

Moteurs : Apprendre à utiliser un moteur pas à pas **bipolaire** (en mode « full-step ») avec une carte Arduino.



Ateliers Arduino

par X. HINAULT

www.mon-club-elec.fr



Tous droits réservés – 2012.

Ce document légèrement payant est soumis au droit d'auteur et est réservé à l'usage personnel.

Afin d'encourager la production de supports didactiques de qualité, ce document est légèrement payant.

La licence d'utilisation est attribuée pour un usage personnel uniquement, dans le cercle familial. Mise en ligne et diffusion non autorisées.

Si vous n'êtes pas le détenteur de la licence attribuée pour l'usage de ce document, soyez sympa, merci d'acheter votre exemplaire personnel ici : <https://monclubelec.dpdcart.com/>

Pour tout problème lié à l'utilisation de ce document, veuillez envoyer une copie ici : support@mon-club-elec.fr

Pour obtenir tout autres types de licence d'utilisation (enseignement, commercial, etc...), veuillez contacter l'auteur ici : support@mon-club-elec.fr

Vous avez constaté une erreur ? une coquille ? N'hésitez pas à nous le signaler à cette adresse : support@mon-club-elec.fr

Truc d'utilisation : visualiser ce document en mode diaporama dans le visionneur PDF. Navigation avec les flèches HAUT / BAS ou la souris.

En mode fenêtre, activer le panneau latéral vous facilitera la navigation dans le document. Bonne lecture !

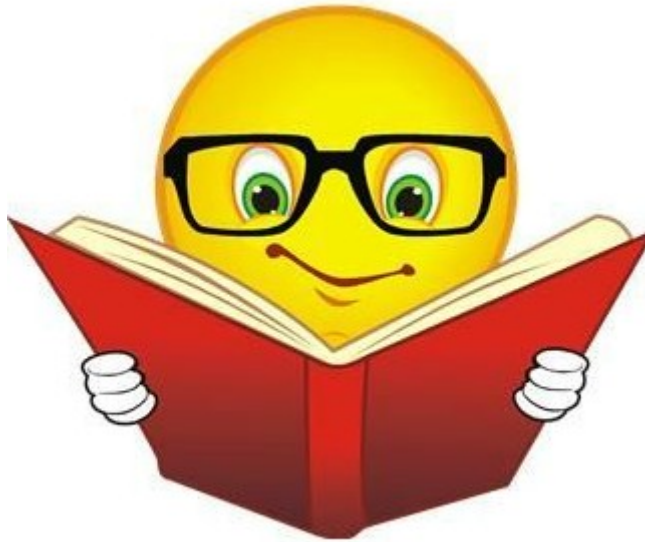
Lancer également le logiciel Arduino et connecter votre carte Arduino afin de pouvoir tester au fur et à mesure les codes d'exemples !

1. Intro

L'objectif ici est :

- de comprendre le fonctionnement d'un moteur pas à pas bipolaire
- de découvrir les interfaces de contrôle d'un moteur pas à pas
- d'apprendre à contrôler un moteur pas à pas bipolaire

... afin d'être capable d'utiliser un moteurs pas à pas bipolaire en mode de base, ou « full-step », avec une carte Arduino.

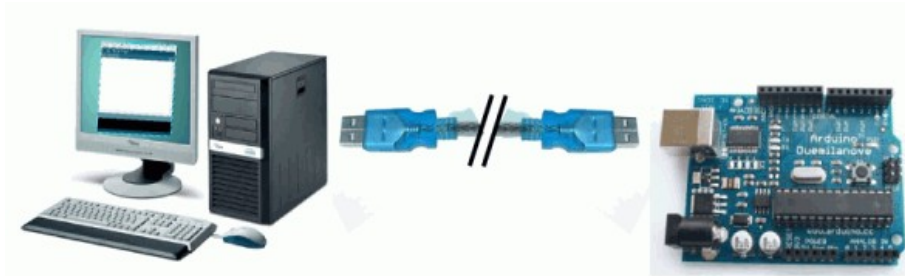


Prêt ? C'est parti !

2. Matériel nécessaire pour les ateliers Arduino

Pour cet atelier, vous aurez besoin de tout ou partie des éléments suivants pour pouvoir réaliser les exemples proposés :

De l'espace de développement Arduino

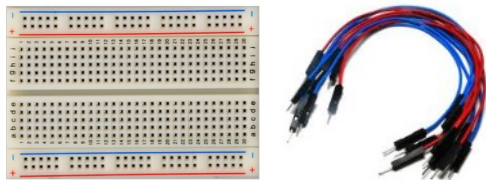


L'espace de développement Arduino associe :

- un ordinateur sous Windows, Mac Os X ou Gnu/Linux (Ubuntu)
- avec le logiciel Arduino installé (voir : <http://www.arduino.cc/>)
- un câble USB
- une carte Arduino UNO ou équivalente.

disponible chez : <http://shop.snootlab.com/> ou <http://www.gotronic.fr/>

Du nécessaire pour réaliser des montages sans soudure

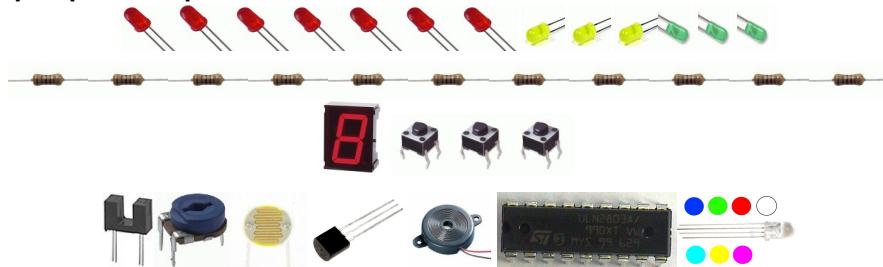


Pour réaliser des montages sans soudure, vous aurez besoin :

- d'une plaque d'essai ou breadboard moyenne (450 points)
- de quelques câbles souples (ou jumpers) mâle/mâle

disponible chez : <http://www.gotronic.fr/>

De quelques composants de base



Pour vous simplifier la vie, nous avons négocié ce kit pour vous !

Vous pouvez commander ce kit complet directement en 1 clic chez notre partenaire
<http://www.gotronic.fr/> avec le code express **701710**

GO TRONIC
ROBOTIQUE ET COMPOSANTS ÉLECTRONIQUES

Pour plus de détails, voir : http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS

Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

- des LEDs 5mm Rouges(x20), Vertes (x5) et 3 Jaunes (x5)
- digit à cathode commune rouge 13mm (x1)
- Résistances (1/4w - 5%) de 270 Ohms (x20), 4,7K Ohms (x1), 1K Ohms (x1)
- mini bouton-poussoir (x3)
- Opto-fourche (x 1)
- Résistance variable linéaire 10K (x 1)
- Photo-résistance 7mm (x 1)
- Capteur de température LM35DZ (-55/+150°C - 10mV/°C) (x 1)
- Capsule son piézoélectrique (x 1)
- ULN 2803A (CI amplificateur 8 voies, 500mA/ voie) (x 1)
- LED 5mm multicolore RVB cathode commune (x 1)

3. Matériel spécifique nécessaire pour cet atelier

Pour cet atelier vous aurez besoin également :

D'une interface pour moteur pas à pas : le shield 2A monté Arduino Motor Shield Rev 3 (L298) – env. 25 € (prix / moteur CC = $26,5/4 = 12,50\text{€}$)



Cette version la plus récente du Motor Shield **livrée montée** est basée sur un CI 298 (double pont en H) :

- permet de contrôler **2 moteurs CC ou 1 moteur pas à pas bipolaire**
 - chaque moteur est contrôlé par une broche de sens et une broche de vitesse PWM
 - dispose de LEDs de fonctionnement et de diodes de protection + bouton reset
 - intensité maximale de **2A/ phase ou moteur (4A en tout)**
 - dispose d'un **capteur analogique d'intensité intégré** (+++) (1,65V/A) pour chaque phase/moteur
 - dispose de connecteurs droits pour 2 entrées analogiques, 2 sorties PWM et TWI
- (Pour mémoire, les connecteurs droits TinkerKit analog IN et OUT ont le brochage suivant : +5V / broche / 0V)
- alimentation entre 7 et 12V / 4000mA
 - borniers à vis pour Vin et les 2 sorties moteurs
 - report des broches de la carte Arduino sur connecteur droit femelle
- IDEAL pour un robot mobile utilisant 2 moteurs CC puissants !**

Dispo ici : <http://shop.snootlab.com/arduino/186-arduino-motor.html>

D'autres interfaces sont possibles, mais ce motor shield est tout spécialement dédié pour être utilisé avec Arduino.

Il présente par ailleurs l'avantage de pouvoir contrôler 2 moteurs CC ou un moteur pas à pas.

D'un moteur pas à pas bipolaire :



Choisir un moteur pas à pas bipolaire neuf ou de récupération :

- alimentable en 6/12V,
- de 200 pas typiquement (mais peu importe)
- bipolaire (= avec 4 fils)

Exemple : <http://www.gotronic.fr/art-moteur-cnc1-878.htm>

Purchased by Franck Ourion, franck.ourion@univ-lorraine.fr #6280170

Atelier Arduino : Moteurs : Apprendre à utiliser des moteurs pas à pas bipolaires (en mode « full step ») avec une carte Arduino.

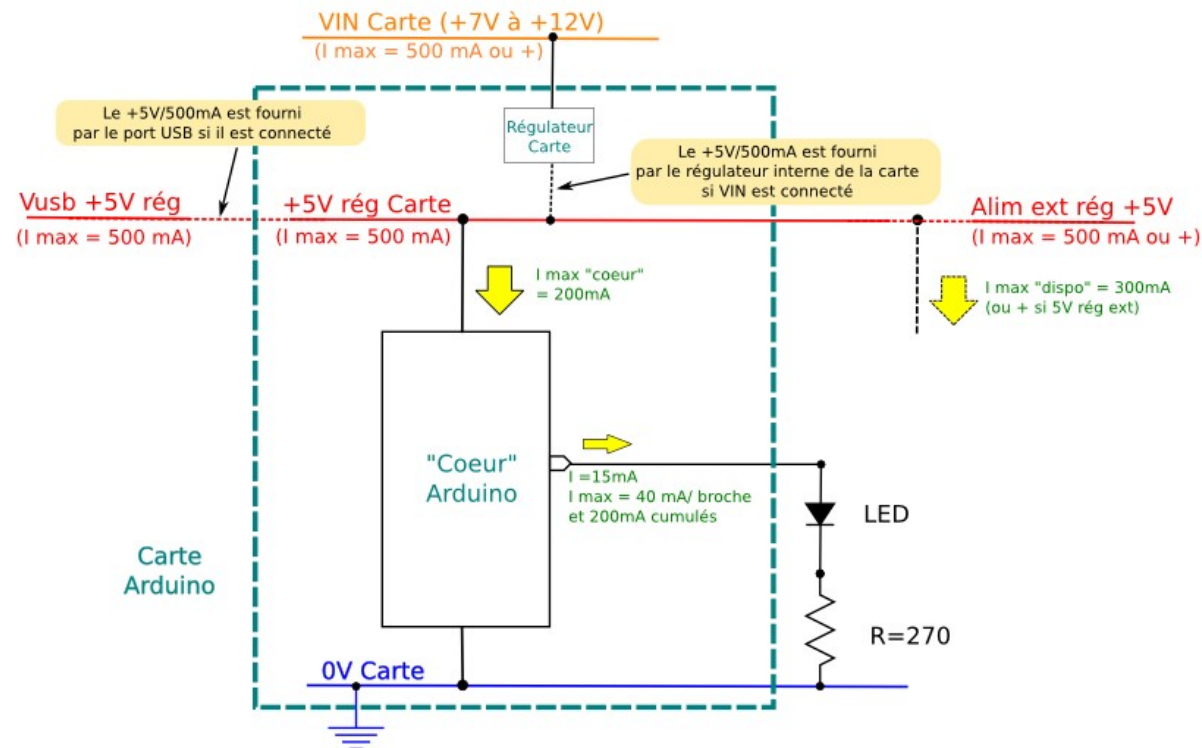
4. Rappel : Technique : l'alimentation de la carte Arduino

