

        if (cumul>1000) //test limite d'augmentation

            cumul=1000; //mise à limite si dépassement

    }

}
```

```

//si le bouton n'est pas appuyé

else{

    cumul--; //on fait diminuer la valeur de la variable

    if (cumul<0) //test si limite de diminution

        cumul=0;//mise à la limite si dépassement

}

Serial.println(cumul); //on affiche la valeur sur la console (Optionnel)

int angle=map(cumul,0,1000,0,179); //on transforme en angle

accel.write(angle); //on positionne le servo
}

```

La condition `if(!etatBouton)` peut paraître étrange, mais nous l'avons déjà vu, en mode `INPUT_PULLUP` quand le contact se fait, la valeur renvoyée est 0. On utilise donc un test du genre `if(!contact)` qui signifie est équivalent à `if(contact==0)` .
 Avant d'aborder le second programme je vais vous apprendre un nouveau mot-clé...

... La condition SWITCH !

Vous connaissez la condition `if` qui réalise un test et exécute un code si il est vérifié. Allez, un petit rappel ne vous fera pas de mal :

```

int nombreDePersonnes=5;

int nombreDeChaises=4;

if (nombreDeChaises<nombreDePersonnes){

    Serial.print("Il y en a qui resteront debout...");

}

```

Vous connaissez les conditions `if` , `else if` et `else` qui permettent de réaliser le test suivant si le premier n'est pas vérifié. Allez, un deuxième petit rappel ne vous fera pas de mal non plus :

```

int nombreDePersonnes=5;

int nombreDeChaises=3;

```

```

if (nombreDeChaises==nombreDePersonnes){

    Serial.print("Tout le monde peut s'asseoir");

}

else if (nombreDePersonne-nombreDeChaises==1){

    Serial.print("Il y en a 1 qui reste debout");

}

else if (nombreDePersonne-nombreDeChaises>1 && nombreDeChaise!=0){

    Serial.print("Il y en a plusieurs qui restent debout");

}

else {

    Serial.print("Tout le monde reste debout");

}

```

J'ai volontairement mis plusieurs sortes de test, pour vous montrer des possibilités. Ici, on ne passe au test suivant que si le test précédent n'a pas été vérifié. Si un test est vérifié, les autres ne seront pas testés.

L'ordre et le contenu des tests doit être précis et réfléchi. En effet, des tests mal réalisés sont souvent la raisons de bien des dysfonctionnements de programmes...

Et bien il existe une autre façon de réaliser des conditions. Il s'agit de la commande `switch`.

On l'utilise en lui indiquant une valeur (variable de type `int`, ou bien de type `char`) puis les actions à faire en fonction de cette valeur. “To switch” en anglais signifie “changer, intervertir”, c’est-à-dire que l’on va passer d’un cas à l’autre, les tester et agir en conséquence.

Voici les exemples précédents utilisés avec une fonction `switch` :

```

int nombreDePersonnes=5;

int nombreDeChaises=3;

int diff=nombreDePersonnes-nombreDeChaises;

switch (diff){

    case 0:

```

```

    Serial.print("Tout le monde peut s'asseoir");

    break;

case 5:

    Serial.print("Tout le monde reste debout");

    break;

case 1:

    Serial.print("Il y en a 1 qui reste debout");

    break;

default:

    Serial.print("Il y en a plusieurs qui restent debout");

}

```

Donc

- On commence par indiquer au switch la valeur à tester (ici la différence entre chaise et personnes) : `switch(diff)`.
- Lignes 6, 9 et 12, le switch compare la valeur à tester (ici `diff`) avec celle qui suit le mot `case` (ici 0, 5 ou 1). Il est impératif de mettre les `:` en fin de ligne.
- Le mot-clé `break` sert à sortir des tests (le test étant positif). Il est impératif de l'accompagner d'un point-virgule `;`. Si vous ne mettez pas une instruction `break;` pour vous échapper des tests, le programme testera toutes les possibilités suivantes avant de rencontrer un `break`.
- Le mot-clé `default:` n'est pas obligatoire, mais il permet de proposer une solution par défaut. c'est-à-dire si aucun des tests précédents n'a été validé.

Les valeurs après case doivent être des constantes. Des variables créent une erreur.

Revenons maintenant à notre accélérateur... et ajoutons des paliers d'angles !

Accélérateur avec paliers d'angles

Nous allons utiliser la fonction `switch` pour définir les paliers d'arrêt pour le servo. Voici le code :

```

#include <Servo.h> //import de la bibliothèque Servo

Servo accel; //création de l'objet Servo "accel"

int pinServo=8; //pin de commande du servo

```

```
int pinBouton=2;//pin de lecture du bouton poussoir

int cumul=0; //variable d'appui

void setup() {

    pinMode(pinBouton,INPUT_PULLUP); //mode INPUT_PULLUP pour le poussoir

    accel.attach(pinServo); //liaison de l'objet Servo au pin de commande

    Serial.begin(9600);//pour lecture sur la console (Optionnel)
}

void loop() {

    boolean etatBouton=digitalRead(pinBouton); //lecture de l'état du bouton

    //si le bouton est appuyé

    if (!etatBouton){// en mode INPUT_PULLUP on obtien 0 quand on appuie !

        cumul++; // on fait augmenter la valeur de la variable

        if (cumul>1000) //test limite d'augmentation

            cumul=1000;//mise à limite si dépassement

    }

    //si le bouton n'est pas appuyé

    else{

        cumul--; //on fait diminuer la valeur de la variable

        if (cumul<0) //test si limite de diminution

            cumul=0;//mise à la limite si dépassement

    }

}
```

```
}

Serial.println(cumul); //on affiche la valeur sur la console (Optionnel)

int pos=map(cumul,0,1000,0,4);//on mappe de 0 à 4 (5 positions) dans un variable pos

int angle=0; //on initialise une variable angle

switch (pos){// va switcher en fonction de la valeur pos

    case 1:

        angle=45;

        break;

    case 2:

        angle=90;

        break;

    case 3:

        angle=135;

        break;

    case 4:

        angle=179;

        break;

    default:

        angle=0;

}

accel.write(angle); //on place le servo à la position angle
```

}



Rien ne vous empêche ensuite de réaliser sur une feuille un cadran pour repérer les positions du servo et de positionner une aiguille sur l'axe du servo... mais je suis sûr que vos idées ne manquent pas !

Je pense que ce sera suffisant pour les servo-moteurs. Plein de nouveautés vous attendent dans le prochain chapitre, alors détendez-vous un peu avant d'enchaîner !

Le moteur à courant continu (partie 1) : transistors et sorties PWM

Vous avez vu au chapitre précédent le servo-moteur, qui permet de réaliser des mouvements de rotation jusqu'à 180°. Mais s'il s'agit de réaliser un robot qui roule, un servo-moteur ne peut pas être utilisé pour la propulsion (ou la traction). Il nous faut donc utiliser un moteur qui peut tourner infiniment dans un sens ou dans l'autre. C'est le moteur à courant continu.

Deux chapitres vont être nécessaires pour bien comprendre l'utilisation de ce type de moteur.

Dans ce chapitre :

- Vous découvrirez d'abord ce que c'est qu'un moteur à courant continu.
- Puis vous apprendrez comment connecter un tel moteur à l'Arduino, et surtout, comment le commander par programmation.
- Enfin vous apprendrez à faire varier sa vitesse grâce à des sorties numériques spéciales : les PWM.

Vous allez du même coup découvrir de nouveaux composants : les transistors et les MOSFET !

Alors j'ai envie de vous dire... qu'attendons-nous ?!

Le moteur à courant continu

Si vous ne vous sentez pas l'âme mécanique, vous pouvez passer au [point suivant](#) 😊

Dans le monde qui nous entoure, depuis l'invention des principes de rotation (faire tourner un truc autour d'un autre, ou sur lui-même ; vers 3500 AV J-C pour la roue) et de la mécanisation (la machine fait à la place de l'homme ; au Moyen-Âge pour les moulins) les moteurs ont pris une place importante. Il en existe de multiples sortes et surtout qui fonctionnent avec des énergies et des principes physiques différents. Mais un principe les relie tous : l'axe du moteur tourne.

Nous allons aborder ici un type de moteur qui peut être utilisé avec l'Arduino : le moteur à Courant Continu ou moteur CC (moteur DC pour *Direct Current* en anglais). Vous allez vous

apercevoir que son utilisation est simple tant qu'il ne s'agit pas de le commander par programmation. Mais avant tout, voyons un peu comment il fonctionne...

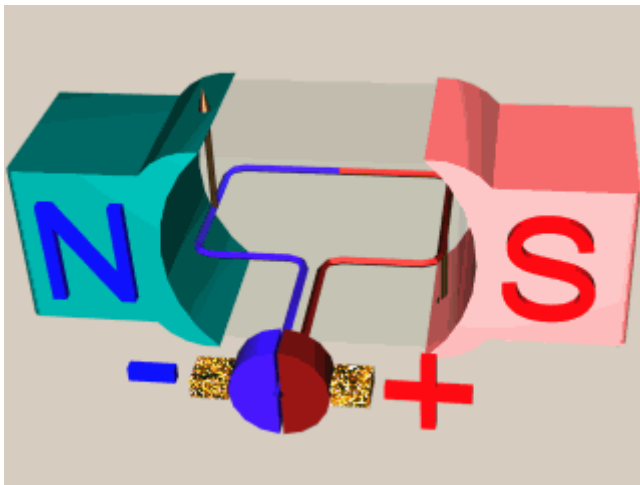
Le courant induit

Je vous l'ai dit plus haut, un principe fondamental du moteur c'est que son axe doit tourner.

Prenons l'exemple d'un moulin : le vent souffle sur la plaine de la Bretagne armoricaine. Sur cette plaine est placé un moulin. Le vent (qui est un déplacement d'air) appuie sur les ailes du moulin. Leur forme en biais fait que l'action de l'air les déplace. Comme chaque aile est accrochée au centre (l'axe), il se met à tourner. Le moulin est donc un moteur à air.

Pour un moteur à électricité c'est un peu la même chose : il y a un axe, il y a des ailes, et il y a un déplacement des ailes grâce à l'énergie électrique.

Voici une petite animation qui va vous éclairer un peu :



Visualisation simplifiée de la rotation d'un moteur CC (elek.chez.com)

Le déplacement des ailes de ce moteur schématisé est lié à un déplacement d'électricité. Mais contrairement au moulin, c'est l'électricité qui passe dans les ailes qui va provoquer le déplacement.

Vous repérez sur ce schéma deux pôles (N et S). Ce sont deux aimants qui sont fixes (statiques) dans le moteur. Aucune électricité ne les parcourt, mais les aimants sont, par défaut, magnétiques. Ça veut dire qu'il ont autour d'eux un champ qui fait qu'ils se repoussent ou s'attirent (vous avez je pense déjà fait l'expérience).

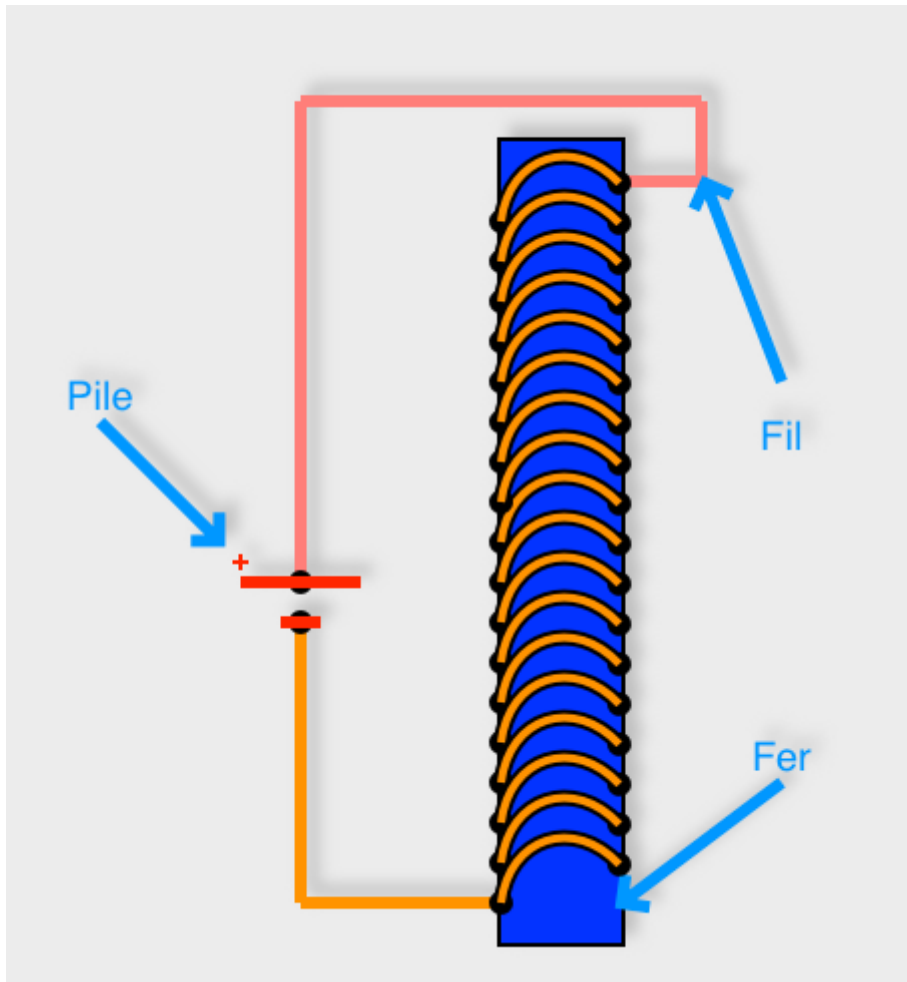
Et bien il est possible de fabriquer un aimant grâce au passage de l'électricité !

En étudiant l'électricité et les phénomènes qui y sont liés, les chercheurs ont découvert qu'il y a un rapport entre le courant et le magnétisme. En effet, un courant qui passe dans un circuit fermé, dans certaines conditions, crée un champ magnétique (et à l'opposé, un champ magnétique qui se déplace dans un circuit fermé, dans certaines conditions, crée du courant). C'est ce qu'on appelle l'induction.

Je ne vais pas entrer dans les détails des lois de l'induction et des théorèmes qui y sont liés car ce n'est pas l'objectif de ce chapitre. Nous allons juste en observer les résultats pour mieux comprendre le moteur CC.

Alors pour fabriquer un aimant avec de l'électricité, il suffit d'avoir un morceau de fer (clou par exemple), du fil de cuivre isolé (fil de connexion) assez long, et une source d'électricité (une pile).

Vous enroulez le fil autour du clou, en étant attentif de bien serrer les spires (commencez côté pointe puis allez vers la tête). Vous êtes en train de fabriquer un électro-aimant. Si vous enlevez le clou, vous obtenez une bobine.



Électro-aimant avec une pile



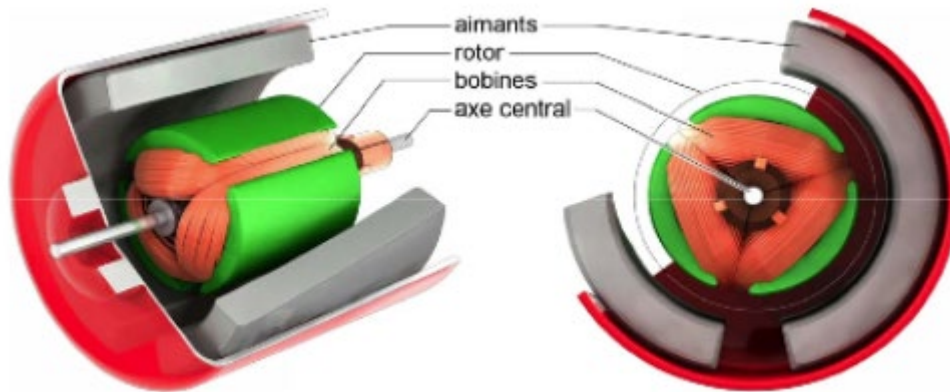
Ne laissez pas la pile branchée plus de quelques secondes, car elle se déchargerait complètement. En effet, le circuit étant très peu résistant, le courant passe dans la bobine de façon très rapide (c'est presque un court circuit) ce qui vide la pile. De même, n'utilisez pas le courant fourni par votre carte Arduino pour réaliser cette expérience.

Pour revenir à notre moteur, en fait les ailes du moteur sont constituées d'électro-aimants (du fil entouré autour d'un noyau en fer), du coup, lorsque l'on fait passer l'électricité, cela crée un pôle magnétique. Les branchements sont faits pour que ce pôle soit le même que celui de l'aimant statique.

On sait que le pôle Nord repousse le pôle Nord, du coup l'aile du moteur va chercher à fuir l'aimant statique. C'est ce qui crée le mouvement du moteur.

Un moteur CC a généralement au moins trois ailes, en effet, cela permet d'être sûr que le mouvement se fasse. De plus, l'arrivée du courant est faite par un système qui permet d'envoyer le courant correctement dans les bobines pour que le mouvement se fasse dans le même sens.

Voici une vue d'un moteur CC coupé :



Vue en coupe

d'un moteur CC (<http://electric-sience.blogspot.fr>)

On y voit bien :

- le corps du moteur et les aimants statiques qui y sont fixés : le tout s'appelle le **stator** car il ne bouge pas.
- Les trois bobines et leur noyau de fer (orange et vert) qu'on appelle le **rotor** car c'est la partie qui tourne.
- L'axe qui est la partie qui sera ensuite utilisée pour récupérer ce mouvement de rotation (en entraînant des engrenages, une poulie...).

Vous remarquerez que la bobine est constituée de plusieurs tours autour du noyau de fer.

Il faut bien comprendre que si l'on fait passer du courant dans un moteur, il tourne : il transforme l'énergie électrique en rotation. Mais il faut retenir que si vous faites tourner l'axe du moteur, il crée du courant, il transforme la rotation en électricité ! C'est important à retenir pour la suite.



Maintenant que vous avez une compréhension de la base de fonctionnement des moteurs CC, je vais vous apprendre à connecter un tel moteur.

Connectez un moteur CC à votre carte Arduino

Pour la suite, je vous propose d'utiliser un moteur CC qui fonctionne avec une tension entre 3V et 6V. C'est souvent indiqué sur le corps du moteur ou sur la datasheet. Sinon, il faut le préciser à l'achat.

Voilà à quoi peuvent ressembler des moteurs CC. Voilà à quoi peuvent ressembler des moteurs CC pour cette gamme de tensions (il en existe de multiples formes) :



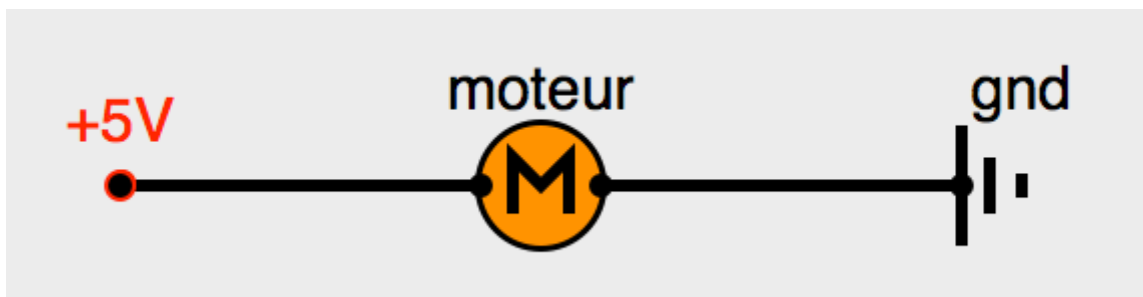
Moteurs CC 6V (amazon.fr)

Vous remarquerez l'axe, et deux petites pattes (de part et d'autre de la partie rose du moteur) pour le branchement. Tout le reste est dans la coque du moteur.

Alors plusieurs remarques :

- Pour se connecter aux pattes, il vous faut souder un fil à chacune ou utiliser un système d'accroche (pince crocodile en métal). Si le contact n'est pas correctement établi, votre moteur aura des ratés ou ne démarrera pas.
- L'axe tourne quand du courant passe dans le moteur. Mais pour transmettre un mouvement à autre chose (une roue, une courroie, un engrenage...), il faudra avoir un matériel qui se fixe sur l'axe.

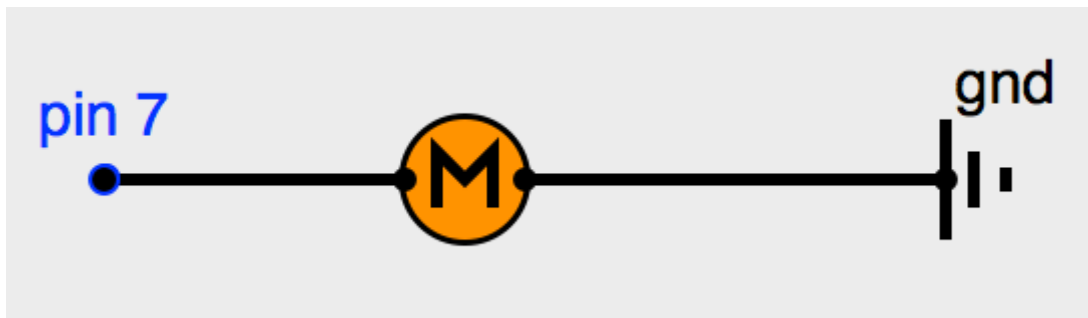
Vous pouvez dans un premier temps connecter directement votre moteur sur le +5V et le ground de l'Arduino comme ceci :



Si tout se passe bien (le moteur correspond au voltage et tout les contacts sont bons) vous devriez voir votre moteur tourner. Bien, maintenant, si vous inversez les fils, il devrait tourner dans l'autre sens ! En effet, un moteur est un dipôle non polarisé, c'est-à-dire qu'il n'a pas besoin d'être connecté d'une façon particulière pour fonctionner (il suffit de brancher une borne au + et l'autre au ground).

(Lukas, cessez avec ce moteur CC !)

Bien maintenant, comme pour les servo-moteurs on se dit qu'il suffit de brancher le moteur sur un pin digital (0 à 13) et sur le ground et de commander le courant par programmation, comme sur le schéma suivant :



Avec le programme qui va bien :