Lorsque l'on utilise un moteur, il est essentiel d'avoir toujours à l'esprit les notions d'intensité et de tension de la carte Arduino. Comme on l'a déjà vu, la carte Arduino intègre une alimentation interne régulée de 5V / 500mA :

- soit en provenance du port USB
- soit à partir de l'alimentation V_{in} 7-12V / 500mA (ou +)

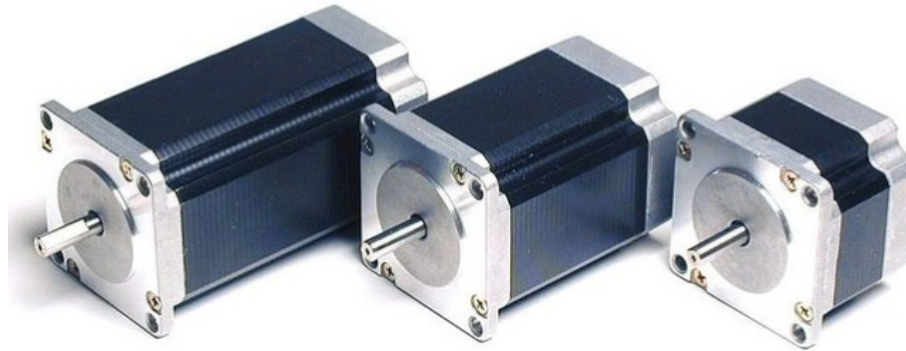
Il est essentiel de distinguer :

- **l'intensité maximale (200mA) que peut fournir le « coeur »** de la carte Arduino et limité à **40mA par broche** en sortie mais **200mA maximum** pour l'ensemble des broches réunies. Une LED en série avec une résistance utilisera par exemple 15mA fournis par le « coeur » Arduino.
- **l'intensité maximale (500mA) que peut fournir l'alimentation +5V régulé/500mA**. Cette alimentation de +5V/500mA de la carte Arduino peut également être mise en parallèle avec une alimentation externe +5V régulée au besoin.
- **On dispose donc de 300mA supplémentaires maxi pour alimenter des dispositifs 5V directement à partir de l'alimentation de la carte Arduino (mais pas à partir des broches !!)**



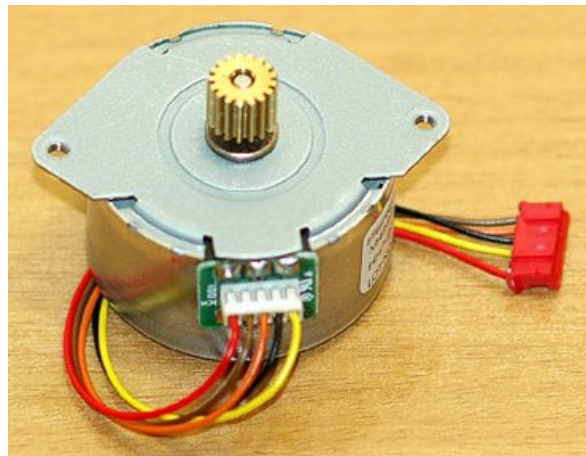
5. Les moteurs pas à pas : concrètement

Les moteurs pas à pas existent en différentes tailles et puissance :



Deux types principaux de moteurs pas à pas : unipolaires (5 fils de commande – à gauche) et bipolaires (4 fils de commande – à droite)

Dans cet atelier, nous parlerons des **moteurs pas à pas bipolaires**.



Purchased by Franck Ourion, franck.ourion@univ-lorraine.fr #6280170

Atelier Arduino : Moteurs : Apprendre à utiliser des moteurs pas à pas bipolaires (en mode « full step ») avec une carte Arduino.

6. Les moteurs pas à pas bipolaires : fiche technique.

Description

- Un moteur pas à pas est un moteur dont la rotation est découpée en plusieurs dizaines ou centaines de « crans unitaires », à la manière d'une trotteuse de montre à aiguille, appelés pas. Ces moteurs sont silencieux, précis et plus complexes à contrôler que de simples moteurs CC.

Typiquement, les moteurs pas à pas sont utilisés pour toutes les motorisations de précision : imprimantes, scanners, lecteur CD.

Type de mouvement

- Plusieurs dizaines ou centaines de « crans unitaires » appelés pas (si on tourne à la main l'axe d'un moteur pas à pas hors tension, on « sentira » les pas) :
 - un moteur pas à pas pourra donc réaliser sur 360° des rotations angulaires précises d'un angle multiple du pas unitaire (typiquement 1,8° par pas pour un moteur de 200 pas)
 - un moteur pas à pas pourra également assurer des rotations continues

Brochage

- Il existe 2 types de moteurs pas à pas qui se distinguent par le nombre de broches utilisées et leur câblage interne.
- Les moteurs pas à pas unipolaires ont 5 broches :
 - une broche commune
 - une broche de contrôle pour chaque phase (4 phases = 4 broches)
- Les moteurs pas à pas bipolaires ont 4 broches : 2 broches de contrôle pour chaque phase (2 phases = 4 broches)

Principe de contrôle d'un moteur pas à pas

- Un moteur pas à pas se contrôle par une séquence de niveau HAUT successifs sur 4 broches entraînant la succession des pas.

Caractéristiques mécaniques

- Un moteur pas à pas est caractérisé par son couple de maintien (ex: 1,25 kg/cm) et son nombre de pas (ex : 200 pas)

Caractéristiques électriques

- Un moteur pas à pas est caractérisé par sa tension d'alimentation (qui devra être entre 7 et 12V avec Arduino) et son intensité par phase (1A par exemple).

Maintien de position « hors-tension »

- Hors tension, un moteur pas à pas est en « roue libre » et ne tient pas la position courante. La position reste bloquée sous tension.

Codage

- Facile avec la librairie **Stepper** du langage Arduino.

Interface de « puissance »

- INTERFACE « de puissance » obligatoire** afin de fournir l'intensité nécessaire par phase et d'assurer le contrôle de la polarité aux bornes des phases (moteur pas à pas bipolaire).

Principe d'alimentation

- Nécessité d'une alimentation externe pouvant fournir une intensité suffisante.

Prix unitaire

- Des modèles à moins de 10€ existent

Coût global unitaire de mise en oeuvre

- Il faut un étage entier « double pont en H » pour contrôler un moteur pas à pas bipolaire. Le coût unitaire de mise en oeuvre est le prix du moteur + l'étage « double pont en H associé » : soit **un minimum de 10€ + 13,5€ = 23,50€ par moteur pas à pas.**



Utilisation type

- Les moteurs pas à pas sont très utiles pour toutes les motorisations de précision, notamment les applications de type découpe numérique (CNC)
- Les moteurs pas à pas sont également intéressants pour des mouvements de robot mobile précis et silencieux.

Avantages

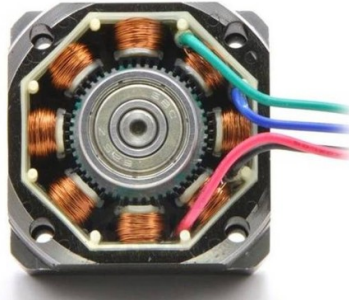
- Silence et précision

Inconvénients

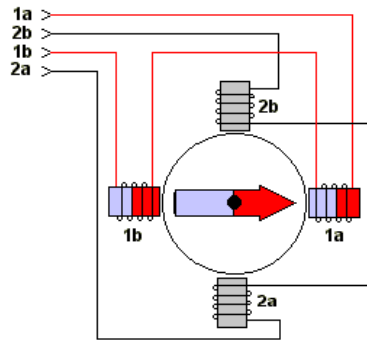
- Nombre de broches de commande important : jusqu'à 4 par moteur à la mise en oeuvre selon le type d'interface utilisée...
- Couple faible en rotation continue à vitesse rapide**

7. Pour comprendre : Les moteurs pas à pas bipolaires : structure interne et principe de fonctionnement

- Il est beaucoup plus facile de comprendre le fonctionnement d'un moteur pas à pas bipolaire une fois que l'on connaît le principe de sa structure fonctionnelle interne.
- Voici la vue interne réelle d'un moteur pas à pas bipolaire :

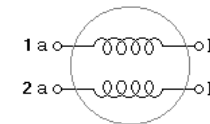


- Afin de pouvoir assurer une rotation par « crans », on utilise une structure interne basée sur l'utilisation de 2 « bobines » astucieusement combinées de façon à permettre d'entraîner une rotation du « rotor » (axe du moteur) par « à-coup » précis.



- Le rotor interne est aimanté, comme pour un moteur à courant continu. Ainsi, pour assurer la rotation par « pas », il suffira d'utiliser une séquence précise de mise sous tension des bobines du moteur de façon à ce que le rotor s'oriente successivement en position 1a puis 1b puis 2b puis 2a, etc...

- Dans ce schéma, remarquer :
 - les 2 bobines appelées 1 et 2 (chaque bobine est divisée en 2 sous bobines reliées entre-elles mais l'ensemble constitue une seule bobine).
 - chaque bobine est appelée « phase » et dispose donc de 2 pôles appelés a et b
 - on retrouve bien au final les 4 broches du connecteur d'un moteur pas à pas, à savoir :
 - les 2 broches 1a et 1b de la bobine 1
 - les 2 broches 2a et 2b de la bobine 2
- Pour simplifier les choses, électriquement, un moteur pas à pas, cela revient à un ensemble de 2 « bobines » :



Truc technique :

Il est très facile de savoir quelles sont les broches d'un moteur pas à pas correspondant aux 2 phases : il suffit de tester les broches du connecteur du moteur à l'aide d'un multimètre réglé sur la mesure de résistance. 2 broches reliées entre-elles appartiennent à la même phase. Faites le test !

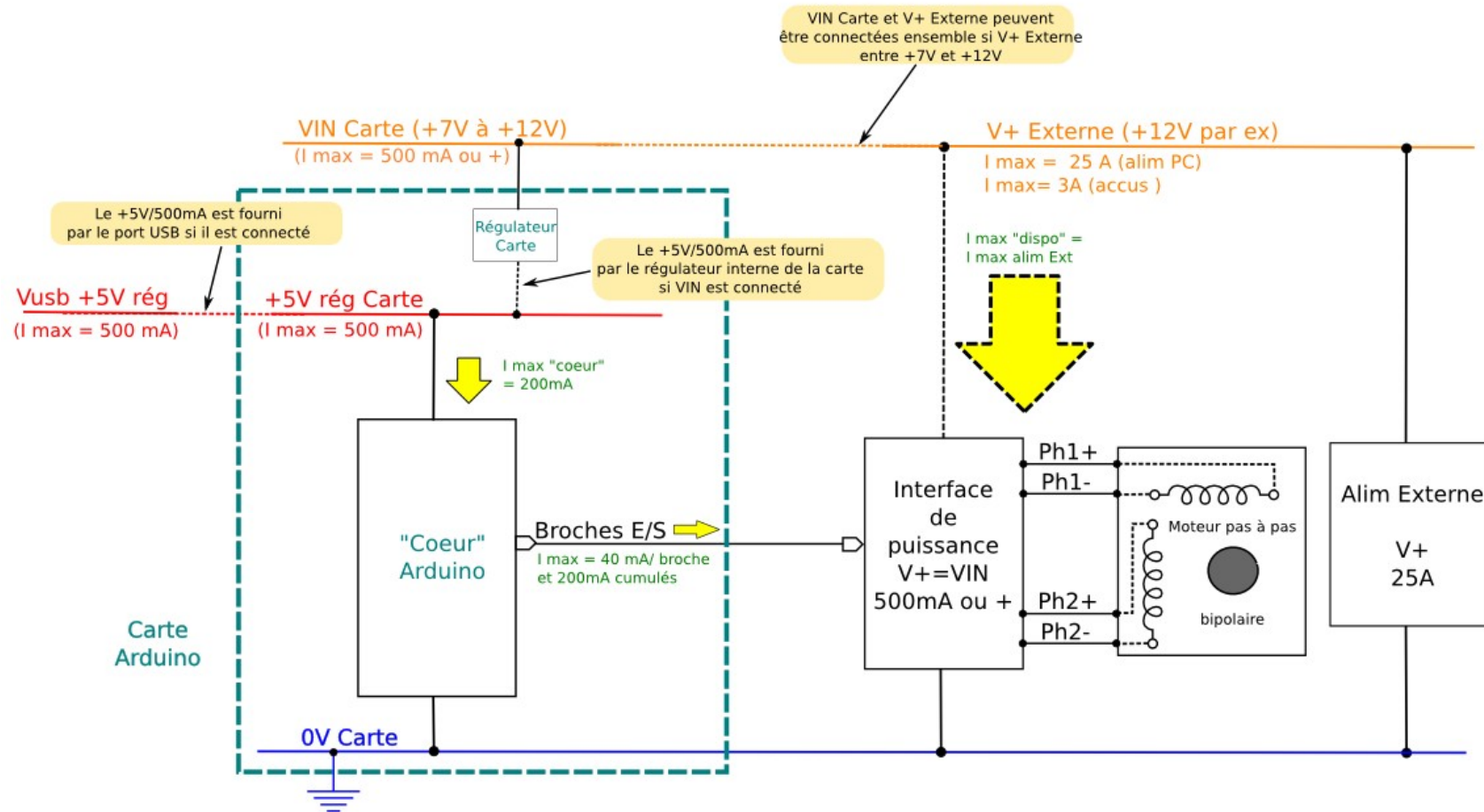
- D'un point de vue électrique, **une bobine c'est l'équivalent d'un moteur à courant continu** (en fait c'est l'inverse : c'est un moteur à courant continu qui est électriquement l'équivalent d'une bobine... mais bref, passons...) Tout ça pour dire que **contrôler un moteur pas à pas bipolaire, cela revient à contrôler 2 bobines et donc l'équivalent de 2 moteurs à courant continu.**
- Comme nous l'avons vu, pour contrôler un moteur à courant continu, il faut utiliser un circuit appelé « pont en H » : **pour contrôler un moteur pas à pas bipolaire, il en faudra... 2 (« double pont en H »)!**

Vous comprenez à présent pourquoi une interface capable de contrôler 2 moteurs à courant continu peut également contrôler 1 moteur pas à pas !

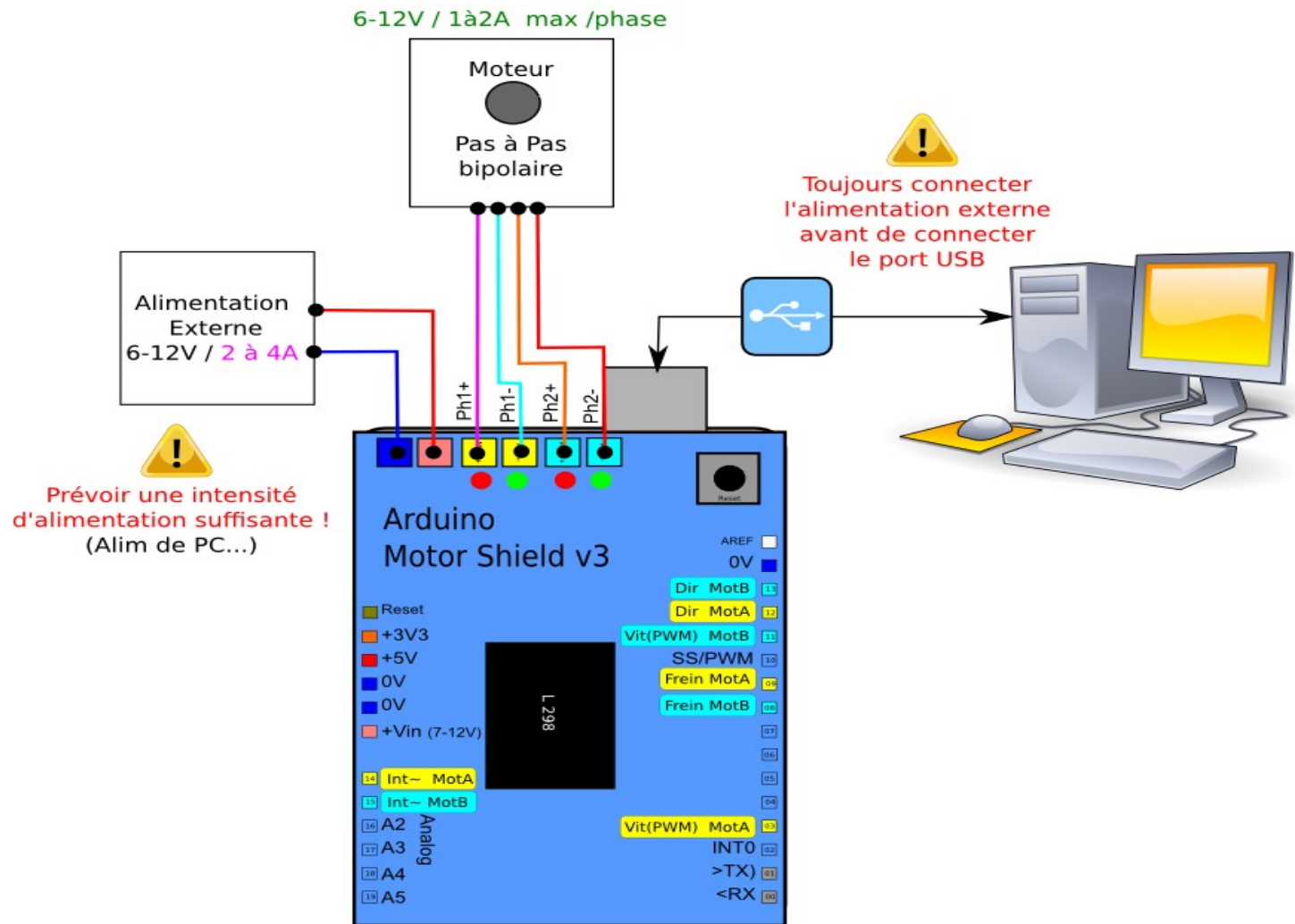
8. Les moteurs pas à pas bipolaires : schéma électrique type d'utilisation avec Arduino

A présent que l'on connaît la structure interne d'un moteur pas à pas, on comprendra facilement l'utilisation avec une carte Arduino. On va connecter :

- la phase 1 sur la sortie d'un 1er « pont en H »
- la phase 2 sur la sortie d'un 2ème « pont en H »

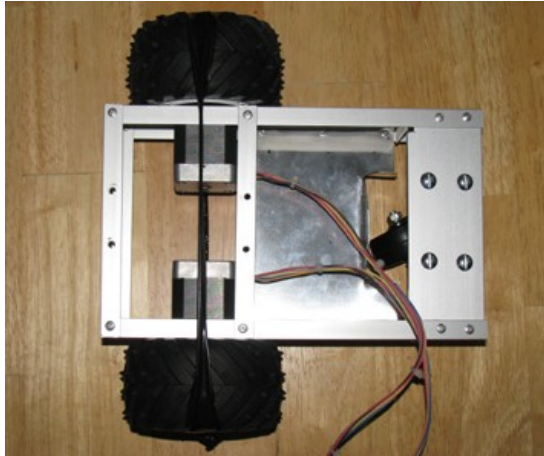


9. Les moteurs pas à pas bipolaires : montage type avec une carte Arduino



Truc : pour connaître le brochage d'un moteur pas à pas bipolaire, connecter 2 à 2 les broches à l'aide d'un multimètre réglé sur la fonction Ohmètre : lorsque le contact se fait (résistance=0), les 2 broches appartiennent à la même phase du moteur.

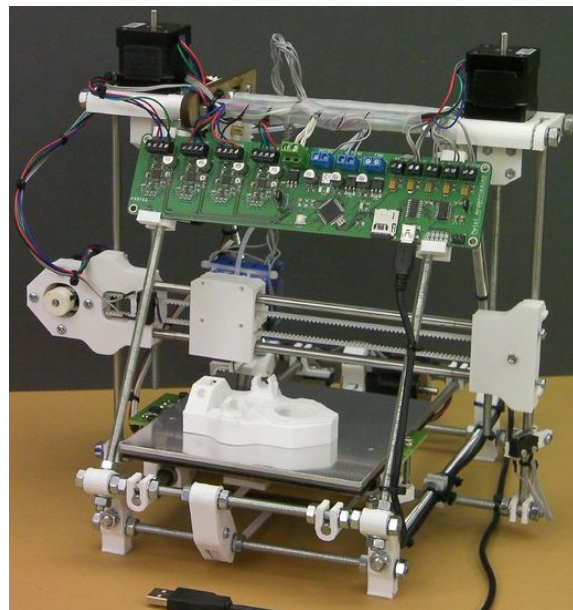
10. Les moteurs pas à pas : exemples d'utilisation



Châssis de robot mobile (silencieux et précis)



Découpe et perçage CNC



Imprimante 3D

voir le projet open source RepRap : <http://reprap.org/>

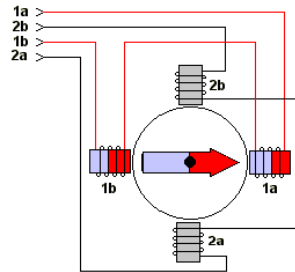
Purchased by Franck Ourion, franck.ourion@univ-lorraine.fr #6280170

Atelier Arduino : Moteurs : Apprendre à utiliser des moteurs pas à pas bipolaires (en mode « full step ») avec une carte Arduino.