```
int pinMoteur=7;

void setup(){

    pinMode(pinMoteur,OUTPUT);

    digitalWrite(pinMoteur,HIGH);

}

void loop(){

    //vide car inutile pour l'exemple

}
```

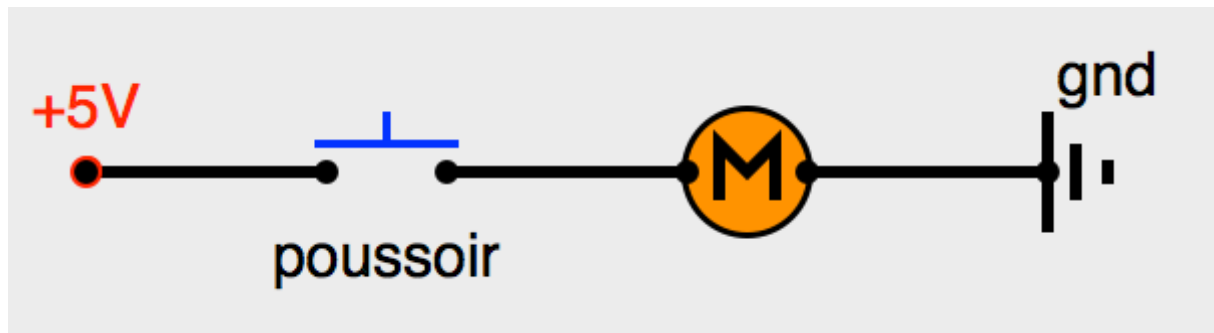
Et bien il ne faut pas le faire !

Tout d'abord parce qu'un moteur ça envoie des tensions de façon peu contrôlable quand ça tourne (on appelle ça des parasites) et lorsqu'il s'arrête, il continue d'envoyer du courant (dans l'autre sens vers l'Arduino) qui peut endommager votre carte. Enfin, le moteur a de fortes chances de ne pas démarrer, on dit qu'il ne "décolle" pas. En effet, l'Arduino fournit à ses bornes digitales un courant trop faible pour que le moteur fonctionne. Et on ne peut pas augmenter ce courant !

Donc on retient qu'on ne connecte pas un moteur CC directement à une borne digitale de l'Arduino.

Alors comment faire ?

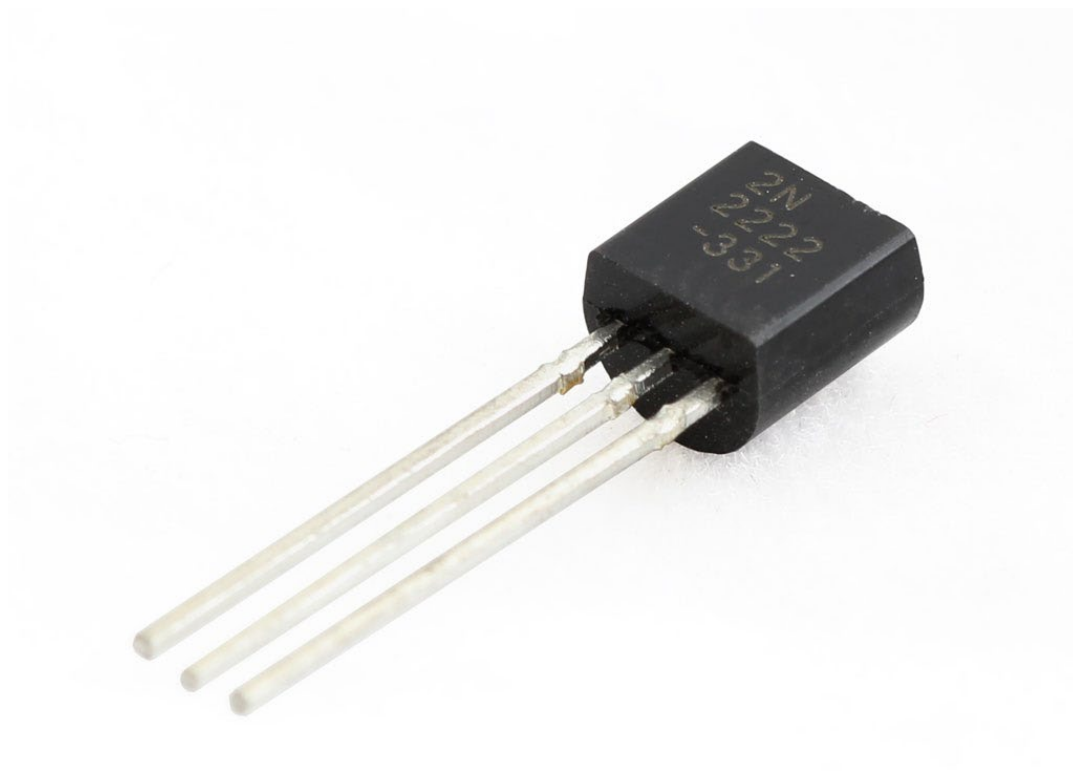
Il vous faut pouvoir commander l'arrivée d'un courant plus fort. Voici un schéma avec un bouton poussoir :



On voit bien pour le coup que si l'on appuie, le moteur reçoit le courant (ici directement du +5V de l'Arduino) et démarre. Si on relâche, le moteur s'arrête. Mais tout ceci est commandé par votre cerveau qui commande votre doigt, pas par programmation automatique... Il nous faut donc une sorte d'interrupteur que l'on peut commander par programmation. Et bien ça existe ! En fait, il existe plusieurs interrupteurs que l'on peut commander par programmation. Nous en verrons deux dans ce chapitre : le **transistor** et le **MOSFET** (qui est un transistor un peu différent). Dans mon cours de perfectionnement à Arduino, vous découvrirez un autre interrupteur de ce type : le **relai**.

Le transistor bipolaire

Alors je vous arrête tout de suite, il ne s'agit pas d'écouter la radio dans le froid (je dis ça pour les anciens dont le mot transistor parle un peu). Un transistor bipolaire est un composant électronique. C'est un semi-conducteur (il ne laisse pas passer l'électricité dans n'importe quel sens) qui a été inventé en 1947 et ne ressemblait à l'époque pas du tout à son apparence actuelle. Voici justement un transistor bipolaire de notre époque :



Un transistor NPN



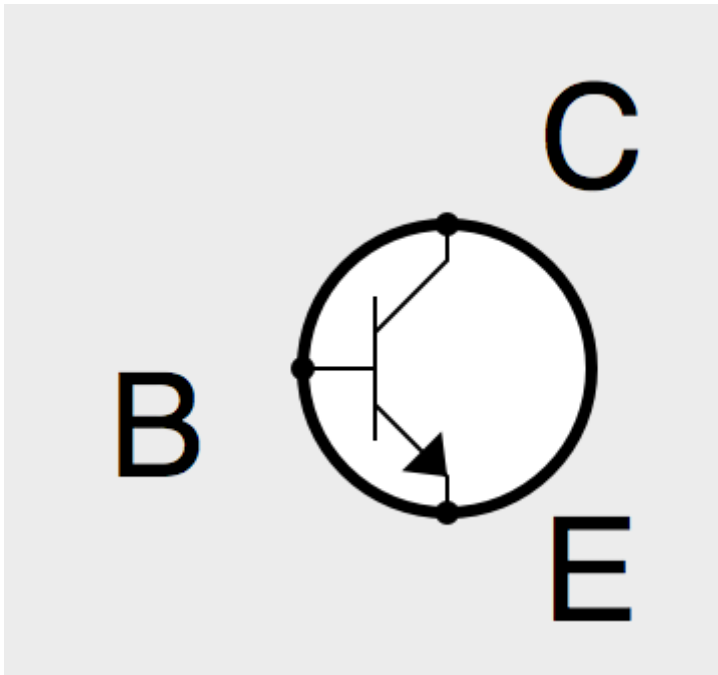
- Comme amplificateur de signal électrique (ce qui ne nous concerne pas),
- Comme interrupteur dans un circuit (ce qui nous intéresse !).

Il existe deux principales familles de transistor bipolaires : les transistors NPN et les transistors PNP.

Je n'entre pas dans les détails de fonctionnement à l'intérieur de ce composant, mais il faut savoir que les transistors NPN sont bien plus couramment utilisés que les PNP. C'est d'ailleurs ce type de transistor que nous allons utiliser dans ce cours.

Pour différencier un NPN d'un PNP, il n'y a que la référence et la datasheet.

Bon, il faut savoir tout de suite qu'un transistor, c'est fragile. Il faut donc le protéger. Mais avant tout voyons comment il se connecte. Voici le symbole d'un transistor NPN :



Symbole électrique d'un transistor NPN

Il faut expliquer plusieurs choses :

- Le B représente la base : c'est ici que l'on va commander le transistor.
- Le C représente le collecteur : C'est cette borne que l'on va relier au moteur.
- Le E représente l'émetteur : Cette borne va être reliée au ground.
- Pour la connexion, il faut se référer absolument à la datasheet pour repérer les pattes.



Lorsque l'on va envoyer du courant dans la base (B) le transistor va devenir conducteur entre le collecteur (C) et l'émetteur (E) ce qui fermera le circuit et permettra au moteur de tourner.

Vous savez déjà (si Lukas, on en déjà parlé pendant que vous répariez votre stylo 4 couleurs...) que le moteur envoie des parasites et des tensions dangereuses pour l'Arduino en

particulier lorsqu'il s'arrête. Là il ne s'agit plus de l'Arduino (car le transistor l'isole du danger) mais du transistor, qui risque de griller. Il faut donc empêcher le retour de courants néfastes. Nous allons utiliser une diode qui va empêcher un retour de courant lorsque le moteur continue de tourner. Car rappelez-vous, la diode ne laisse passer le courant que dans un sens. Du coup le transistor (qui sera en mode bloqué) enverra le courant vers la diode (qui ne l'autorisera que dans un sens). Ce courant viendra en opposition du courant induit (qui est dû à l'inertie du moteur qui s'arrête) et du coup viendra freiner le moteur.