11. Les moteurs pas à pas bipolaires : principe de contrôle à l'aide d'un « double-pont en H »

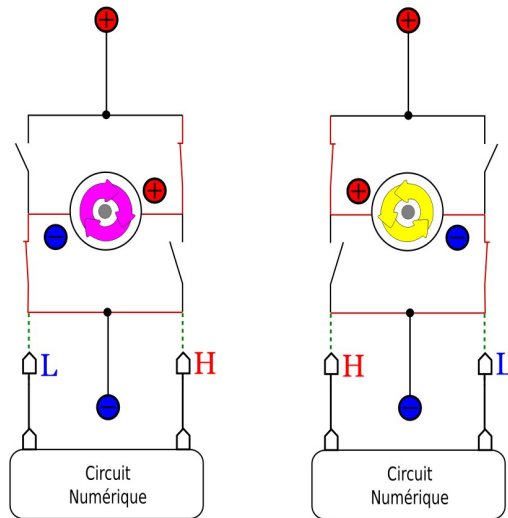
Schéma interne d'un moteur pas à pas bipolaire

Comme on l'a déjà vu, le moteur pas à pas est construit autour de 2 bobines appelées phases (ici chaque bobine est divisée en 2 sous-bobines reliées entre-elles) et contrôlées chacune par 2 broches, permettant d'assurer la rotation de l'axe par « pas » :



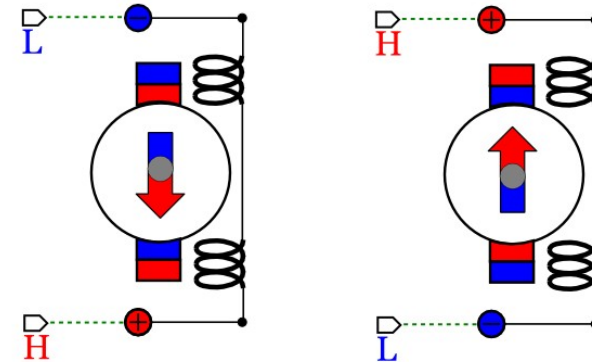
Rappel du principe d'un « pont en H »

- Le principe interne d'un circuit « H-Bridge » ou « pont en H » est le suivant : **à chaque broche numérique de commande sont associés électroniquement l'équivalent de 2 « interrupteurs » contrôlant la connexion d'une broche du moteur au V+ et au V-.** Ainsi :
 - lorsqu'une broche de commande est au niveau **HAUT**, la broche correspondante du moteur est au niveau **V+**
 - lorsqu'une broche de commande est au niveau **BAS**, la broche correspondante du moteur est au niveau **V-**



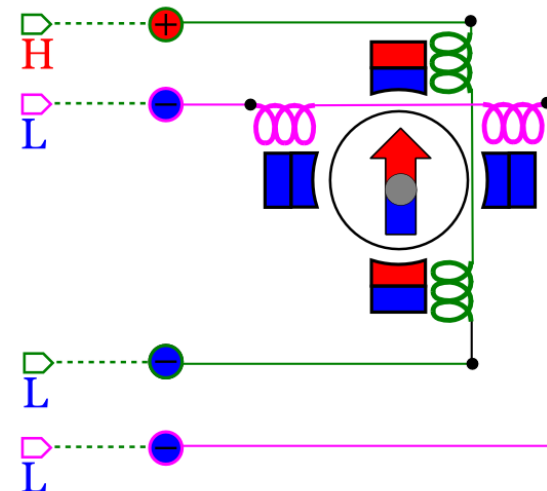
Principe de contrôle d'une bobine avec un « pont en H »

Une bobine peut être assimilée à un moteur d'un point de vue électrique et dès lors son contrôle pourra se faire de la même façon qu'un moteur, à savoir à l'aide d'un pont en H :



Principe du contrôle des 2 bobines avec un double « pont en H »

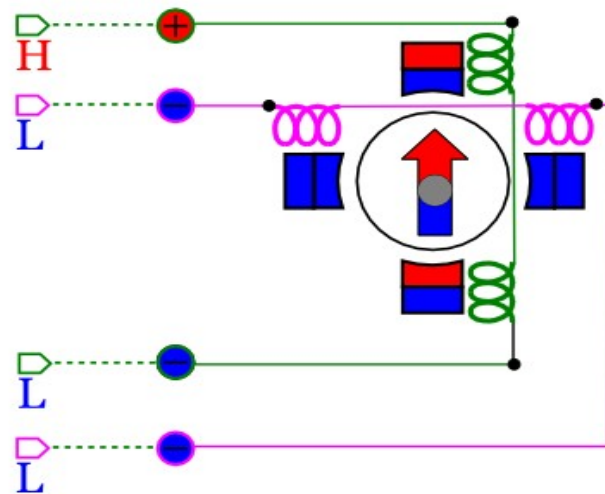
De la même façon le contrôle des 2 bobines (ou phases) nécessitera 2 pont en H ou « double-pont en H » :



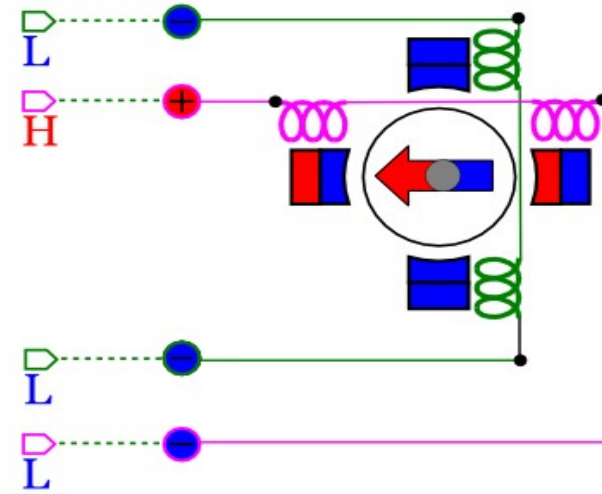
Vous comprenez ainsi pourquoi une interface capable de contrôler 2 moteurs CC sera capable de contrôler 1 moteur pas à pas bipolaire.

12. Les moteurs pas à pas bipolaires : principe de contrôle des pas

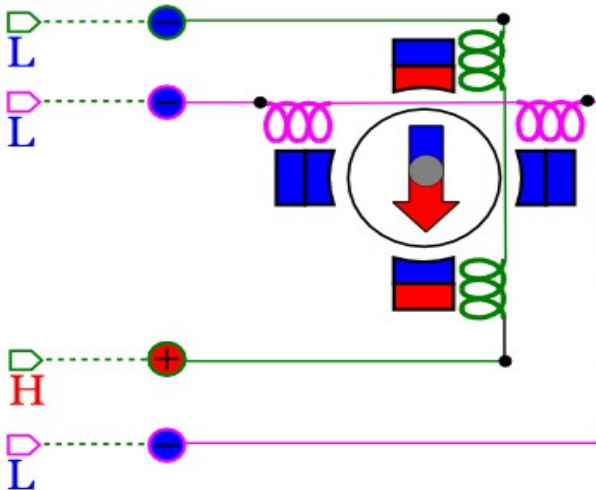
Il devient dès lors facile de comprendre comment réaliser le contrôle du moteur pas à pas bipolaire à partir de 4 broches numériques de commandes et un double « pont en H » de façon à obtenir une rotation :



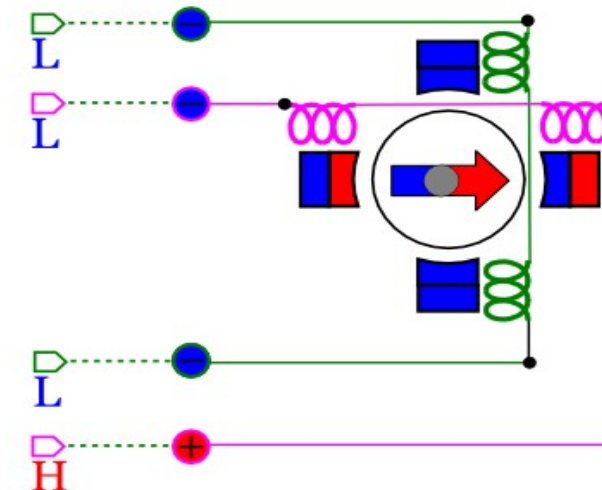
Pas 1



Pas 2



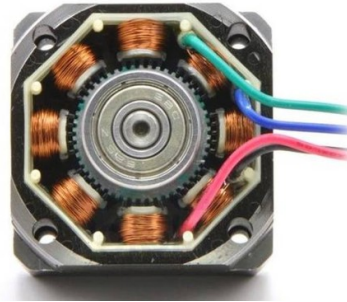
Pas 3



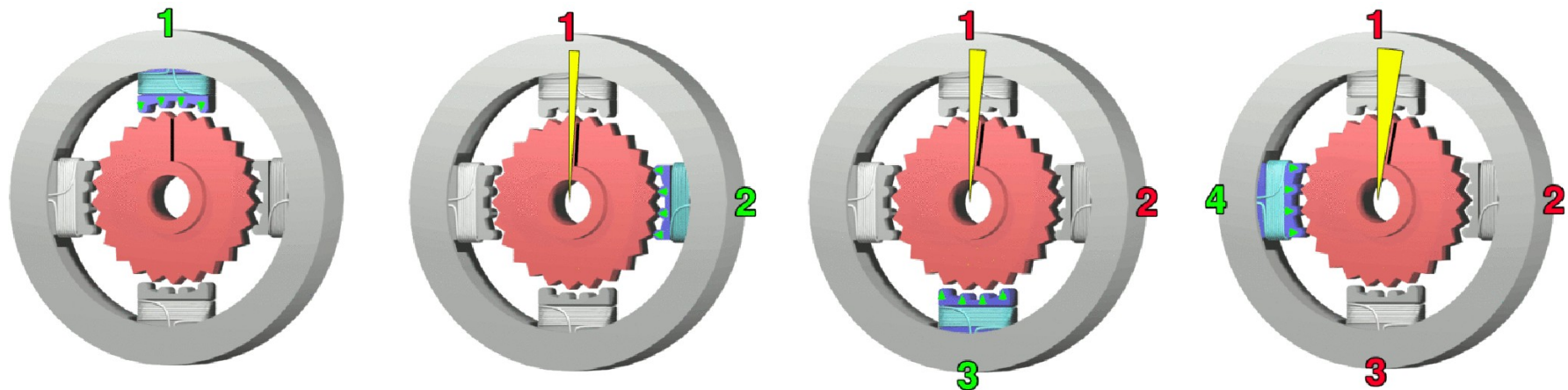
Pas 4

13. Mouvement du moteur pas à pas réel

Dans un moteur pas à pas réel, les bobines de phases sont subdivisées en plusieurs sous-bobines de façon à entraîner de façon harmonieuse le rotor cranté et qui possède un grand nombre de pas (typiquement 200).



Voici la synthèse du mouvement sur le moteur réel :



L'animation est disponible ici : <http://en.wikipedia.org/wiki/File:StepperMotor.gif>

14. Les moteurs pas à pas bipolaires : séquences de contrôle des pas

La séquence de base

Il est plus pratique de synthétiser dans un tableau la séquence précédente, ce qui donne :

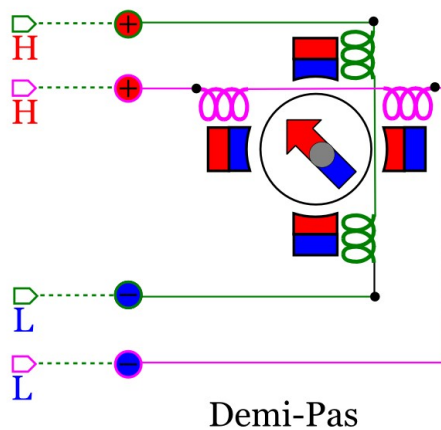
n° pas	Ph 1 +	Ph 2 +	Ph 1 -	Ph 2 -
1	■	■	■	■
2	■	■	■	■
3	■	■	■	■
4	■	■	■	■
5	■	■	■	■
6	■	■	■	■
7	■	■	■	■
8	■	■	■	■
Etc...				

Ainsi, contrôler un moteur pas à pas en rotation revient à générer cette séquence sur 4 broches de la carte Arduino.

La rotation inverse est obtenue simplement en inversant l'ordre des pas.

Demi-pas

Il est possible également de créer des pas intermédiaires en activant simultanément 2 phases, le rotor se positionnant alors de façon intermédiaire :



Séquence « half-step »

On obtient alors soit la séquence simple mais en se basant sur des « demi-pas », soit la combinaison de la séquence simple et de la séquence « half-step » ce qui double le nombre de pas :

n° pas	Ph 1 +	Ph 2 +	Ph 1 -	Ph 2 -
1	■	■	■	■
2	■	■	■	■
3	■	■	■	■
4	■	■	■	■
5	■	■	■	■
6	■	■	■	■
7	■	■	■	■
8	■	■	■	■
Etc...				

Vous n'avez pas besoin de retenir ces séquences, mais simplement d'en comprendre le principe.

Autres séquences

En combinant astucieusement et de façon complexe l'activation des phases et l'intensité circulant dans les phases, on peut ainsi obtenir des séquences de plus en plus poussées permettant d'utiliser le 1/4 de pas, le 1/8e de pas et même le 1/16e de pas ! (Voir l'atelier consacré au mode « micropas »)

Ces séquences ne sont pas accessibles à l'aide d'un « double-pont en H » mais en utilisant des circuits spécialisés de contrôle de moteurs pas à pas.

L'intérêt majeur de ces séquences est d'augmenter considérablement la précision des moteurs pas à pas : ainsi un moteur de 200 pas permet d'accéder à 3200 positions intermédiaires en utilisant une séquence au 1/16ème de pas !

Exemple d'application

Imaginons un moteur de 200 pas dont l'axe est directement connecté à une tige filetée M6 (1mm par tour) et utilisé en mode 1/16ème de pas : on obtient donc une précision théorique de contrôle de la rotation de $1 \text{ mm} / 3200 \text{ pas} = 0,000312 \text{ mm}$ soit $3,12 \mu\text{m/pas}$! Et tout ça avec du matériel standard !

15. Moteurs pas à pas bipolaires : interfaces de contrôle

Vue d'ensemble

Grosso modo, pour faire simple, on peut distinguer 2 types d'interfaces :

- les interfaces de **contrôle direct de la séquence par 2 ou 4 broches numériques** par moteur (voir ci-dessous)
- les interfaces avec « driver » de séquence intégré contrôlé par **2 broches numériques (sens et vitesse)** et gérant les séquences en mode demi-pas, quart, huitième ou même seizième de pas !

Les interfaces de contrôle direct de la séquence :

Avec ces interfaces :

- La **séquence de contrôle doit être générée par Arduino** :
 - sur 2 broches numériques (cas des interfaces avec broches sens et vitesse pour chaque étage « pont en H »)
 - ou 4 broches numériques,
 - ce qui est fait assez simplement avec la librairie **Stepper**
- Seules les séquences principales et half-step sont disponibles en pratique
- Ces interfaces sont **basées sur des CI « double pont en H »** et sont les mêmes que pour les moteurs CC : **avec 1 interface pour 2 moteurs CC, on pourra gérer 1 moteur pas à pas.**

Les interfaces avec driver de séquence intégré

- Le contrôle par Arduino se limite à l'utilisation de **2 broches numériques** comme pour les interfaces de moteurs CC :
 - une broche de sens
 - une broche de vitesse = en fait cadencage des pas
- Ces **interfaces s'occupent de générer les séquences de contrôle élaborées et complexes** en mode demi-pas, quart, huitième ou même seizième de pas !
- Le **type de séquence utilisée est fixé à l'aide de broches numériques** de sélection câblées selon les besoins.
- Ces interfaces sont **basées sur des CI spécialisés** (mais pas forcément plus chers... !) Notamment le A4988 de chez Allegro, sur lequel est basé le très pratique carte de Polulu A4988 (utilisée sur les imprimantes 3D rebrap !)

Remarquer ici comme il est possible de décharger la partie numérique de commande attribuée au micro-contrôleur Arduino en utilisant une interface spécialisée qui s'occupera automatiquement de la partie « fastidieuse » libérant le micro-contrôleur pour des tâches plus spécifiques.

Exemples de cartes d'interface à base de L293 (double Pont H – 1A) x1 ou à base de L298 (double Pont H – 2A) x1

- Ces cartes permettent de contrôler **2** moteurs CC ou 1 moteur pas à pas bipolaire.



Exemples de cartes d'interface à base de L293 (double Pont H – 1A) x2

- Ces cartes permettent de contrôler **4 moteurs CC** ou 1 moteur pas à pas bipolaire.



Exemples de cartes d'interfaces à driver de séquence pour 1 moteur pas à pas

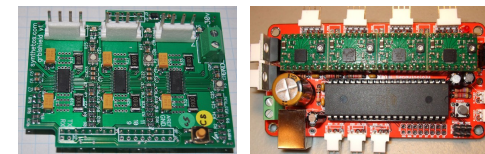
- Ces cartes permettent de contrôler 1 moteur pas à pas et sont capables de générer des séquences de contrôle élaborées jusqu'au 16ème de pas (soit 3200 pas avec un moteur de 200 pas !).



Mini-carte polulu A4988 – 2A par phase

Exemples de cartes d'interfaces à driver de séquence pour plusieurs moteurs pas à pas

- Ces cartes permettent le contrôle de 3 à 5 moteurs pas à pas bipolaires et sont très pratiques notamment pour les applications CNC (découpe numérique) ou type imprimante 3D



grbl shield

Sanguinolulu

16. Langage Arduino : Introduction aux librairies

C'est quoi une librairie Arduino ?

Le langage Arduino comporte de nombreuses instructions comme vous avez pu le constater, une quarantaine en tout. Ces instructions sont intégrées dans ce que l'on appelle le « noyau » ou « coeur » (core en Anglais) du langage Arduino. Ces instructions sont « générales » et servent souvent.

Le langage Arduino peut cependant être étendu à la demande avec des instructions dédiées à certaines applications particulières : afin de ne pas surcharger inutilement le « coeur », ces instructions spécifiques ont été intégrées dans des « paquets d'instructions » appelés librairies.

Comment ça marche ?

Par exemple, si on utilise un afficheur LCD, un servomoteur ou encore si l'on utilise un shield ethernet (réseau), on va intégrer dans notre programme la librairie dédiée correspondante.

Principe général d'utilisation

Pour intégrer une librairie dans un programme Arduino, c'est très simple : il suffit d'ajouter en début de programme une ligne de la forme :

```
#include <nomlibrairie.h> // librairie pour servomoteur
```

ATTENTION : l'instruction include est un peu particulière :
la ligne commence par un # et il n'y a pas de point virgule de fin de ligne !

Ensuite, dans le code, au niveau de l'entête déclarative, là où vous déclarez vos variables, il va falloir déclarer un objet (une sorte de super variable) représentant la librairie. Cet objet est en fait une instance (= un exemplaire) d'une Classe (=le moule) qui regroupe les fonctions de la librairie. On a :

```
ClasseObjet monObjet; // déclare un objet
```

Généralement ensuite :

- au niveau de la fonction `setup()`, on initialise l'objet avec les paramètres voulus
- au niveau de la fonction `draw()`, on appelle les fonctions de la librairie sous la forme que vous connaissez déjà :

```
monobjet.fonction( param, param, ..);
```

Rappel : pour utiliser une fonction d'une classe du langage Arduino, on utilise le nom de la classe + un point + le nom de la fonction.

Il peut exister des variantes selon les librairies, mais grosso-modo, ça fonctionne de cette façon pour la plupart des librairies Arduino.

Vous avez déjà utilisé une librairie !

Si vous êtes attentifs à tout ce qu'on a déjà vu, vous me direz que ça ressemble étrangement à l'utilisation de la classe **Serial...** et vous aurez raison ! En fait, la classe Serial est une librairie qui est intégrée implicitement lorsque vous lancez Arduino : c'est pour ça que vous n'avez pas besoin d'utiliser **#include** pour l'utiliser.

Les librairies standards Arduino

Les librairies Arduino disponibles sont nombreuses, et disposent chacune de quelques fonctions à plusieurs dizaines... ce qui étend considérablement la puissance du langage Arduino et qui en fait aussi tout son intérêt. Voici la liste des librairies standards du langage Arduino (**le logiciel donne la liste...**) :

- [La librairie Serial](#) - pour les communications séries entre la carte Arduino et l'ordinateur ou d'autres composants
- [La librairie LCD](#) - pour l'utilisation et le contrôle d'un afficheur LCD alpha-numérique standard.
- [La librairie Servo](#) - pour contrôler les servomoteurs.
- [La librairie Stepper](#) - pour contrôler les moteurs pas à pas (nécessite une interface de commande)
- [La librairie Ethernet](#) - pour se connecter à Internet en utilisant le module Arduino Ethernet
- [La librairie EEPROM](#) - référence - pour lire et écrire dans la mémoire EEPROM non volatile.
- [La librairie SD](#) - référence - pour utiliser une carte mémoire SD (utiliser des fichiers, stocker des données, ...)
- [La librairie SoftwareSerial \(Série Logicielle\)](#) - référence - pour communication série logicielle sur n'importe quelles broches de la carte Arduino
- [La librairie Wire / I2C](#) - référence - Interface "deux fils" (TWI/I2C) pour envoyer et recevoir des données sur un réseau de modules ou capteurs.
- [La librairie SPI \(Serial Peripheral Interface\)](#) - pour communication série avec des modules externes supportant le protocole SPI
- [Firmata](#) - pour communiquer avec des applications sur l'ordinateur utilisant un protocole série standard.

En jaune les plus utiles. Impressionnant non ? On les étudiera pas à pas...

Les librairies de la communauté

A côté de ces librairies standards, il existe toute une série de librairies proposées par les uns et les autres et qui concernent des matériels spécifiques, ou autre. Par exemple :

- [La librairie Keypad](#) - pour l'utilisation des claviers matriciels. (hors référence)

Faites un tour ici pour voir ce qui existe : <http://arduino.cc/playground/Main/LibraryList>

17. Langage Arduino : la librairie **Stepper** pour le contrôle des moteurs pas à pas

Présentation

La librairie **Stepper** est une librairie qui fournit les fonctions de base utiles pour utiliser un moteur pas à pas avec contrôle direct de la séquence.

Inclusion

La librairie **Stepper** s'intègre dans un programme avec la ligne (pas de ; !!) :

```
#include <Stepper.h> // inclut la librairie Stepper
```

Le constructeur de la classe

Le constructeur de la classe existe sous 2 formes :

```
Stepper moteurPAP(nombre_pas, broche1, broche2); // déclare un objet  
Stepper contrôlé par 2 broches
```

```
Stepper moteurPAP(nombre_pas, broche1, broche2, broche3, broche4); //  
déclare un objet Stepper contrôlé par 4 broches
```

Les fonctions de la librairie

La librairie dispose des 2 fonctions suivantes :

- [setSpeed\(vitesse\)](#) : fixe la vitesse de rotation en nombre de tours/min
- [step\(nombre_pas\)](#) : fixe le nombre de pas dont doit tourner le moteur : un nombre positif fait tourner dans un sens et un nombre négatif fera tourner dans l'autre sens.