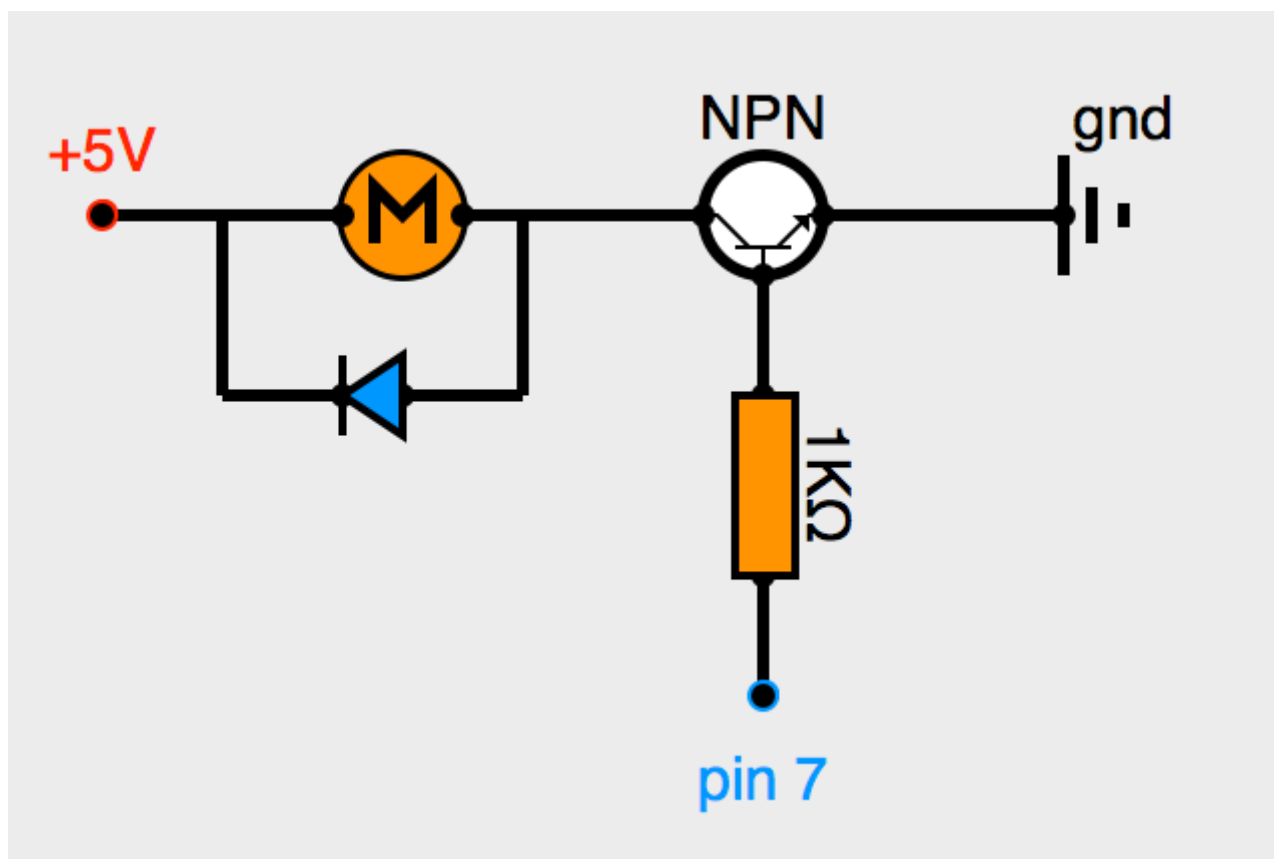
On y gagne ! D'une on protège le transistor, de deux on freine le moteur ! Cette diode est appelée **diode de roue libre**.

Quelle diode on met ?

On pourrait mettre une LED. Mais la LED n'est pas assez rapide pour passer de l'état passant à l'état bloquant. Il existe une autre sorte de diode (qui n'éclaire pas) : **la diode Schottky**, qui elle est très rapide pour passer à l'état bloquant.

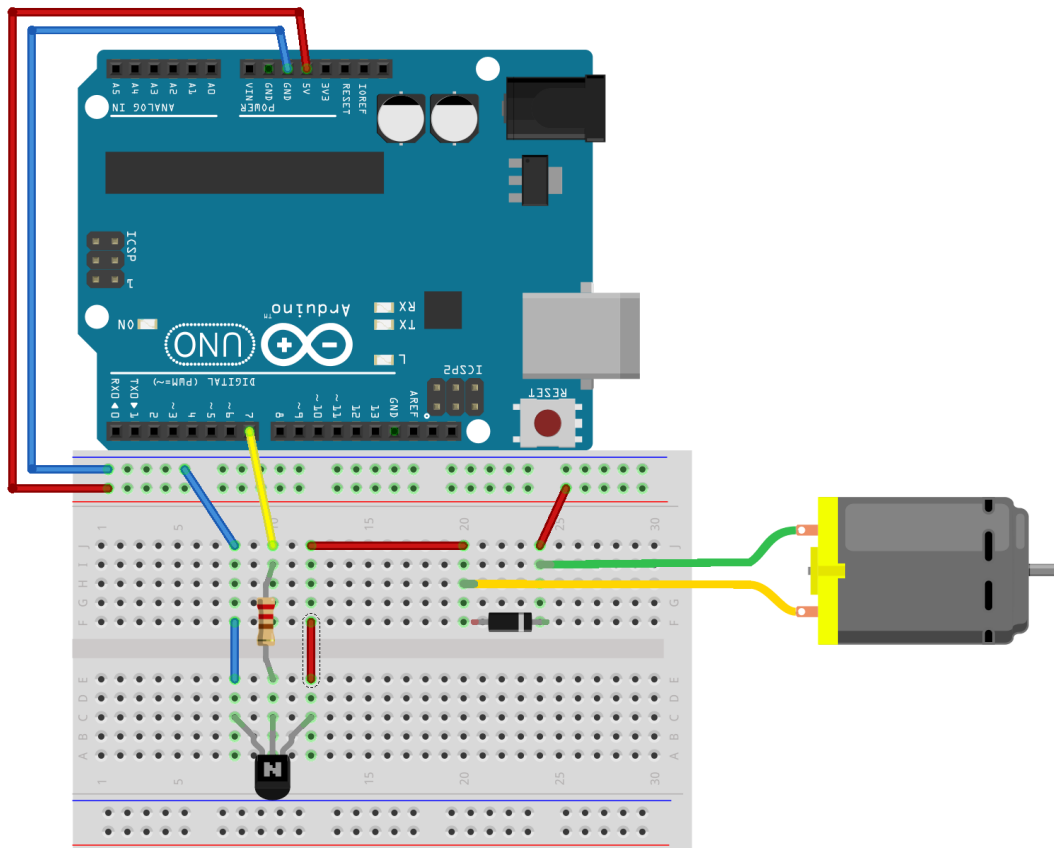
Il faut aussi prévoir une résistance entre l'Arduino et la base du transistor. Elle permet de protéger ce dernier si la tension est trop importante (en effet, il peut chauffer et griller).

Du coup on obtient le montage qui suit :



Il est aussi possible d'ajouter un condensateur aux bornes du moteur pour absorber les pics de tension (parasites) et protéger la diode et le transistor.

Voici le montage en représentation sur la breadboard :



fritzing



Bon vous pouvez tester votre montage en utilisant le code suivant (qui ne devrait vous poser aucun problème) :

```
int pinMoteur=7;

void setup(){

    pinMode(pinMoteur,OUTPUT);

}

void loop(){

    digitalWrite(pinMoteur,HIGH); //le moteur se lance

    delay(1000);

    digitalWrite(pinMoteur,LOW); //le moteur s'arrête

    delay(1000);
```


}

Si vous changez la valeur de la résistance, vous verrez que le moteur tourne plus rapidement. En fait la tension du courant qui va passer dans le transistor quand il ferme le circuit, dépend en partie de la tension d'entrée sur sa base. Donc la valeur de cette résistance peut jouer sur le résultat attendu.

Bon et bien ce n'est pas fini ! Le transistor bipolaire est une solution, mais ce n'est pas la meilleure. En effet, il se peut que ce dernier ne passe pas en saturation (c'est-à-dire qu'il ne joue pas son rôle d'interrupteur) sans raison réelle. En fait les raisons sont électriques, mais le résultat paraît aléatoire. En gros, on ne peut pas toujours compter sur lui de façon fiable.

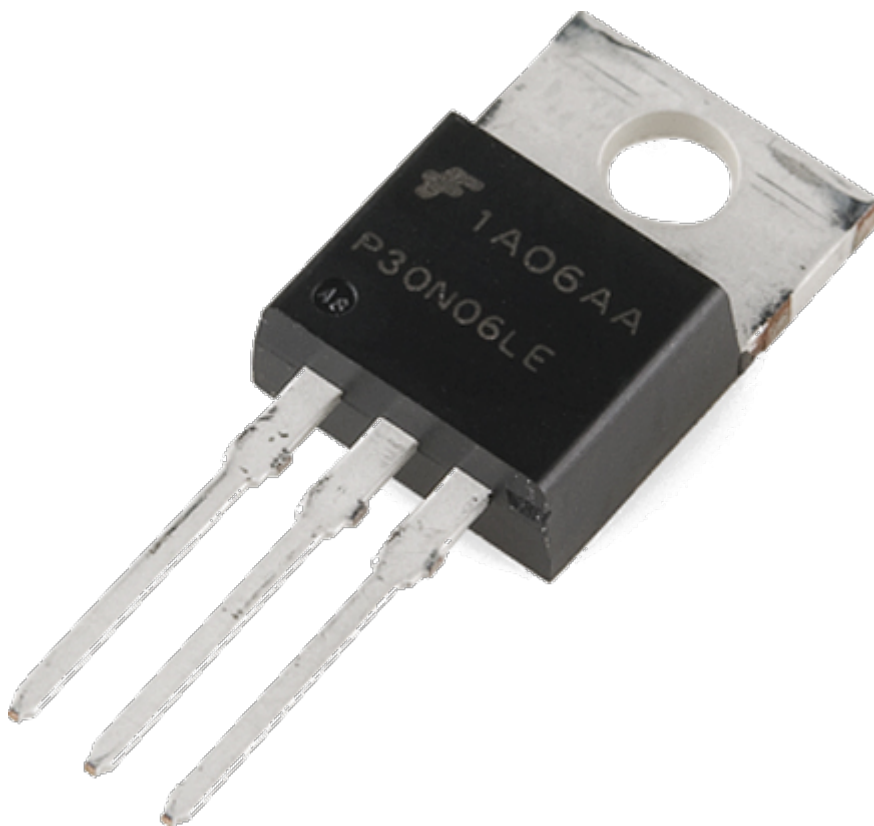


Il existe donc un autre type de transistor : les transistors à effet de champs, dits **transistors MOSFET** (pour *Metal Oxide Semiconductor Field Effect Transistor*).

Le transistor à effet de champs ou MOSFET

En fait, le MOSFET est bien plus indiqué pour jouer le rôle d'interrupteur que le transistor bipolaire. Si l'on veut grossièrement expliquer la différence entre l'un et l'autre, le transistor bipolaire est un amplificateur de tension et le MOSFET un amplificateur de courant.

Voici de quoi il a l'air :

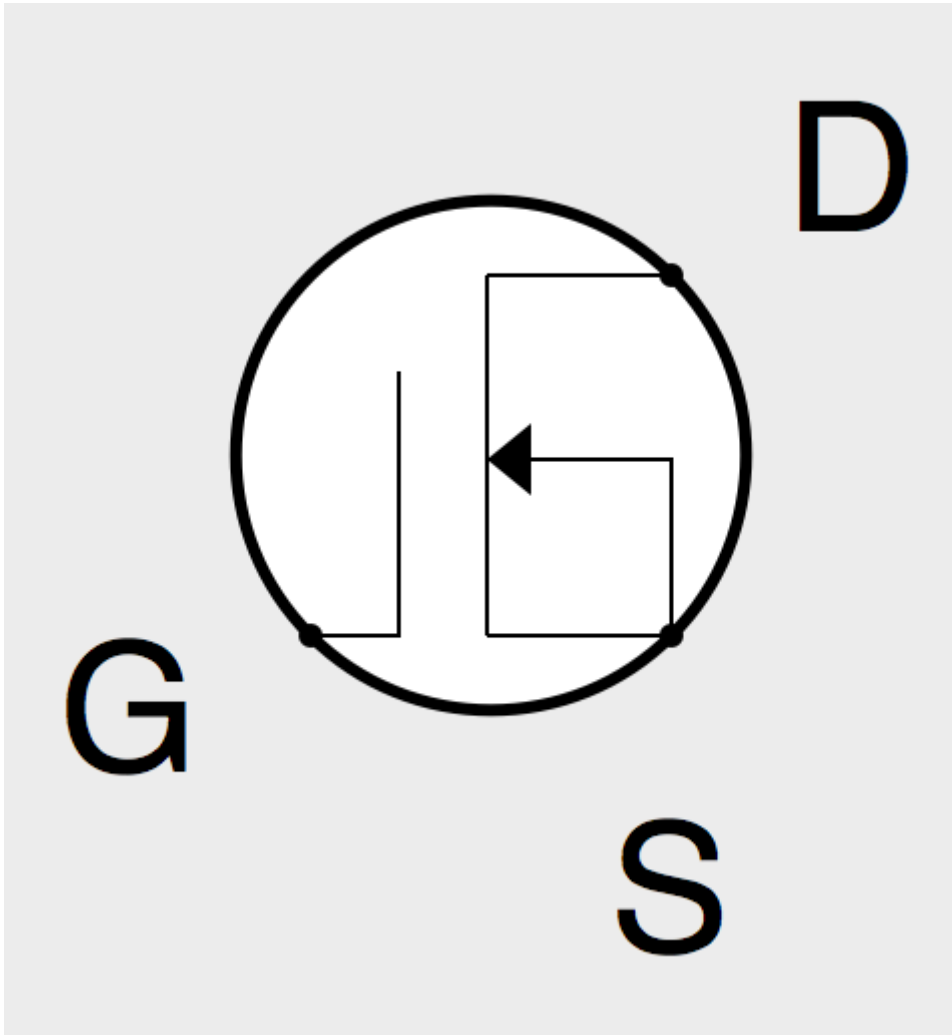


Il suffit donc au MOSFET de recevoir une très faible tension à sa patte de commande pour déclencher le passage du courant. Il est plus stable, il chauffe moins (le métal en haut sert de

dissipateur de chaleur), mais il est plus cher (de l'ordre de 10 fois) que le bipolaire. Je vous propose d'utiliser un MOSFET IRF540N.

Je ne m'étendrai pas sur son fonctionnement interne, mais il faut savoir qu'il en existe aussi deux sortes : le canal N et le canal P.

Voici le symbole d'un MOSFET canal N :



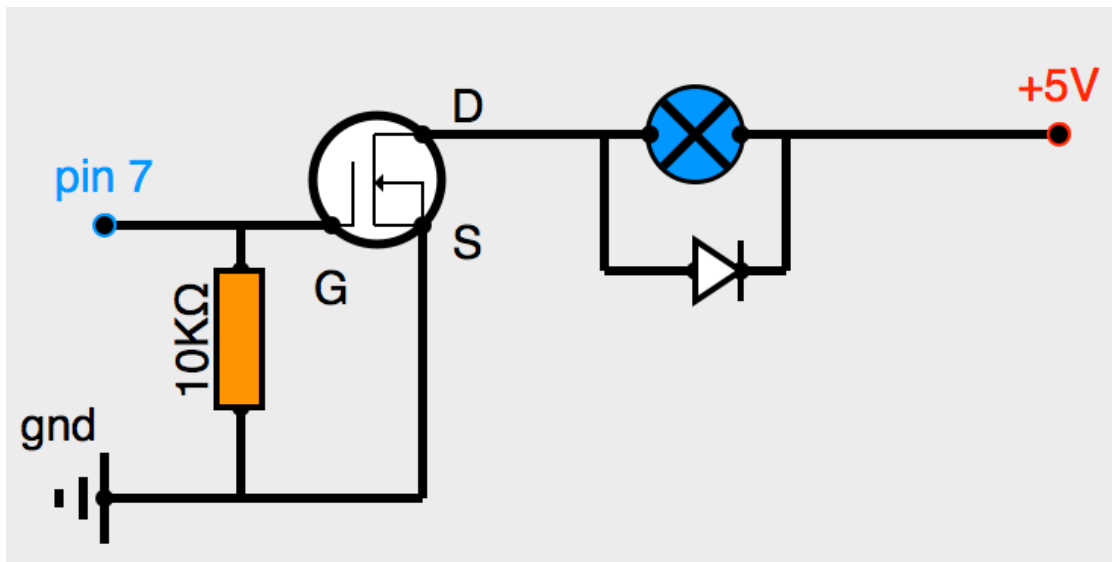
MOSFET canal N

Explications :

- G représente la borne Gate : c'est ici que l'on commande le transistor.
- S représente la borne Source : pour un canal N on la connecte au ground.
- D représente la borne Drain : pour un canal N on la connecte au moteur.
- Il faut se référer à la datasheet du MOSFET pour le connecter correctement !

Ne confondez pas la répartition des pattes dans un MOSFET et dans un transistor bipolaire. Il faut à chaque fois vérifier la datasheet du composant que vous choisissez.

Voici le schéma de montage d'un MOSFET canal N. On dit souvent qu'il coupe le ground car il commande l'accès au ground par sa source. Le drain étant côté charge (ici le moteur) :



La résistance de 10 K Ω sert de résistance PULL-DOWN afin que le signal envoyé au MOSFET soit clair (+5V ou 0V).

Le programme associé est identique à celui que l'on a utilisé pour commander le moteur CC avec un transistor bipolaire.