Attention : la fonction [step\(\)](#) est bloquante : elle durera tout le temps nécessaire pour exécuter la rotation voulue.

Utilisation type

- Inclusion de la librairie **Stepper**
- Déclaration d'un ou plusieurs objets **Stepper** en fixant :
 - le nombre de pas
 - les broches de commande utilisées
- Initialisation de la vitesse de rotation avec la fonction [setSpeed\(\)](#)
- Rotation du nombre de pas voulus avec la fonction [step\(\)](#)

Plus d'infos sur la librairie Stepper ici :

[http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?
n=Main.LibrairieStepper](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieStepper)

```
// programme type utilisant un moteur pas à pas
```

```
//---- inclusion de librairie  
#include <Stepper.h> // inclut la librairie Stepper
```

```
//--- entete déclarative = variables et constantes globales
```

```
const int brochesPAP[]={2,3,4,5}; // tableau des broches de controle du  
moteur pas a pas  
const int nombrePas=200; // nombre de pas du moteur
```

```
Stepper moteurPAP(nombrePas,  
brochesPAP[0],brochesPAP[1],brochesPAP[2],brochesPAP[3]); // déclaration  
d'un objet moteur Pas à pas
```

```
void setup() { //--- la fonction setup() : exécutée au début et 1 seule  
fois
```

```
    moteurPAP.setSpeed(50); // 50 tours par minute
```

```
} // fin de la fonction setup()
```

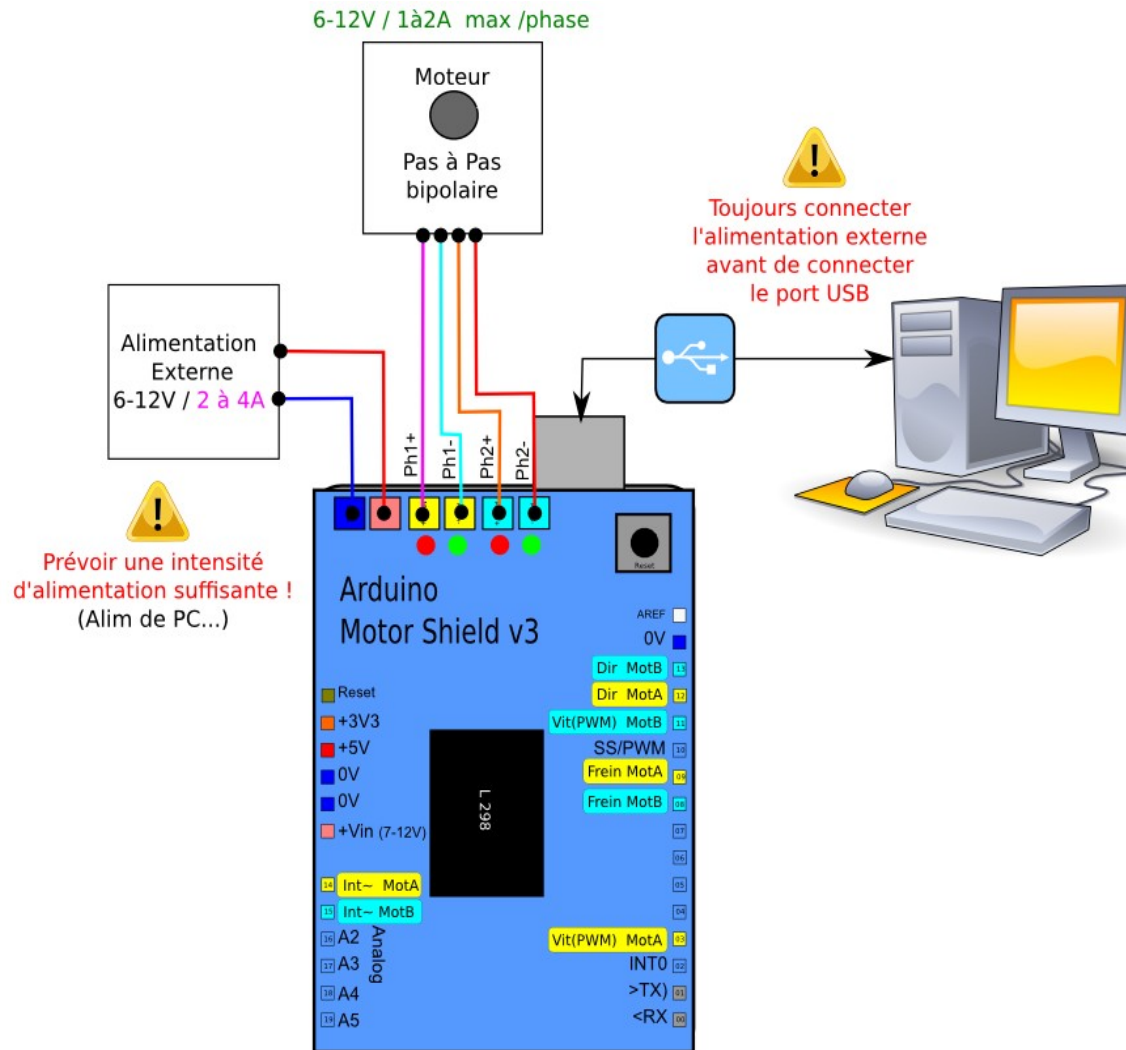
```
void loop() { //--- la fonction loop() : exécutée en boucle sans fin
```

```
    moteurPAP.step(100); // 100 pas sens positif
```

```
} // fin de la fonction loop()
```

18. Contrôle simple d'un moteur pas à pas (la trotteuse) : le montage

Voici le montage d'un moteur pas à pas bipolaire avec le Arduino Motor Shield v3, carte d'extension Arduino basée sur un « double pont en H », le L298, permettant le contrôle en **2A** de 2 moteurs CC ou 1 moteur pas à pas bipolaire (ce qui est le cas ici) :



Le brochage du Motor Shield V3

Ce shield (=carte d'extension) permet le contrôle de la polarité de 2 « bobines », soit 2 moteurs CC, soit 2 phases de moteurs pas à pas (=1 moteur pas à pas).

Chaque étage moteur (appelés A et B) est contrôlé par 3 broches :

- broche **Dir** qui fixe la polarité, le sens de rotation
- broche **Vit** (de type pwm) qui fixe la vitesse de rotation
- broche **frein** qui désactive l'étage moteur sur le niveau HAUT

Chaque étage dispose également d'une **broche analogique permettant la mesure de l'intensité circulante** dans la bobine en question selon 0V + 1,65V/A (soit 3,3V pour 2A)

Pour une description complète du shield, voir :

http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.MATERIELArduinoShieldArduinoMotroShieldV3L298

Truc : pour connaître le brochage d'un moteur pas à pas bipolaire, connecter 2 à 2 les broches à l'aide d'un multimètre réglé sur la fonction Ohmètre : lorsque le contact se fait (résistance=0), les 2 broches appartiennent à la même phase du moteur.

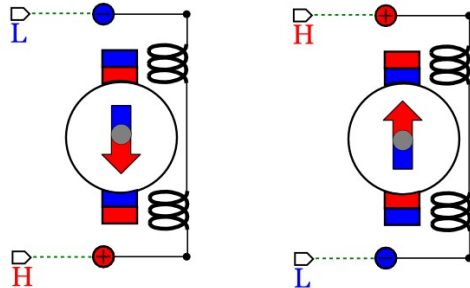
19. Contrôle simple d'un moteur pas à pas (la trotteuse) : note technique sur les interfaces « directes »

Rappel

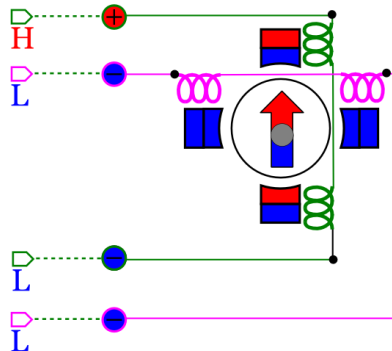
- Comme on l'a déjà dit précédemment, pour les interfaces qui réalisent un **contrôle direct de la séquence**, la **séquence de contrôle doit être générée par Arduino** (facile avec la librairie **Stepper**) :
 - sur 2 broches numériques (cas des interfaces avec broches sens et vitesse pour chaque étage « pont en H »)
 - ou sur 4 broches numériques

Cas d'un contrôle direct sur 4 broches

- Le cas de figure du contrôle par 4 broche est le plus simple à comprendre : sur ces interfaces, chaque « pont en H » est directement contrôlé par 2 broches numériques de telle façon que un niveau HAUT sur la broche donnera une polarité + sur la broche correspondante du moteur et inversement.

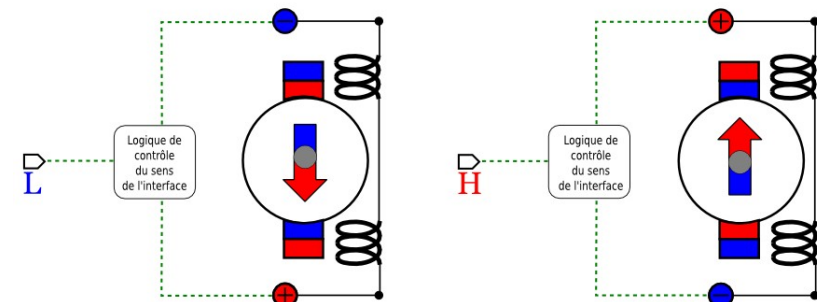


- Le moteur ayant 2 phases, on a donc bel et bien besoin de 4 broches numériques pour générer la séquence de contrôle du moteur pas à pas bipolaire.

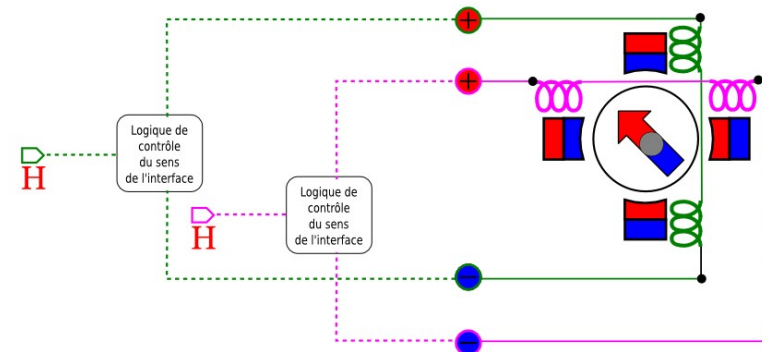


Cas d'un contrôle direct sur 2 broches

- Le cas de figure du **contrôle direct par 2 broches** est un peu plus subtil à comprendre. Il concerne les interfaces pour lesquelles chaque étage « pont en H » est contrôlé par une broche de sens et une broche de vitesse PWM.
- C'est le cas de l'interface **Arduino Moto Shield v3** utilisée ici mais aussi de nombreuses interfaces pour moteurs CC.
- En fait, avec ces interfaces, la **polarité aux bornes du moteur est contrôlée UNIQUEMENT par la broche de sens**, la broche PWM fixant simplement la durée de mise sous tension du moteur.

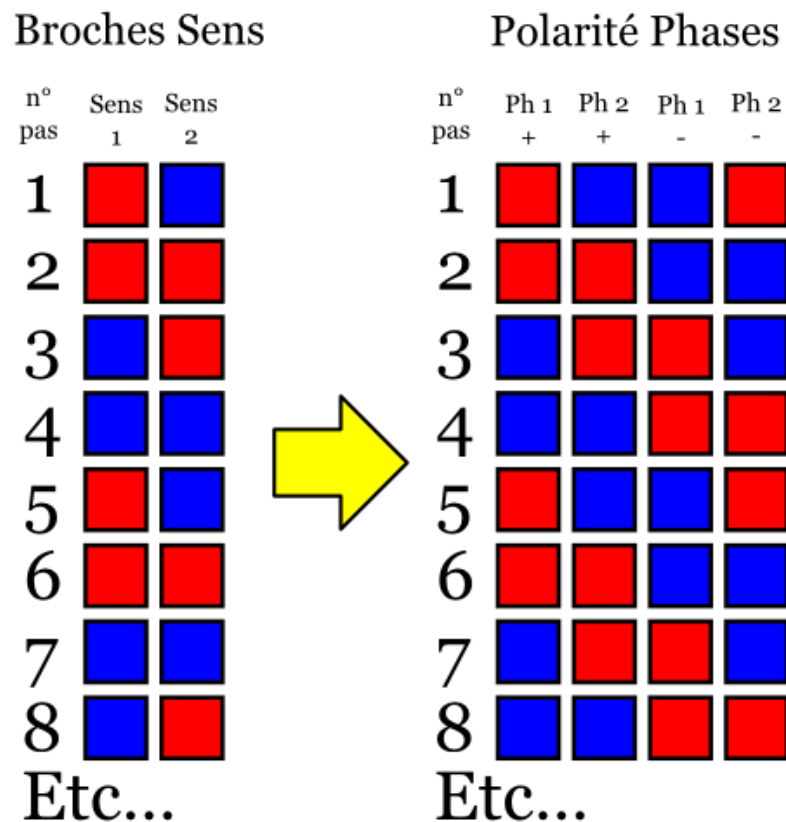


- Du coup, pour le contrôle du moteur pas à pas, il va être possible :
 - de **contrôler le moteur à partir de seulement 2 broches numériques** (la séquence obtenue sera de type « half-step »)
 - et il faudra **laisser les broches PWM au niveau HAUT**



20. Contrôle simple d'un moteur pas à pas (la trotteuse) : la séquence de contrôle direct avec 2 broches

Au final, voici la correspondance obtenue entre l'état des 2 broches de sens utilisée pour le contrôle du moteur pas à pas et la polarité des phases du moteurs pas à pas dans le cas d'une interface directe avec contrôle par 2 broches de sens (les broches PWM devant être mises à l'état HAUT) :



Remarquer que la séquence obtenue est « half step »

Je prends le temps de décrire cela ici pour vous aider à comprendre mais vous n'êtes absolument pas obligés de retenir tous ces détails.

Retenir simplement qu'à partir de 2 broches numériques, il est possible de contrôler les 4 broches de phase d'un moteur pas à pas bipolaire.

21. Contrôle simple d'un moteur pas à pas (la trotteuse) : le programme

Ce qu'on va faire ici...

- Le programme que je vous propose est assez simple : ce programme va faire tourner le moteur pas à pas « cran par cran » à une vitesse lente (une demi-seconde entre chaque pas) à la façon d'une trotteuse de montre à aiguille.
- On utilise ici une interface Arduino Motor Shield v3 qui rappelons-le permet le contrôle de 2 moteurs CC à l'aide de 2 broches numériques de sens et de vitesse (PWM) pour chaque moteur.
- Au vu des explications présentées dans les pages précédentes, vous comprendrez qu'ici, nous allons :
 - utiliser seulement les 2 broches de sens pour contrôler le moteur pas à pas bipolaire
 - laisser les 2 broches de vitesse (PWM) au niveau HAUT car elles ne servent pas pour le contrôle du moteur pas à pas.

Entête déclarative

- On commence par inclure la librairie **Stepper** qui va nous permettre d'utiliser simplement le moteur pas à pas.
- On déclare ensuite une constante définissant le nombre de pas du moteur, à adapter à votre situation. Ici, moteur bipolaire de 200 pas.

Noter que si le pas vous est donné en degrés, faire $360 / \text{valeur pas}^\circ$ pour connaître le nombre de pas. Exemple : pas = $1,8^\circ$ donc $360 / 1,8^\circ = 200$ pas.

- On déclare ensuite toutes les broches utiles de l'interface utilisée. Dans mon cas, j'utilise le Arduino Motor Shield V3. Pour chaque phase, on a :
 - une broche **pwm** qui sera laissée à HIGH
 - une broche **dir** fixe la polarité de la phase (la seule utile)
 - une broche de **frein** inactivée au niveau LOW
 - une broche **analogique** de mesure de l'intensité.
- Une variable de pause entre 2 pas
- Un objet Stepper pour lequel on fixe le nombre de pas, et ici les 2 broches de contrôle de la polarité :

Stepper moteurPAP(nombre_pas, broche1, broche2); // déclare un objet Stepper contrôlé par 2 broches

- Noter que nous n'utilisons pas ici la forme de déclaration avec 4 broches qui est à utiliser seulement si vous utilisez une interface utilisant 2 broches de contrôle par phase.

Stepper moteurPAP(nombre_pas, broche1, broche2, broche3, broche4); // déclare un objet Stepper contrôlé par 4 broches

```
// --- Inclusion des librairies ---
#include <Stepper.h> // librairie pour moteurs pas à pas

// --- Déclaration des constantes utiles ---

const int NombrePas=200; // Nombre de pas du moteur pas à pas

// --- Déclaration des constantes des broches E/S numériques ---

//----- les broches du Motor Shield V3 -----
const int pwmPhaseA=3; // Constante pour la broche pwm phase A
const int dirPhaseA=12; // Constante pour la broche dir phase A
const int freinPhaseA=9; // Constante pour la broche frein phase A
const int intensitePhaseA=A0; // intensité de la phase A

const int pwmPhaseB=11; // Constante pour la broche pwm phase B
const int dirPhaseB=13; // Constante pour la broche dir phase B
const int freinPhaseB=8; // Constante pour la broche frein phase B
const int intensitePhaseB=A1; // intensité de la phase B

// --- Déclaration des constantes des broches analogiques ---

// --- Déclaration des variables globales ---

int pause=250; // delai entre 2 pas en ms

// --- Déclaration des objets utiles pour les fonctionnalités utilisées
---

Stepper stepper(NombrePas, dirPhaseA, dirPhaseB); // crée un objet
Stepper pour contrôler le moteur pas à pas avec 2 broches
```

Fonction **setup()**

Configuration des broches utilisées

- On commence par configurer en sortie toutes les broches de contrôle de chaque phase avec l'instruction **pinMode**(broche, sens), et ce pour les 2 phases.

Initialisation des broches de contrôle

- On met ici les broches pwm au niveau **HIGH** car elles ne sont pas utilisées pour le contrôle de la polarité. Pour mémoire, ces broches servent à fixer la vitesse dans le cas d'un moteur CC.
- On met au niveau **LOW** :
 - la broche dir de chaque phase,
 - la broche frein de chaque phase qui est ainsi inactive.
- Enfin, on fixe la vitesse angulaire de rotation du moteur en tours par minutes avec la fonction **setSpeed()**. Pas très important ici car on utilisera une pause entre chaque pas. Laisser une valeur moyenne entre 10 et 30 par exemple.

```
// ----- Broches en sorties numériques -----  
pinMode (pwmPhaseA,OUTPUT); // Broche vitesseMotA configurée en sortie  
pinMode (dirPhaseA,OUTPUT); // Broche sensMotA configurée en sortie  
pinMode (freinPhaseA,OUTPUT); // Broche freinMotA configurée en sortie  
  
pinMode (pwmPhaseB,OUTPUT); // Broche vitesseMotB configurée en sortie  
pinMode (dirPhaseB,OUTPUT); // Broche sensMotB configurée en sortie  
pinMode (freinPhaseB,OUTPUT); // Broche freinMotB configurée en sortie  
  
// ----- Broches en entrées numériques -----  
  
// ----- Activation si besoin du rappel au + (pullup) des broches en e  
ntrées numériques -----  
  
// ----- Initialisation des variables utilisées -----  
  
// ----- Codes d'initialisation utile -----  
digitalWrite(pwmPhaseA,HIGH); // inutilisée = laissée au niveau HAUT  
digitalWrite(dirPhaseA,LOW);  
digitalWrite(freinPhaseA,LOW); // frein off  
  
digitalWrite(pwmPhaseB,HIGH); // inutilisée = laissée au niveau HAUT  
digitalWrite(dirPhaseB,LOW);  
digitalWrite(freinPhaseB,LOW); // frein off  
  
// initialise la vitesse de rotation du moteur pas à pas en tour par min  
ute  
stepper.setSpeed(10); // fixe la vitesse d'exécution d'un pas à l'autre
```

Fonction `loop()`

- A l'aide d'une boucle `for()` on défile tous les pas 1 à 1 :
 - la fonction `step(1)` permet d'avancer de 1 pas à chaque fois
 - la fonction `delay()` permet de réaliser une pause entre chaque pas.
- On fait ensuite la même chose en sens inverse à l'aide de la fonction `step(-1)`

```
void loop(){ // debut de la fonction loop()

for (int i=1; i<=NombrePas; i++){ // boucle de défilement du nombre de pas
  stepper.step(1); // un pas en sens positif
  delay(pause);
}

delay (1000);

for (int i=1; i<=NombrePas; i++){ // boucle de défilement du nombre de pas
  stepper.step(-1); // un pas en sens négatif
  delay(pause);
}

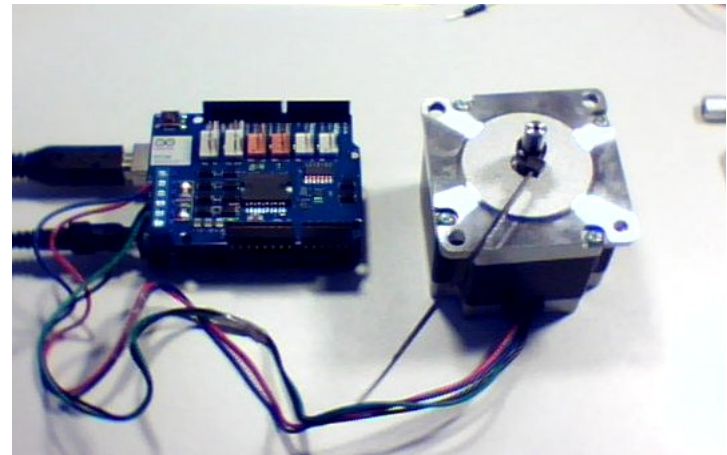
delay (1000);

} // fin de la fonction loop() - le programme recommence au début de la fonction loop sans fin
```

Fonctionnement du programme

- Hors tension, mettre le shield en place sur la carte Arduino et connecter les phases du moteur.
- Ensuite, connecter l'alimentation des moteurs
- Puis ensuite seulement connecter le port USB et programmer votre carte. Une fois fait, l'axe du moteur réalise un mouvement de trotteuse dans un sens puis sans l'autre.
- Sur le shield Arduino Motor Shield V3, remarquer les LEDs de visualisation qui montrent la polarité de chaque phase et donc la séquence des pas !