```
int pinMoteur=7;

void setup(){

    pinMode(pinMoteur,OUTPUT);

}

void loop(){

    digitalWrite(pinMoteur,HIGH); //le moteur se lance

    delay(1000);

    digitalWrite(pinMoteur,LOW); // le moteur s'arrête

    delay(1000);

}
```

Pour l'utilisation des transistors bipolaires PNP ou les MOSFETs canal P, il faut savoir que les connexions sont différentes et le pilotage inversé (c'est-à-dire que pour lancer le moteur il faut mettre la commande sur LOW et non HIGH). Je ne vous propose pas d'explications plus précises pour l'utilisation de ces deux modèles pour deux raisons :

- La première est que dans la grande majorité des cas (sinon tous) vous utiliserez ceux que j'ai présentés précédemment car ce sont souvent les montages proposés dans les ouvrages ou sur les sites. Pour des raisons à la fois de fiabilité et d'efficacité.

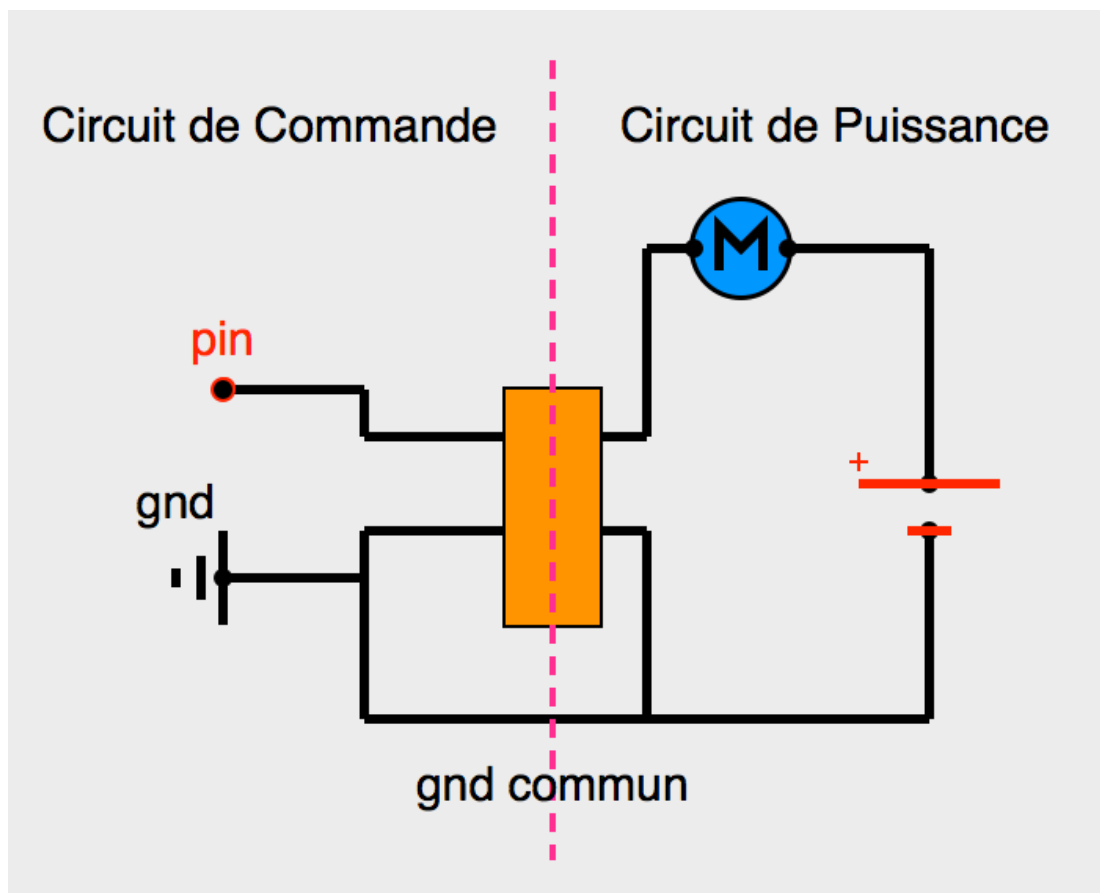
- La seconde est abordée dans le [chapitre suivant](#). Vous allez voir qu'il existe des composants très pratiques pour réaliser le pilotage de moteurs CC.

Nous avons vu jusqu'à l'utilisation de moteurs alimentés par la carte Arduino : l'énergie électrique provenait du +5V de notre carte. En fait cette façon de procéder est pratique pour des raisons de connexions et de complexité liées à la gestion de plusieurs sources électriques dans un robot. Mais très vite il va falloir utiliser une énergie séparée pour l'Arduino et les moteurs. Ce principe est maintenant assez clairement utilisé en électricité : c'est la séparation du **circuit de commande** et du **circuit de puissance**. Je m'explique :

- Le circuit de commande envoie des informations binaires (0 ou 1, donc LOW ou HIGH) à une tension qui est celle de l'Arduino (+5 V) et un ampérage souvent faible (de l'ordre de 40 mA). L'alimentation de ce circuit est réalisée par le câble USB lorsque l'Arduino est connecté à votre ordinateur, ou par une batterie (pile 9 V) ou un transformateur adapté. C'est un circuit qui doit être protégé des parasites (sauts de tension) pour fonctionner au mieux et sans risque de dommages.
- Le circuit de puissance fournit l'énergie nécessaire pour faire fonctionner la charge (ce qui est branché) comme dans notre cas un moteur. Ça pourrait être une lampe, un écran ou autre appareil consommant bien plus que +5 V et 200 mA.

Séparez votre circuit de commande de votre circuit de puissance

Le principe est à l'image du schéma suivant :

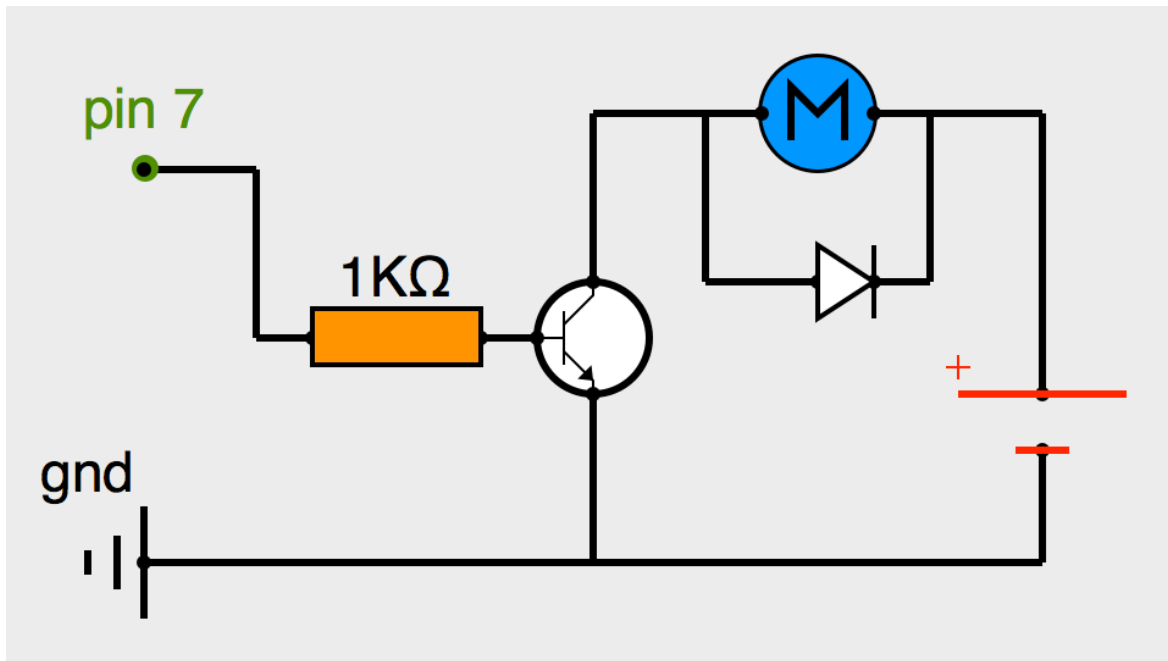


Vous noterez que dans le cas de pilotages de moteurs par des transistors, il est important que le ground soit commun (c'est-à-dire que le - de la pile ou du générateur doit être connecté aussi au gnd de l'Arduino). Ce ne sera plus le cas lorsque nous aborderons le pilotage en

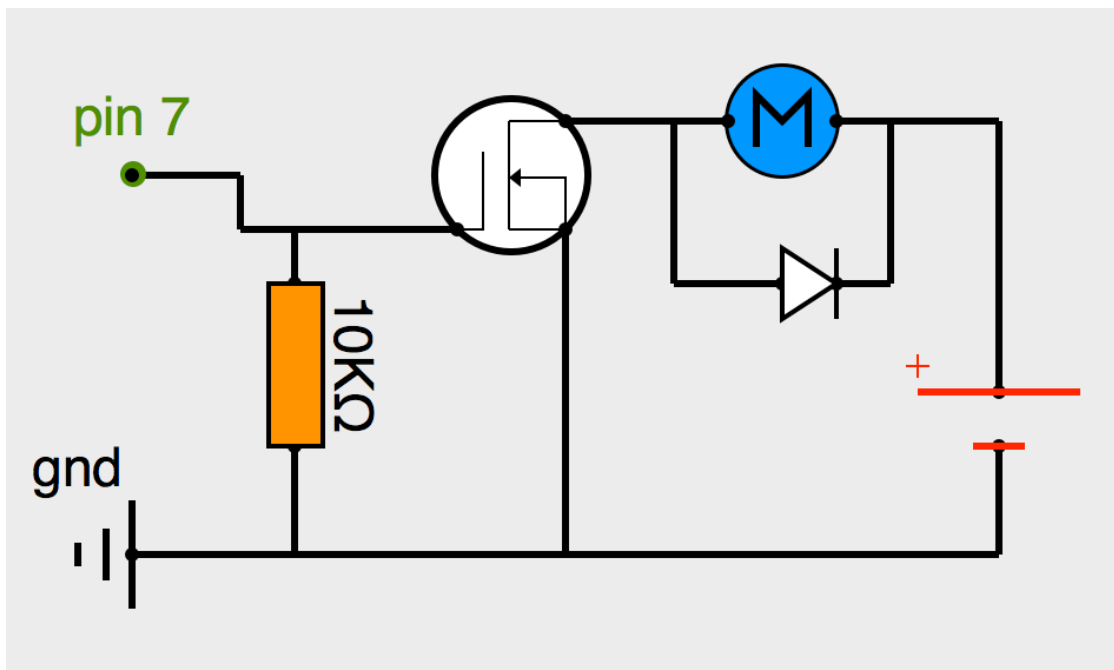
domotique (dans mon cours de perfectionnement), qui nécessite une sécurité accrue liée au risques graves d'électrocution avec des tension importantes.

Avec ce principe de circuit de commande et circuit de puissance, on peut donc piloter des moteurs 12 V, ou 24 V sans risque d'endommagement de la carte Arduino.

Voici donc le schéma de connexion modifié avec l'utilisation d'un transistor bipolaire NPN :



Et maintenant le schéma qui utilise un MOFSET canal N :



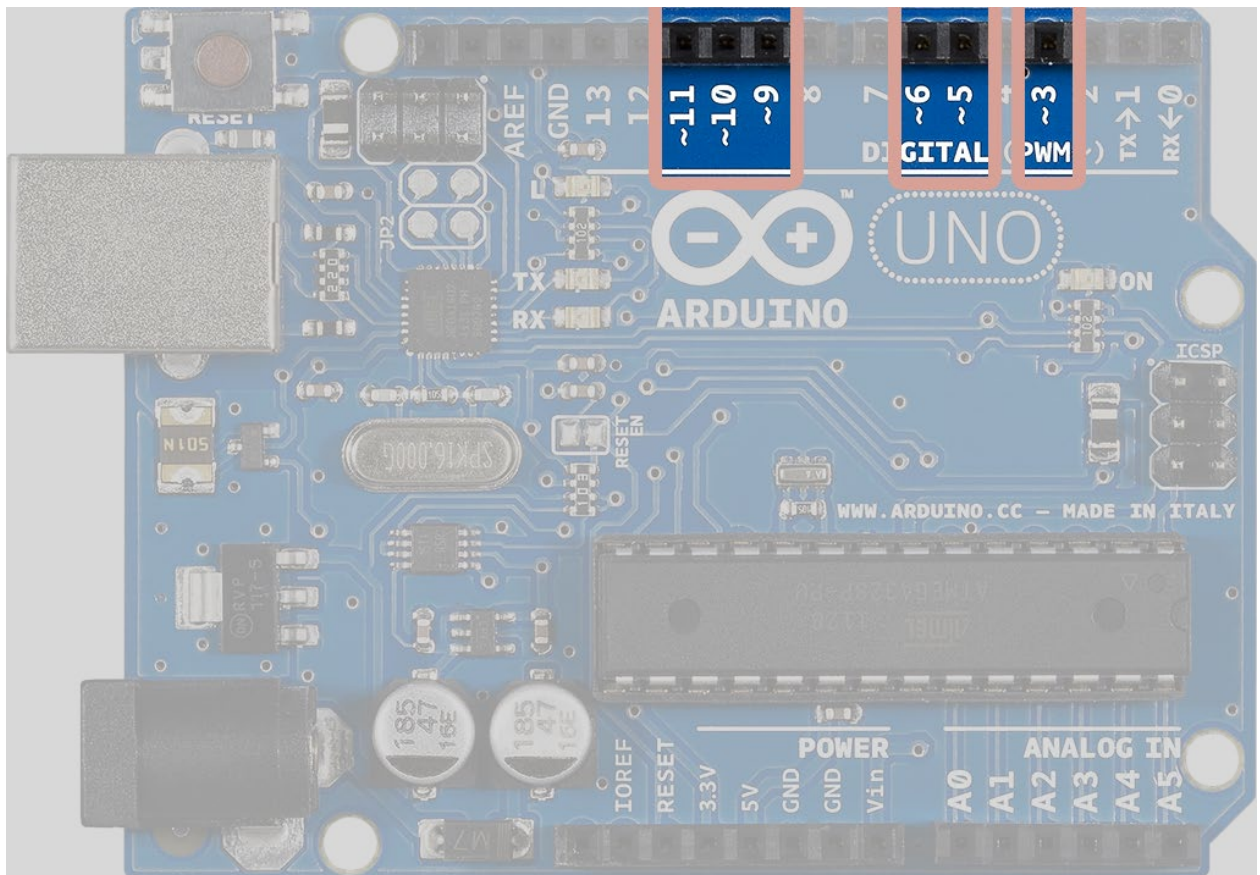
Bien. Vous avez vu comment lancer et arrêter un moteur CC, et comment séparer le circuit de commande du circuit de puissance. Il vous reste une dernière chose à apprendre pour ce chapitre...

Faites varier la vitesse du moteur

Si vous avez conçu un robot de type R2D2 (deux roues motrices et une roue libre à l'arrière), votre robot peut avancer et tourner (si, si, nous verrons ce point dans mon cours d'approfondissement !), mais il ne pourra pas faire varier sa vitesse. En effet, soit le moteur est lancé à pleine puissance, soit il est arrêté (l'inertie d'arrêt n'est pas prise en compte).

Et bien sachez qu'il est possible d'agir sur la vitesse d'un moteur !

C'est même assez simple, à condition d'utiliser certains des pins de sorties digitales de l'Arduino : ceux qui sont notés PWM. Voyez plutôt (non Lukas, rien à voir avec le Pluto de Disney™) :



On les reconnaît car elles ont le symbole " ~ " à côté de leur numéro.

Quelle différence entre ces pins (3, 5, 6, 9, 10 et 11) et les autres ?

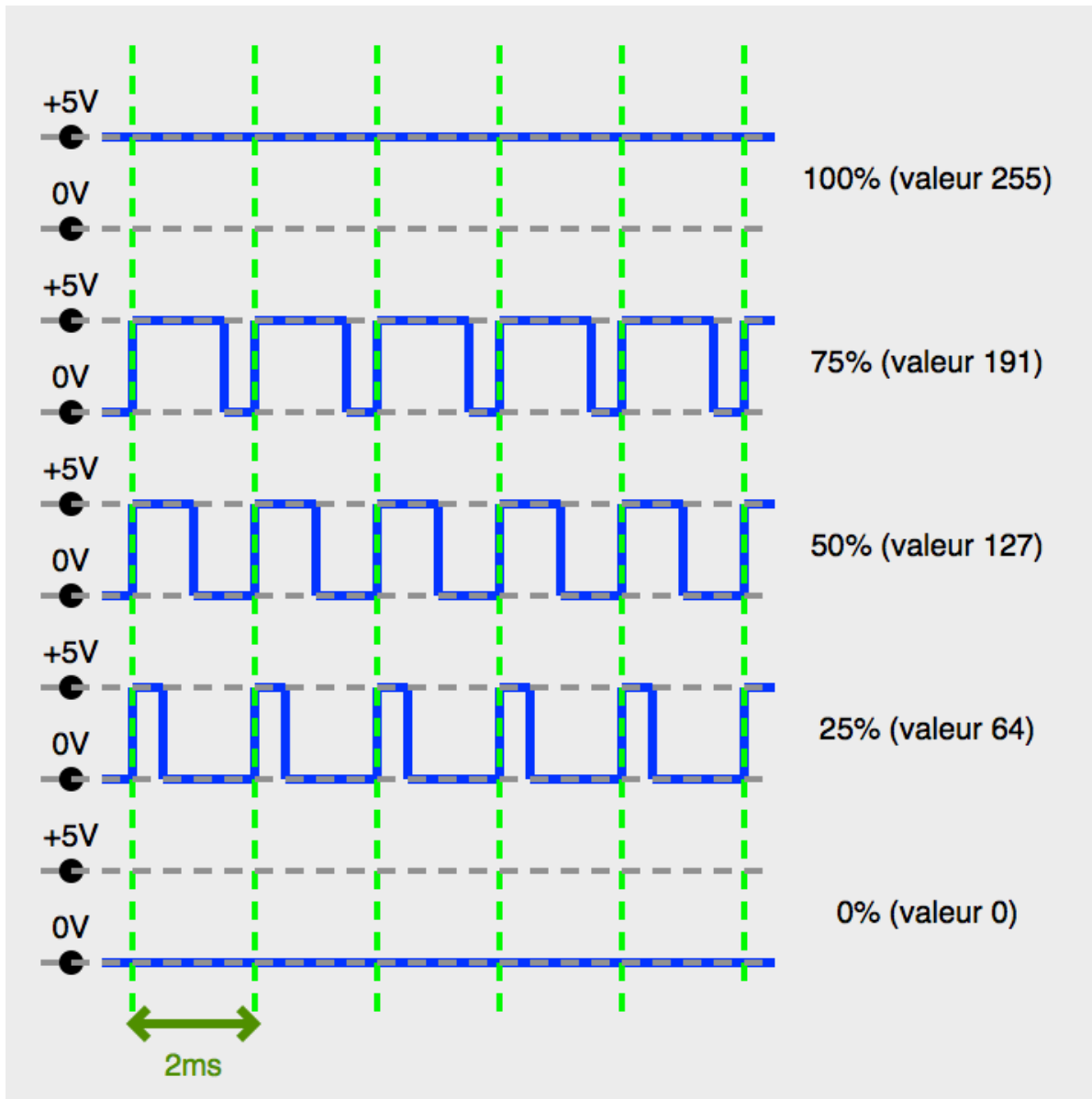
Et bien ils sont capables, tout comme les autres, d'envoyer soit un +5 V, soit du 0 V en mode OUTPUT. Mais ils ont un autre avantage, ils peuvent envoyer un [train d'impulsions](#) variable. On les commande avec une fonction de l'Arduino qui va prendre une valeur entre 0 et 255 :

```
analogWrite(pin,valeur);
```

où `pin` est le pin PWM choisi et mis en mode OUTPUT, et `valeur` est la valeur comprise entre 0 et 255 qui va définir la part de temps pendant laquelle le pin sera à l'état HAUT durant chaque intervalle (255 correspondra à 100% du temps d'intervalle à l'état HAUT)

Le temps de cycle du début d'une impulsion à une autre est de 2 ms. On appelle ce temps la **période du cycle** des impulsions.

Toutes les 2 ms, le pin va rester à l'état HAUT pendant un pourcentage du temps et à l'état BAS pour le reste des 2 ms.



Vous pouvez observer que la valeur varie entre 0 et 255 et que le pourcentage de HAUT par rapport à BAS évolue en fonction. Toutes les valeurs entre 0 et 255 sont utilisables. C'est comme si on utilisait la fonction de mappage :

```
valeur=map(pourcentage,0,100,0,255);
```

Du coup pour le moteur, comment ça se passe ?

On pourrait imaginer un moteur saccadé (il s'arrête, il repart, il s'arrête, il repart...). En fait pas du tout. Rappelez-vous que le moteur a une inertie mécanique (il ne s'arrête pas d'un coup mais continue de tourner même sans électricité) donc le résultat est un peu comme si on le relançait par petits coups. Le cycle des impulsions est sur 2 ms. Donc on le relance plus ou moins sur ce temps très rapide.

On qualifie de **rapport cyclique** le temps à l'état haut (Th) divisé par la période du cycle (T) ce qui donne :

$RC = Th/T$. Ce rapport est situé entre 0 et 1.

Plus la valeur du rapport cyclique est proche de 1, plus le temps de relance est long, plus le moteur va tourner vite. À l'inverse, moins on le relance (rapport cyclique proche de 0), plus il a tendance à ralentir. Donc il va tourner moins vite. À la valeur 0, il s'arrête (avec une inertie).



Voici donc un programme qui fait varier la vitesse du moteur avec les valeurs 0%, 25%, 50%, 75% et 100% pendant 3 secondes chacune :

```
int pinMoteur=5; //pin de commande relié au transistor

void setup() {

  pinMode(pinMoteur,OUTPUT); // pin de commande en mode sortie

}

void loop() {

  for (int pourcentage=0;pourcentage<=100;pourcentage+=25){//boucle de variation en %