Bravo, vous savez contrôler un moteur pas à pas !



IMPORTANT

IL EST INDISPENSABLE D'ALIMENTER LA CARTE ARDUINO OU LE SHIELD AVEC UNE **ALIMENTATION EXTERNE 6 à 12V / 1 à 2 A (en fonction du moteur que vous utilisez)** à brancher dans le connecteur d'alimentation ou sur le bornier Vin du shield.

22. Truc technique... : une alimentation régulée de « labo » à pas cher !

Un point essentiel avec les moteurs pas à pas puissants : l'intensité de l'alimentation !

La principale difficulté lors de la mise en oeuvre des moteurs pas à pas est là encore d'ordre électrique, la programmation n'est pas plus difficile que ce que vous avez déjà vu. Chaque phase du moteur va consommer une intensité qui peut aller de 500mA jusqu'à 2A selon les modèles. Dans l'hypothèse d'un moteur pas à pas bipolaire en 12V avec 2 phases consommant chacun 1,5A par exemple, il va falloir fournir 3A minimum.

Le point crucial ici va donc être de pouvoir fournir une telle intensité à la tension voulue avec une alimentation adaptée.

Une solution simple et peu coûteuse est l'utilisation d'une alimentation de PC de récupération.

BON à savoir :

une alimentation de PC de récupération, dite alimentation ATX, est une alimentation régulée peu coûteuse (<10€) et qui fournit du 5V régulé sous plusieurs ampères, du 12V régulé sous plusieurs ampères, et aussi du 3.3V régulé, du -12V régulé, etc.. Pratique !

A comparer à un bloc secteur 220V AC / 6-12V DC qui ne fournira que 0.5 à 1A dans le meilleur des cas, pour le même prix...

Avec ça, vous êtes sur de pouvoir alimenter tous vos projets, y compris (et surtout) si vous utilisez des moteurs ou de nombreux servomoteurs !



Pour activer l'alimentation, au niveau du connecteur ATX, il suffit de mettre un strap entre la broche verte (16) et la masse (0V - noir)

23. Mouvement de va-et-vient avec un moteur pas à pas (« l'essuie-glace ») : le programme

Ce qu'on va faire ici...

- Le programme que je vous propose est assez simple mais va permettre de continuer de se familiariser avec le moteur pas à pas bipolaire.
- On utilise ici le même montage avec une interface Arduino Motor Shield v3 qui rappelons-le permet le contrôle de 2 moteurs CC à l'aide de 2 broches numériques de sens et de vitesse (PWM) pour chaque moteur.
- Au vu des explications présentées dans les pages précédentes, vous comprendrez qu'ici, nous allons à nouveau :
 - utiliser seulement les 2 broches de sens pour contrôler le moteur pas à pas bipolaire
 - laisser les 2 broches de vitesse (PWM) au niveau HAUT car elles ne servent pas pour le contrôle du moteur pas à pas.

Entête déclarative

- On commence par inclure la librairie **Stepper** qui va nous permettre d'utiliser simplement le moteur pas à pas.
- On déclare ensuite une constante définissant le nombre de pas du moteur, à adapter à votre situation. Ici, moteur bipolaire de 200 pas.

Noter que si le pas vous est donné en degrés, faire $360 / \text{valeur pas}^\circ$ pour connaître le nombre de pas. Exemple : pas = $1,8^\circ$ donc $360 / 1,8^\circ = 200$ pas.

- On déclare ensuite toutes les broches utiles de l'interface utilisée. Dans mon cas, j'utilise le Arduino Motor Shield V3. Pour chaque phase, on a :
 - une broche **pwm** qui sera laissée à **HIGH**
 - une broche **dir** fixe la polarité de la phase (la seule utile)
 - une broche de **frein** inactivée au niveau **LOW**
 - une broche **analogique** de mesure de l'intensité.
- Un objet Stepper pour lequel on fixe le nombre de pas, et ici les 2 broches de contrôle de la polarité :

Stepper moteurPAP(nombre_pas, broche1, broche2); // déclare un objet Stepper contrôlé par 2 broches

- Noter que nous n'utilisons pas ici la forme de déclaration avec 4 broches qui est à utiliser seulement si vous utilisez une interface utilisant 2 broches de contrôle par phase.

Stepper moteurPAP(nombre_pas, broche1, broche2, broche3, broche4); // déclare un objet Stepper contrôlé par 4 broches

```
// --- Inclusion des librairies ---
#include <Stepper.h> // librairie pour moteurs pas à pas

// --- Déclaration des constantes utiles ---

const int NombrePas=200; // Nombre de pas du moteur pas à pas

// --- Déclaration des constantes des broches E/S numériques ---

//----- les broches du Motor Shield V3 -----
const int pwmPhaseA=3; // Constante pour la broche pwm phase A
const int dirPhaseA=12; // Constante pour la broche dir phase A
const int freinPhaseA=9; // Constante pour la broche frein phase A
const int intensitePhaseA=A0; // intensité de la phase A

const int pwmPhaseB=11; // Constante pour la broche pwm phase B
const int dirPhaseB=13; // Constante pour la broche dir phase B
const int freinPhaseB=8; // Constante pour la broche frein phase B
const int intensitePhaseB=A1; // intensité de la phase B

// --- Déclaration des constantes des broches analogiques ---

// --- Déclaration des variables globales ---

// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---

Stepper stepper(NombrePas, dirPhaseA, dirPhaseB); // crée un objet Stepper pour contrôler le moteur pas à pas avec 2 broches
```


Fonction **setup()**

Configuration des broches utilisées

- On commence par configurer en sortie toutes les broches de contrôle de chaque phase avec l'instruction **pinMode**(broche, sens), et ce pour les 2 phases.

Initialisation des broches de contrôle

- On met ici les broches pwm au niveau **HIGH** car elles ne sont pas utilisées pour le contrôle de la polarité. Pour mémoire, ces broches servent à fixer la vitesse dans le cas d'un moteur CC.
- On met au niveau **LOW** :
 - la broche dir de chaque phase,
 - la broche frein de chaque phase qui est ainsi inactive.
- Enfin, on fixe la vitesse angulaire de rotation du moteur en tours par minutes avec la fonction **setSpeed()**. Laisser une valeur moyenne entre 10 et 30 par exemple. **Vous pourrez modifier cette valeur pour modifier la vitesse de votre « essuie-glace »...**

```
// ----- Broches en sorties numériques -----
pinMode (pwmPhaseA,OUTPUT); // Broche vitesseMotA configurée en sortie
pinMode (dirPhaseA,OUTPUT); // Broche sensMotA configurée en sortie
pinMode (freinPhaseA,OUTPUT); // Broche freinMotA configurée en sortie

pinMode (pwmPhaseB,OUTPUT); // Broche vitesseMotB configurée en sortie
pinMode (dirPhaseB,OUTPUT); // Broche sensMotB configurée en sortie
pinMode (freinPhaseB,OUTPUT); // Broche freinMotB configurée en sortie

// ----- Broches en entrées numériques -----

// ----- Activation si besoin du rappel au + (pullup) des broches en e
ntrées numériques -----

// ----- Initialisation des variables utilisées -----

// ----- Codes d'initialisation utile -----
digitalWrite(pwmPhaseA,HIGH); // inutilisée = laissée au niveau HAUT
digitalWrite(dirPhaseA,LOW);
digitalWrite(freinPhaseA,LOW); // frein off

digitalWrite(pwmPhaseB,HIGH); // inutilisée = laissée au niveau HAUT
digitalWrite(dirPhaseB,LOW);
digitalWrite(freinPhaseB,LOW); // frein off

// initialise la vitesse de rotation du moteur pas à pas en tour par min
ute
stepper.setSpeed(20); // fixe la vitesse d'exécution d'un pas à l'autre
```

Fonction **loop()**

- la fonction **step()** permet d'avancer du nombre de pas voulu en une fois en sens positif,
- puis la fonction **step()** permet d'avancer du nombre de pas voulu en une fois en sens négatif.
- noter que la vitesse utilisée est celle fixée précédemment avec la fonction **setSpeed()**

Remarquer avec quelle simplicité ce code vous permet d'obtenir le résultat voulu : 2 lignes suffisent !

```
void loop(){ // debut de la fonction loop()

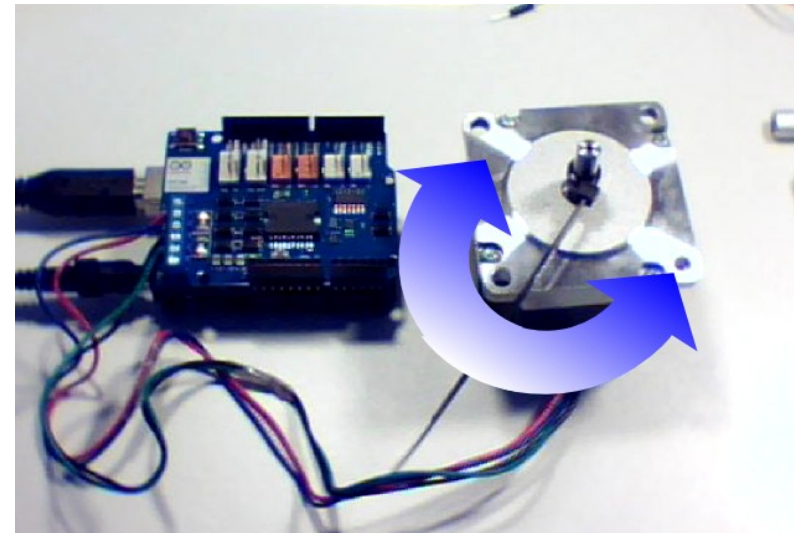
    stepper.step(NombrePas/2); // demicercle dans 1 sens
    stepper.step(-NombrePas/2); // demi-cercle dans l'autre sens

}
```

Fonctionnement du programme

- Hors tension, mettre le shield en place sur la carte Arduino et connecter les phases du moteur.
- Ensuite, connecter l'alimentation des moteurs
- Puis ensuite seulement connecter le port USB et programmer votre carte. Une fois fait, l'axe du moteur réalise un mouvement d'essuie-glace.
- Sur le shield Arduino Motor Shield V3, remarquer les LEDs de visualisation qui montrent la polarité de chaque phase et donc la séquence des pas !

Tester différentes valeurs pour la vitesse avec `setSpeed()` ou le nombre de pas avec `step()`. Sympa non ?



IMPORTANT

IL EST INDISPENSABLE D'ALIMENTER LA CARTE ARDUINO OU LE SHIELD AVEC UNE **ALIMENTATION EXTERNE 6 à 12V / 1 à 2 A (en fonction du moteur que vous utilisez)** à brancher dans le connecteur d'alimentation ou sur le bornier Vin du shield.

24. Positionnement par quarts de tours : « la « boussole »

Ce qu'on va faire ici...

- Allez, on continue sur notre lancée : afin de montrer la précision du positionnement du moteur pas à pas, je vous propose de réaliser un positionnement de l'axe par quart de tour : Nord, Est, Sud, Ouest, etc..
- Nous allons à nouveau :
 - utiliser seulement les 2 broches de sens pour contrôler le moteur pas à pas bipolaire
 - laisser les 2 broches de vitesse (PWM) au niveau HAUT car elles ne servent pas pour le contrôle du moteur pas à pas.

Entête déclarative

- On commence par inclure la librairie **