    int valeur=map(pourcentage,0,100,0,255); //conversion en valeur

    analogWrite(pinMoteur,valeur); // pin de commande en mode impulsion

    delay (3000); // attente de 3 secondes

  }

}
```

Il est possible, si vous n'avez pas utilisé un circuit de puissance différent du circuit de commande (donc une pile pour le moteur différente de l'alimentation de l'Arduino) que votre moteur ne décolle pas à 25% de la puissance.

Vous noterez l'utilisation de `<=` dans la boucle `for` qui permet de bien atteindre la valeur 100.

Bien nous allons clore ici la partie 1 du chapitre sur l'utilisation d'un moteur CC.

Le chapitre qui suit va vous apporter de nouvelles connaissances sur ce moteur CC, qui s'avèreront bien utiles pour la suite !

Le moteur à courant continu (partie 2) : le pont en H et les circuits intégrés

Dans le chapitre précédent, vous avez appris à connecter un moteur en protégeant à la fois l'Arduino et les transistors de commande. Vous avez aussi vu comment gérer la vitesse du moteur grâce aux pins PWM.

En revanche votre moteur, pour le moment, ne peut tourner que dans un sens.

Et bien l'objet de ce chapitre est de vous montrer comment on peut faire tourner un moteur dans les deux sens (en continuant bien sûr de gérer sa vitesse) grâce au **pont en H**.

Puis vous verrez comment réaliser la même chose plus facilement grâce aux circuits intégrés qu'ont développés des électroniciens et qui sont maintenant disponibles en commerce.

Enfin, nous construirons un programme qui permet de faire réagir les moteurs de deux ventilateurs en fonction de l'environnement (l'un piloté par bouton poussoir, l'autre piloté par potentiomètre).

Le pont en H

Vous avez remarqué que votre moteur peut maintenant tourner plus ou moins vite grâce aux montages précédents. En revanche, il ne tourne que dans un sens.

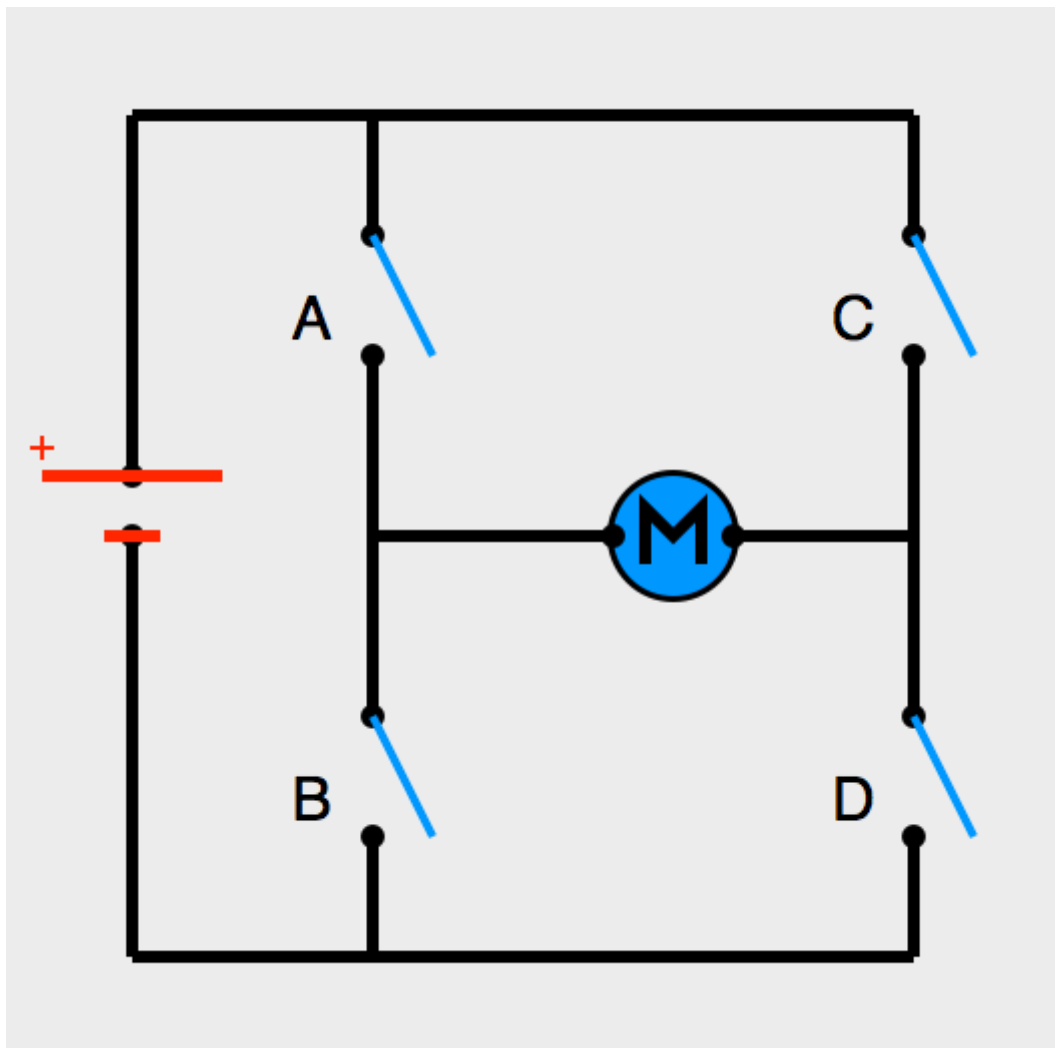
Forcément, si vous le connectez à l'envers, il tournera dans l'autre sens. Mais ce n'est quand même pas pratique de refaire les branchements à chaque fois !

Il existe, en réfléchissant un peu, une méthode qui peut résoudre notre problème : elle consisterait à faire changer l'électricité de sens à volonté !

Mais comment !?

Ha ! Toujours cette soif d'apprendre qui nous fait avancer dans le blizzard de l'ignorance, l'oeil humide et le nez gelé, délaissant famille et patrie pour de nouveaux horizons aux paysages fabuleux et dont... ? Oui, Lukas, vous avez raison je m'égare !

Voici donc un petit schéma qui vous donne un début de réponse :



Le pont en H

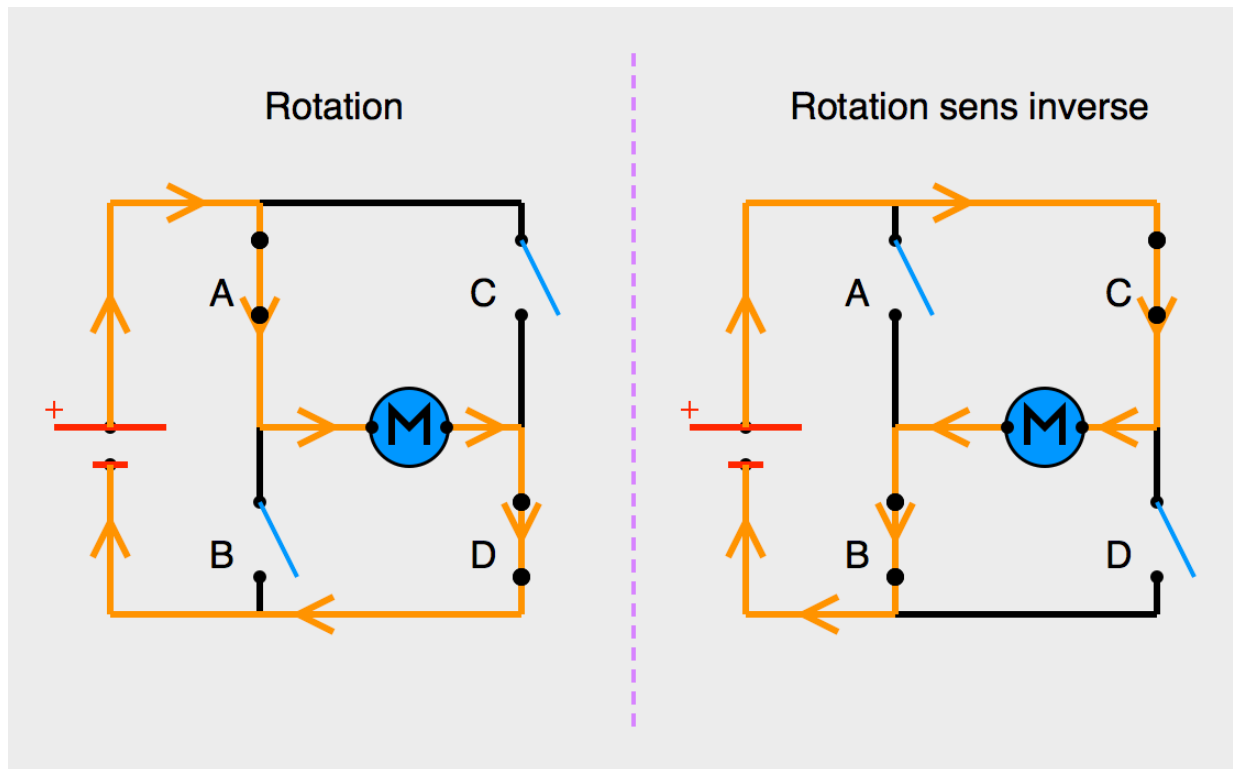
Voici ce qu'on appelle le pont en H. Il tient son nom de la forme en H du circuit autour du moteur.

Ce circuit (ici réalisé avec des interrupteurs) permet de résoudre notre problème. Il faut tout de même être attentif à certaines erreurs possibles.

Le principe

Les interrupteurs fonctionnent deux par deux. Le A est associé au D et le B est associé au C. Dans le schéma ci-dessus, rien ne se passe car tous les interrupteurs sont ouverts (ils ne laissent pas passer le courant). Le moteur est arrêté.

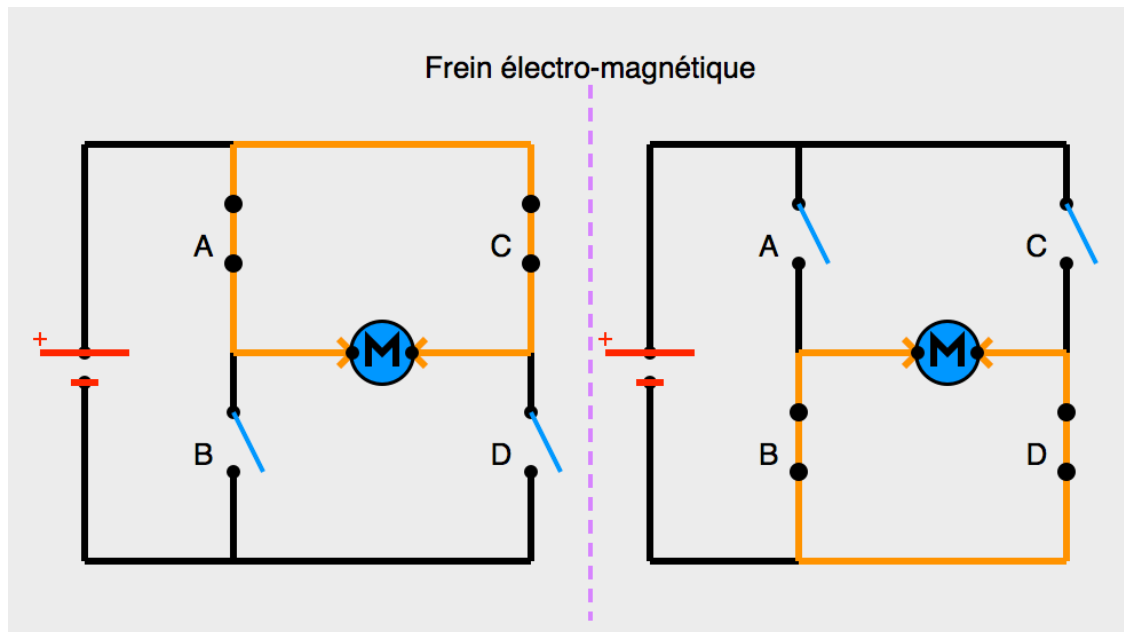
Voyons maintenant ce qui arrive lorsqu'on actionne en même temps les interrupteurs A et D (schéma de gauche), ou les interrupteurs B et C (schéma de droite) :



Vous voyez donc :

- Sur le schéma de gauche : les interrupteurs A et D sont fermés. Le courant entre par la patte gauche du moteur et sort par sa droite. Le moteur tourne.
- Sur le schéma de droite : les interrupteurs B et C sont fermés. Le courant entre par la patte droite du moteur et sort par sa gauche. Le moteur tourne donc dans le sens inverse !

On peut aussi associer le A au C et le B au D. Dans le [chapitre précédent](#), nous avons vu que lorsqu'un moteur est en roue libre (c'est-à-dire qu'il tourne à cause de sa force d'inertie mais pas à cause du courant), il génère un courant. Ce courant peut être utilisé dans le pont en H comme frein électro-magnétique. Le moteur s'envoie son propre courant à l'envers ! Ça permet de contrôler l'arrêt du moteur plutôt que de le laisser en roue libre.



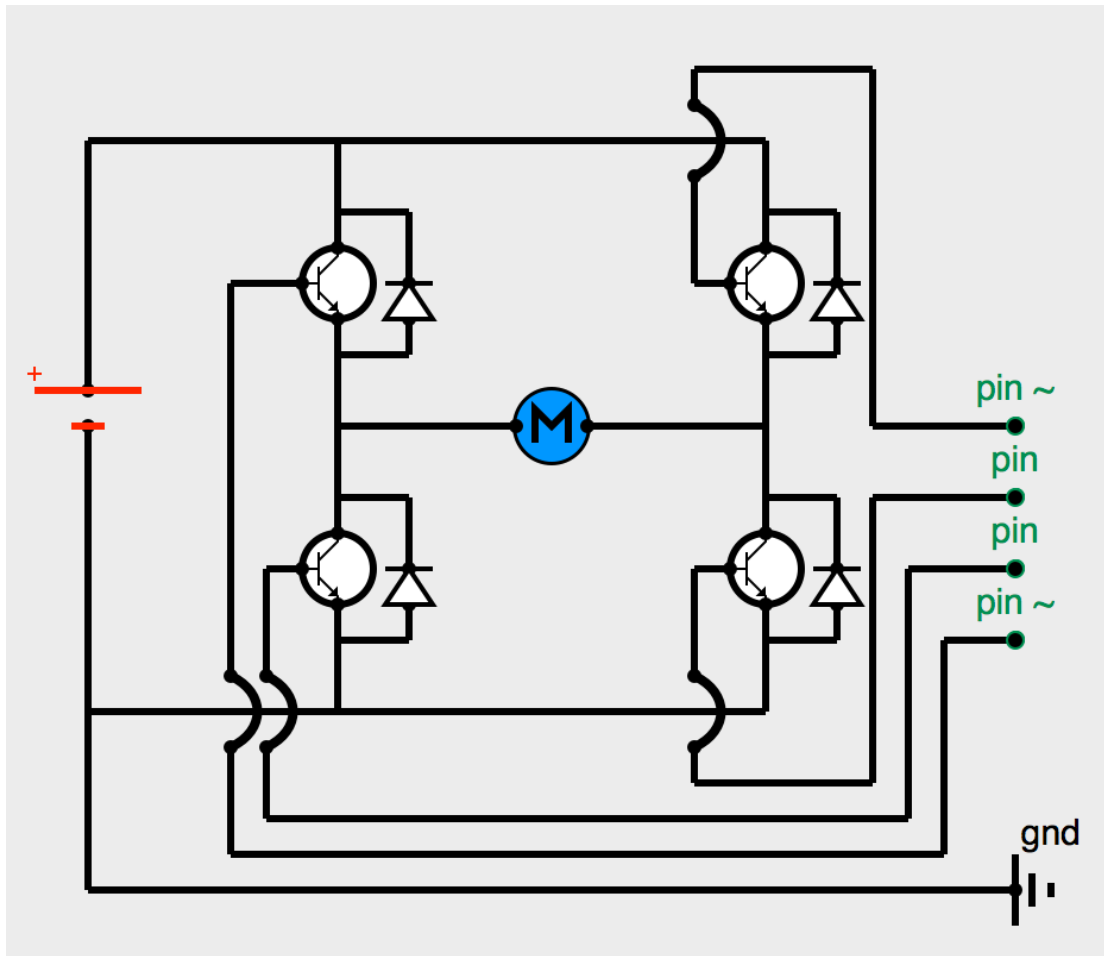
Frein électro-magnétique créé par l'inertie du moteur

Si vous fermez tous les interrupteurs, ou A et B, ou C et D, vous réalisez un court-circuit. Dans le cas d'une pile, elle se décharge extrêmement vite. Dans le cas de composants électroniques ou d'une carte Arduino, ils grillent. Du coup, vous l'aurez compris, évitez de fermer tous les interrupteurs !

Alors pour une fois, je suis d'accord avec vous Lukas, c'est un peu fastidieux (oui, ce n'est pas le mot que vous avez employé, je sais, mais j'édulcore) de commander un moteur, et par extension, un robot, avec des interrupteurs !

Heureusement, vous avez suivi le chapitre précédent et vous savez maintenant que l'Arduino peut piloter des transistors qui servent d'interrupteurs pour le circuit de puissance !

Mais comme nous l'avons vu précédemment, il faut les protéger des retours de tensions ! Voici donc un schéma, avec les diodes placées au bon endroit :



Vous remarquerez que seuls deux pins sont en PWM. Ce n'est pas obligatoire, mais ça en libère pour d'autres besoins éventuels. Il faut aussi penser aux résistances entre l'Arduino et les transistors (pas mises sur le schéma). Je ne vous livre pas le circuit avec des MOSFETs (c'est le même principe) et je ne vous livre pas non plus le code de commande des transistors.

Mais pourquoi est-il si méchant ?!

Tout simplement parce que la suite va nous faciliter la vie, et ce n'est rien de le dire !

Les circuits double ponts en H

Sachez que vous n'êtes pas les premiers à vous dire que ça fait quand même beaucoup de connexions pour commander un seul moteur CC au final. Or vous allez voir que les moteurs CC sont tout de même très utilisés en micro-robotique, en particulier pour les déplacements. Du coup, pour nous faciliter la vie, une nouvelle puce a vu le jour dans le merveilleux monde de l'électronique.

Alors pour lever les doutes éventuels, une puce électronique est un circuit intégré (et vice-versa), c'est-à-dire une sorte de petite boîte qui contient des composants miniaturisés et connectés entre eux afin de répondre à une fonction particulière. Vous l'avez peut-être remarqué, mais les transistors se ressemblent et pourtant ce ne sont pas tous les mêmes. Et bien une puce peut très bien ressembler à une autre, mais seule sa référence et sa datasheet vont pouvoir vous dire quel est son rôle et comment elle se connecte !

Dans notre cas, il existe des puces qui servent de pont en H. C'est leur fonction. Nous allons même en découvrir une qui permet de répondre à notre attente : la L293D.

Alors le nom des circuits intégrés n'est pas toujours clair : souvent des lettres et des chiffres. Parfois se tromper d'une lettre vous donne un autre composant. C'est le cas de cette puce. Il existe la L293 et la L293D. Le D signifie diodes. Ce sont les diodes diodes roues libre qui protègent les transistors. La différence entre la L293 et la L293D est donc simple : la L293 ne contient pas de diodes roues libres.

Nous utiliserons la L293D car cela nous évite de régler nous-mêmes le problème des diodes.

Voici donc la [datasheet](#)

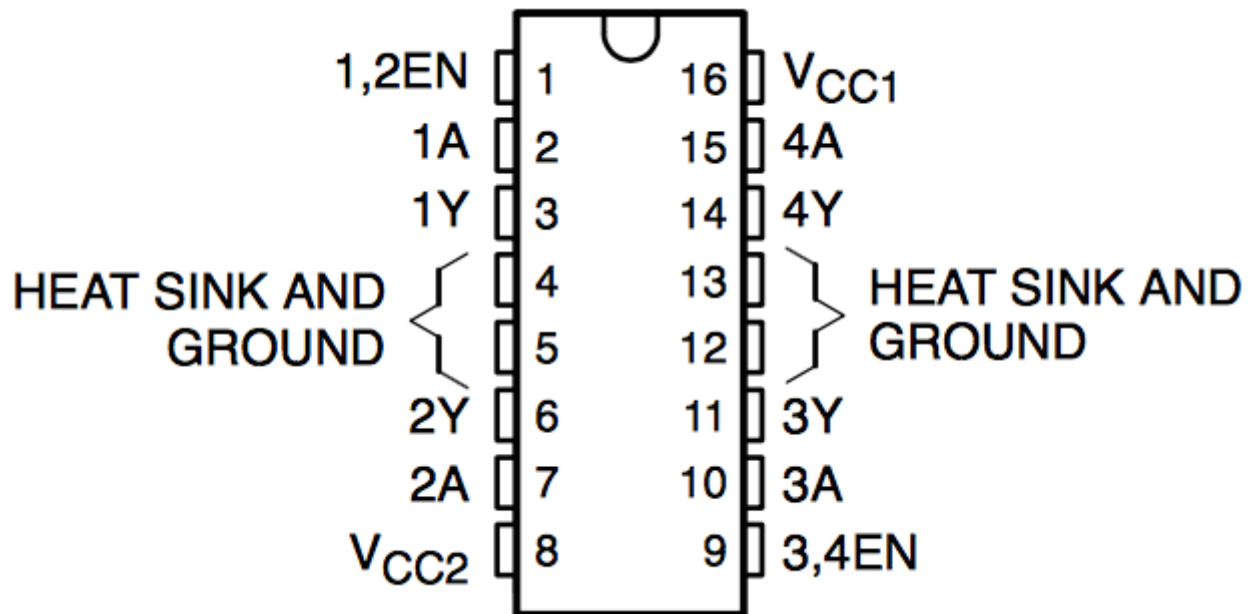


Le document s'appelle "quadruple half-H driver" qu'on peut traduire par "pilote de quadruple demi-pont en H". Donc double pont en H. Alors pourquoi ce nom de quadruple ? Tout simplement parce qu'avec ce circuit, vous pouvez piloter un moteur dans un sens à l'aide d'un demi-pont, soit 4 moteurs en sens unique, ou 2 moteurs dans les deux sens.

Voici une photo de la bête :



Et le schéma pour les connexions :



Vous remarquerez qu'un circuit comporte toujours une marque pour son orientation. Les pins de connexion étant numérotés, c'est toujours bon de savoir par où commencer.

Vous allez voir que c'est finalement assez simple car chaque partie est regroupée.

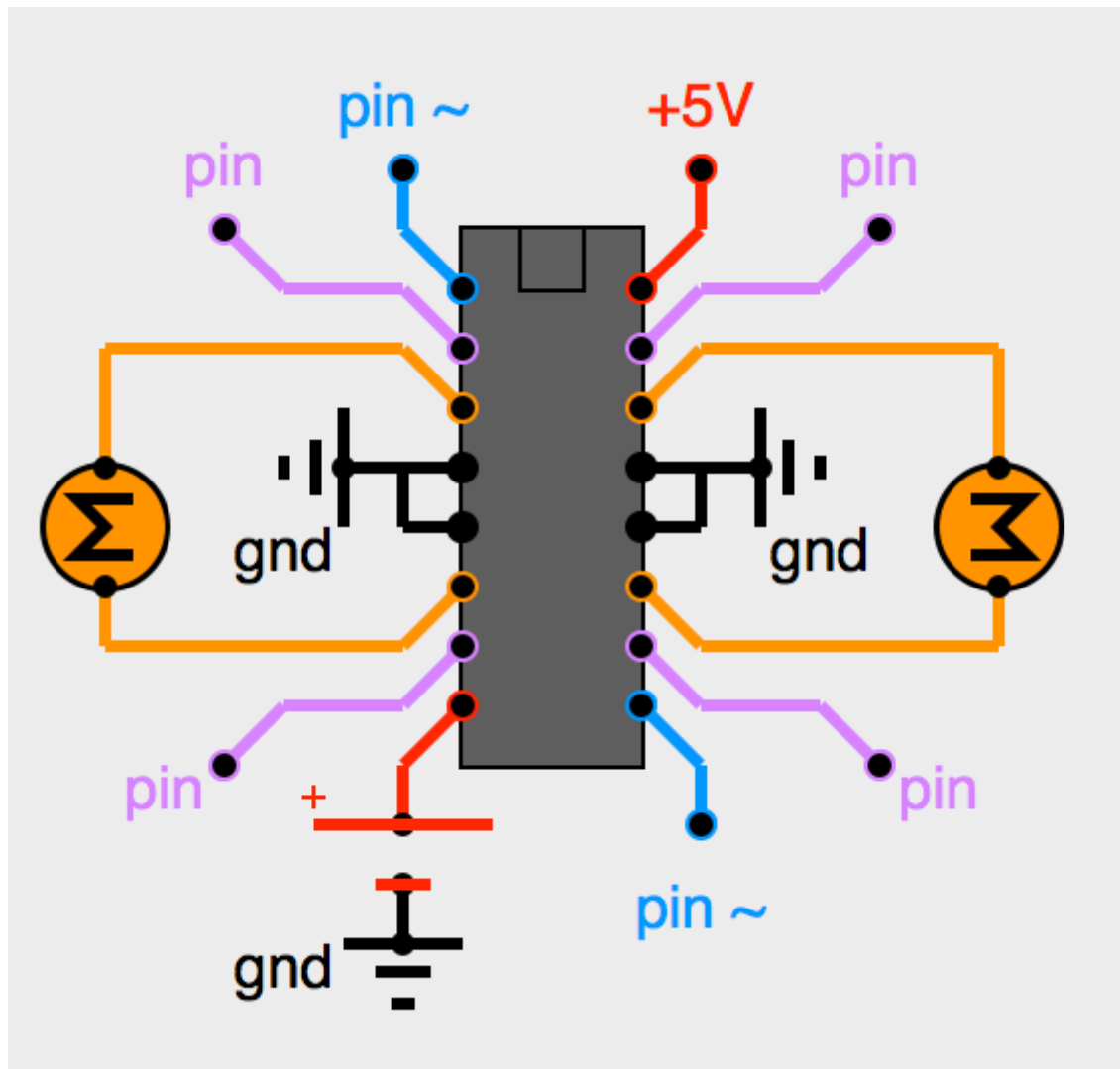
La partie gauche commande le premier pont (demi-pont 1 et demi-pont 2) :

- pin 1 : active la partie gauche si l'on y envoie un état haut (+5V dans notre cas) et la désactive à l'état bas. On peut donc l'utiliser pour envoyer un signal PWM.
- pin 2 : c'est le pin de commande du demi-pont 1 (là où on envoie le courant au transistor pour qu'il laisse ou non passer le courant).
- pin 3 : on branche une patte du moteur ici.
- pin 4 : c'est le gnd (et le radiateur, ou dissipateur de chaleur). on y branche l'autre patte d'un moteur si on l'utilise que dans un sens. Il faut le relier au gnd de l'Arduino.
- pin 5 : gnd pour le demi-pont 2 (il est d'ailleurs conseillé de relier tous les gnd utilisés ensemble).
- pin 6 : on connecte l'autre patte du moteur (si utilisation dans les deux sens).
- pin 7 : commande du demi-pont 2.
- pin 8 : c'est ici que l'on connecte la source d'alimentation des moteurs (circuit de puissance).

La partie droite commande le second pont (demi-pont 3 et 4) :

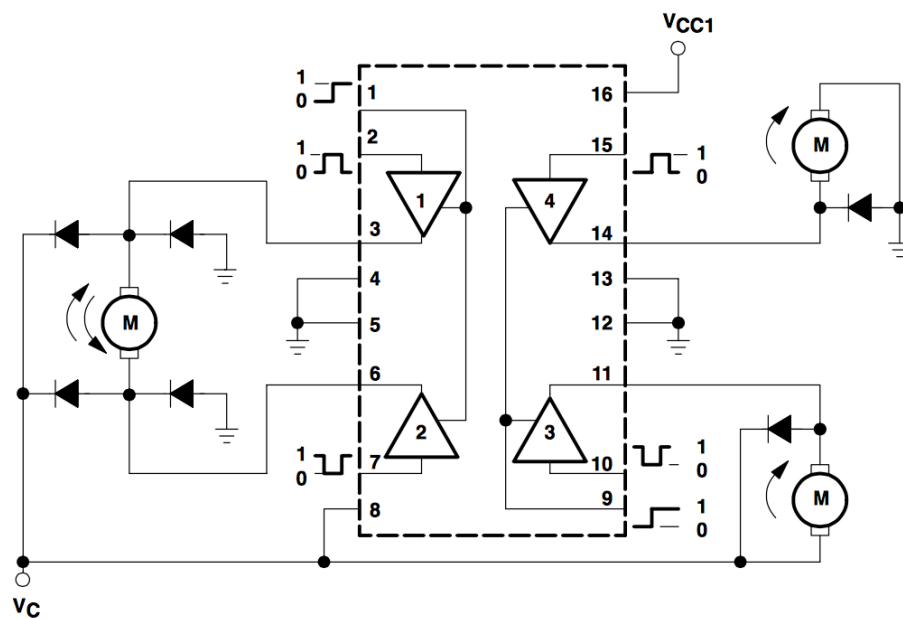
- pin 16 : c'est ici qu'on connecte le +5V de référence (circuit de commande).
- pin 15 : commande du demi-pont 4.
- pin 14 : connexion d'une patte du second moteur.
- pin 13 et 12 : gnd.
- pin 11 : connexion de l'autre patte.
- pin 10 : commande du demi-pont 3.
- pin 9 : active la partie droite (donc demi-pont 3 et 4). Possible en PWM.

Bon voici un autre schéma en couleurs qui permet de mieux visualiser :



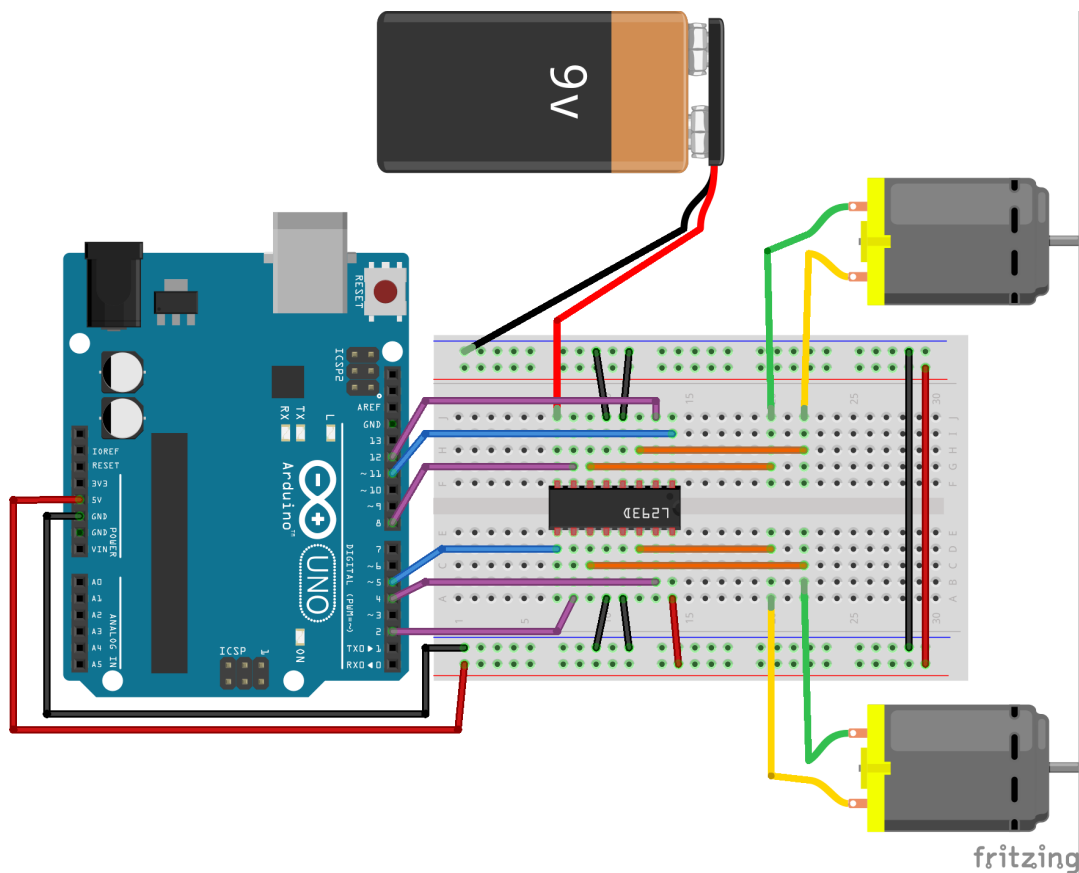
Vous noterez, sur le diagramme de la datasheet qui suit, qu'il est possible de connecter soit un moteur dans les deux sens, soit deux moteurs en sens unique.

block diagram



Les diodes placées autour des moteurs ne sont pas indispensables.

Voici enfin un exemple de connexion sur une breadboard avec une pile 9V et une carte Arduino :



Ce montage est assez compliqué. Repérez bien les différents câbles, c'est important.

Les commandes du moteur du haut se font avec les pins 12 et 8. La commande PWM est sur le pin 11.

Les commandes du moteur du bas se font avec les pins 2 et 4. La commande PWM est sur le pin 5.

Il existe un circuit intégré permettant de gérer un plus grand ampérage et qui fonctionne sur le même principe : Le L298. Si vous comprenez le fonctionnement du L293D, le L298 ne vous posera aucun problème.

Bien, voyons maintenant la programmation avec cette petite puce de rêve...

Programmez deux moteurs CC avec un circuit intégré L293D

Maintenant que vous avez réussi à connecter vos fils, vos moteurs, votre L293D, votre pile et votre Arduino (oui vous pouvez prendre une pause Lukas, vous avez l'air épuisé) nous allons voir comment piloter le tout.

Si l'on suit l'[image précédente](#), vous allez pouvoir piloter en avant et/ou en arrière deux moteurs, à des vitesses variables. En robotique, ce n'est pas rien !

Je vous ai dit plus haut que les moteurs se pilotent par moitié de L293D. Il faut donc repérer le mieux possible quel pin commande quoi et dans quel moteur ! Le plus simple est de donner un nom de variable significatif aux pins attachés. Par exemple :

```
int pin1Moteur1=12; //cmd 1 du moteur 1

int pin2Moteur1=8; // cmd 2 du moteur 1

int pinPMoteur1=11;// PMM du moteur 1

int pin1Moteur2=2;// cmd 1 du moteur 2

int pin2Moteur2=4;// cmd 2 du moteur 2

int pinPMoteur2=5;// PWM du moteur 2
```

L'avantage des variables c'est que si vous décidez d'attacher un autre pin à la variable, ça ne change pas le programme.

Voyons maintenant un tableau qui montre le comportement d'un moteur CC (Moteur1 ou Moteur2) en fonction de l'état de ses pins :

pin1Moteur	pin2Moteur	Comportement du moteur
HIGH	LOW	Le moteur tourne dans un sens
LOW	HIGH	Le moteur tourne dans l'autre sens
LOW	LOW	Frein électromagnétique
HIGH	HIGH	Frein électromagnétique

Ce tableau est donc valable pour chaque moteur. Il suffit que les pins soient en mode OUTPUT et qu'on les active avec la commande :

```
digitalWrite(pin,etat);
```

avec `etat` qui prend la valeur 1 (HIGH) ou 0 (LOW).

En ce qui concerne la gestion du PWM, je vous rappelle qu'on utilise la commande :

```
analogWrite(pin,valeur);
```

avec `valeur` compris entre 0 (stop) et 255 (à fond).

Enfin, je vous conseille d'utiliser des fonctions afin de mieux organiser votre programme. En effet, en fonction de ce que vous voulez obtenir, la succession de commandes risque de devenir lourde dans la `loop()` .

Vous ne voyez pas comment faire ? Regardons ça ensemble dans la section suivante !

Silence... Moteurs CC... Ça tourne !

Voici un programme qui va gérer chaque moteur grâce à une fonction que l'on va appeler `actionMoteur`. Cette fonction va prendre 3 arguments :

- Le moteur concerné : moteur 1 ou 2.
- Le sens de rotation : 1 ou -1 (sens contraire). Tout autre nombre entier stoppera le moteur.
- Le pourcentage de puissance : entre 0% et 100%.

Cette fonction `actionMoteur` va ensuite analyser ces arguments pour envoyer les bonnes informations aux pins rattachés au L293D.

La communication avec le moniteur série n'est pas obligatoire, mais elle permet de bien vérifier le fonctionnement du programme. Et voici ce programme de gestion des moteurs à partir de la fonction `actionMoteur` :

```
int pin1Moteur1=12; //pin de commande moteur 1

int pin2Moteur1=8; // pin de commande moteur 1

int pinPMoteur1=11;// pin PWM moteur 1

int pin1Moteur2=2; // pin de commande moteur 2

int pin2Moteur2=4; // pin de commande moteur 2

int pinPMoteur2=5; // pin PWM moteur 2


void setup() {

    // put your setup code here, to run once:

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

```

pinMode(pin1Moteur1,OUTPUT);

pinMode(pin2Moteur1,OUTPUT);

pinMode(pinPMoteur1,OUTPUT);

pinMode(pin1Moteur2,OUTPUT);

pinMode(pin2Moteur2,OUTPUT);

pinMode(pinPMoteur2,OUTPUT);
}

void loop() {

    actionMoteur(1,1,100); //moteur 1 100% puissance sens 1

    actionMoteur(2,1,100); //moteur 2 100% puissance sens 1

    delay(5000); //attente de 5 secondes

    actionMoteur(1,1,50); //moteur 1 50% puissance sens 1

    actionMoteur(2,-1,50); //moteur 2 50% puissance sens -1

    delay(2000);

    actionMoteur(1,0,0); //arrêt moteur 1

    actionMoteur(2,0,0); //arrêt moteur 2

    delay(1000);
}

void actionMoteur(int moteur,int sens,int pourcentage){

    int pin1,etat1,pin2,etat2,pinP,puissance; //variable de la fonction

    //test numéro du moteur

    if (moteur==1){

```

```
pin1=pin1Moteur1;

pin2=pin2Moteur1;

pinP=pinPMoteur1;

}

else {

pin1=pin1Moteur2;

pin2=pin2Moteur2;

pinP=pinPMoteur2;

}

//test sens du moteur 1,-1 (sens contrainre) ou tout autre valeur (stoppe le moteur)

if (sens==1){

etat1=1;

etat2=0;

}

else if (sens== -1){

etat1=0;

etat2=1;

}

else {

etat1=0;

etat2=0;

}

puissance=map(pourcentage,0,100,0,255);

analogWrite(pinP,puissance);
```

```

digitalWrite(pin1,etat1);

digitalWrite(pin2,etat2);

//affichage sur le moniteur série (facultatif)

Serial.print("Moteur : ");

Serial.print(moteur);

if (sens== -1 || sens==1){

    Serial.print(" sens : ");

    Serial.print(sens);

}

else {

    Serial.print(" ! stop ! ");

}

Serial.print(" puissance : ");

Serial.println(pourcentage);

}

```

La ligne `if (sens== -1 || sens==1)` signifie : si ("sens" est égal à -1 **ou bien** si "sens est égal à 1"). Le signe `||` en programmation est un opérateur booléen.

Vous pouvez donc tester plusieurs conditions d'un coup grâce à ces opérateurs booléens :

- `||` : veut dire "ou" (*or* en anglais). On obtient le `|` sur PC en tapant `[Alt Gr] + [6]` et sur Mac en tapant `[⇧] + [alt] + [L]`.
- `!` : veut dire "non/pas" (*not* en anglais).
- `&&` : veut dire "et" (*and* en anglais).



Moteurs et interaction

Nous le verrons dans le chapitre sur le robot autonome, mais il est souvent utile de commander des moteurs en fonctions de tests réalisés par des capteurs d'environnement. Pour le moment, en terme de capteurs, vous connaissez le bouton poussoir (qui est soit baissé, soit

levé) et le potentiomètre (dont on peut récupérer la position de rotation avec une valeur entre 0 et 1 024).

Que pouvons-nous imaginer avec ce matériel ?

Et bien tout simplement un ventilateur. En fait non, deux ventilateurs. L'un dont la vitesse sera commandée par un bouton poussoir, l'autre dont la vitesse sera commandée par le potentiomètre.

En terme de connexions, nous gardons celles du L293D vues au-dessus :

- Moteur 1 piloté par les pins 12, 8 et 11 pour le PWM ;
- Moteur 2 piloté par les pins 2, 4 et 5 pour le PWM.

Le potentiomètre sera relié au pin analogique A0.

Le bouton poussoir sera relié au pin 7 de l'Arduino en mode INPUT_PULLUP

Pour ce programme, vous n'êtes pas obligés de fabriquer un vrai ventilateur avec vos moteurs. Mais c'est une bonne expérience, car la partie mécanique et fabrication occupe une part importante dans la réalisation robotique.

L'objectif du programme reprend des programmes réalisés précédemment, avec quelques subtilités.

- Le premier moteur doit être commandé avec le bouton poussoir :
 - début à l'arrêt,
 - un appui = 25% de la vitesse,
 - un autre appui = 50% de la vitesse,
 - un autre appui = 75% de la vitesse,
 - un autre appui=100% de la vitesse,
 - un autre appui = arrêt du ventilateur.
- Le second moteur doit être commandé avec le potentiomètre :
 - potentiomètre au centre = moteur arrêté,
 - potentiomètre vers la droite = moteur accélère proportionnellement dans un sens,
 - potentiomètre vers la gauche = moteur accélère proportionnellement dans l'autre sens.

Vous pouvez créer de nouvelles fonctions pour répondre à vos besoins et réussir cet exercice. Comme d'habitude, il n'y a pas qu'une façon de procéder.

La difficulté va se situer sur la lecture de la position du potentiomètre et sa transformation en sens et vitesse.

Un affichage sur le moniteur série peut vous aider à vérifier votre programme.

Le moniteur série est à la fois un outil de communication avec l'Arduino, et un bon moyen pour déboguer un programme (lecture des valeurs, état des pins...). Les lignes utilisées comme test lors de la conception, peuvent ensuite être mises en commentaires ou effacées pour le programme finalisé.

Un exemple de fabrication du ventilateur...

Si vous n'êtes pas encore à l'aise avec la fabrication de robots ou d'objets, vous pouvez commencer simplement. Dans le cas des ventilateurs, vous pouvez créer les pâles avec des rectangles découpés dans du carton à chaussure.

Vous prenez ensuite un bouchon de liège (type bouteille de vin) que vous entaillez au cutter de la largeur de la pâle (et en biais). Vous y glissez les pâles en carton (3 suffisent).

Vous piquez ensuite ce montage sur l'axe du moteur (avec un point de colle forte) et ça suffira pour le moment.

Surélevez votre moteur en le mettant au bord d'une table (ou en haut d'une pile de livres) et fixez-le (avec du scotch).

Évidemment, cette fabrication est très précaire et n'aura pas une durée de vie très longue. Mais c'est un bon début si vous n'avez pas d'idées pour fabriquer votre ventilateur !

Retenez qu'il n'y a pas UNE bonne façon de faire, c'est ce qui fait toute la richesse de la communauté des utilisateurs de l'Arduino : leur créativité ! Chacun trouvera, en fonction de ses projets, des idées pour concevoir ses maquettes ou ses plateformes. Et on y arrive rarement du premier coup !

J'aborde dans le cours de perfectionnement les différents matériaux utilisables.



Un exemple de programme final...

Voici le code que je propose. Le vôtre sera sûrement différent, car il existe bien des façons de répondre au cahier des charges !

```
/*  
  
 * Nanomaitre 2015  
  
 * Programme de pilotage de deux moteurs, en direction et puissance  
  
 * moteur 1 piloté par un bouton poussoir  
  
 * sens unique, valeurs : 0, 25, 50, 75, 100 % de la puissance  
  
 * moteur 2 piloté par un potentiomètre  
  
 * double sens.  
  
 * potentiomètre au centre : arrêt  
  
 * potentiomètre à gauche sens -1  
  
 * potentiomètre à droite sens 1  
  
 * puissance en fonction de la position du potentiomètre
```



```

* par rapport à sa valeur centrale

* Affichage sur moniteur série pour contrôle.

*/

int pin1Moteur1=12; //pin de commande moteur 1

int pin2Moteur1=8; // pin de commande moteur 1

int pinPMoteur1=11;// pin PWM moteur 1

int pin1Moteur2=2; // pin de commande moteur 2

int pin2Moteur2=4; // pin de commande moteur 2

int pinPMoteur2=5; // pin PWM moteur 2

int pinBouton=7; //pin du bouton poussoir en mode INPUT_PULLUP

int pinPot=A0; //pin du potentiomètre

int vitesseM1=0; //variable de pourcentage de vitesse pour moteur 1

int sensM1=1; //variable de sens pour moteur 1

int vitesseM2=0; //variable de pourcentage de vitesse pour moteur 2

int sensM2=1; //variable de sens pour moteur 2


void setup() {

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

    pinMode(pin1Moteur1,OUTPUT);

    pinMode(pin2Moteur1,OUTPUT);

    pinMode(pinPMoteur1,OUTPUT);

    pinMode(pin1Moteur2,OUTPUT);

    pinMode(pin2Moteur2,OUTPUT);

    pinMode(pinPMoteur2,OUTPUT);

```

```

pinMode(pinBouton,INPUT_PULLUP); //pinBouton en mode INPUT_PULLUP

}

void loop() {

    //gestion du moteur 1 avec le bouton

    if (!digitalRead(pinBouton)){// test d'appui du bouton (mode inversé car INPUT_PULLUP)

        vitesseM1+=25; // on ajoute 25 à la vitesse

        if (vitesseM1>100) // si on dépasse 100

            vitesseM1=0;//on revient à 0

        delay(200); //délai pour éviter les répétitions d'appui

    }

    //gestion du moteur 2 avec le potentiomètre

    int valeur=map(analogRead(pinPot),0,1023,-100,100); // on mappe la valeur lue entre -100
et 100

    if (valeur>=-2 && valeur<=2){//on teste si le potentiomètre est au centre

        vitesseM2=0; // on met la vitesse à 0

        sensM2=0; // on actionne la commande d'arrêt

    }

    else if (valeur<-2){// on teste si la valeur est négative

        vitesseM2=-valeur; //on met à jour la vitesse (en positif)

        sensM2=-1; //on indique le sens

    }

    else if (valeur>2){// on teste si la valeur est positive

        vitesseM2=valeur; //on met à jour la vitesse

        sensM2=1; // on indique le sens

```

```

}

//appel des commandes de moteurs

actionMoteur(1,sensM1,vitesseM1); //commande pour moteur 1

actionMoteur(2,sensM2,vitesseM2); //commande pour moteur 2

Serial.println(); //saut de ligne
}

//fonction de gestion d'un moteur

void actionMoteur(int moteur,int sens,int pourcentage){

    int pin1,etat1,pin2,etat2,pinP,puissance; //variable de la fonction

    //test numéro du moteur

    if (moteur==1){

        pin1=pin1Moteur1;

        pin2=pin2Moteur1;

        pinP=pinPMoteur1;

    }

    else {

        pin1=pin1Moteur2;

        pin2=pin2Moteur2;

        pinP=pinPMoteur2;

    }

    //test sens du moteur 1,-1 (sens contrainre) ou tout autre valeur (stoppe le moteur)

    if (sens==1){

        etat1=1;

```

```
    etat2=0;

}

else if (sens== -1){

    etat1=0;

    etat2=1;

}

else {

    etat1=0;

    etat2=0;

}

puissance=map(pourcentage,0,100,0,255);

analogWrite(pinP,puissance);

digitalWrite(pin1,etat1);

digitalWrite(pin2,etat2);

//affichage sur le moniteur série (facultatif)

Serial.print("Moteur : ");

Serial.print(moteur);

if (sens== -1 || sens==1){

    Serial.print(" sens : ");

    Serial.print(sens);

}

else {

    Serial.print(" ! stop ! ");

}

}
```

```
Serial.print(" puissance : ");

Serial.print(pourcentage);

Serial.print("%\t");

}
```

Bien, je pense que c'est suffisant pour les moteurs... et pour ce cours !

Ce n'est qu'un au revoir


Voici donc la fin de ce cours d'initiation sur la programmation avec la carte Arduino.

Vous savez maintenant créer toutes sortes de montages électroniques et les piloter à l'aide d'une carte Arduino. Je refais le tour des beaux souvenirs de vos accomplissements de ce cours :

- Les montages que vous avez appris à faire : avec des LED, des boutons poussoirs, des potentiomètres, des moteurs et servo-moteurs. Que ce soit pour faire clignoter des LED, programmer un dé numérique, mettre en mouvement un moteur...
- Les programmes que vous avez créés pour piloter ces montages :
 - avec des boucles, des conditions, des fonctions, des tests, les constantes et variables qui vont bien, sans oublier des commentaires de code clairs ;
 - avec les fonctions typiques de l'Arduino, leurs constantes et quelques bibliothèques ;
 - avec le moniteur série de l'IDE d'Arduino pour afficher des informations du montage en direct.

Il reste évidemment bien des choses à apprendre encore, c'est la raison pour laquelle un cours plus avancé va sortir prochainement pour ceux qui cherchent à se perfectionner. En attendant, je vous fournis en annexe de ce cours un complément très utile pour l'utilisation de divers capteurs compatibles avec l'Arduino !

Voici quelques compléments utiles pour vous donner l'eau à la bouche et l'info à la tête :

- La [liste des instructions](#) de base du langage de l'IDE. Vous la trouvez aussi en allant dans le menu Aide -> Références de l'IDE. Vous en reconnaîtrez plusieurs que vous maîtrisez déjà.
 - Quelques sites pour commander votre matériel électronique : [McHobby](#), [AdaFruit](#), [robotshop](#), [gotronic](#)
- 
- Découvrez de nombreuses applications réalisées avec Arduino dans cette [vidéo de présentation](#) de l'un de ses co-fondateurs, Massimo Banzi. Et si vous êtes encore curieux de plus de projets, tapez "projets Arduino" dans un moteur de recherche et laissez-vous embarquer par tous les liens offerts...

Je vous le rappelle, la communauté est suffisamment grande pour que vous trouviez réponses à vos questions (techniques, matérielles, électriques...) en visitant le [forum d'électronique](#) d'OpenClassrooms bien sûr, mais aussi les blogs proposés par les dizaines de passionnés !

J'espère en tous cas que vous avez appris suffisamment pour avoir contracté le virus Arduino !
C'est tout le mal que je vous souhaite !

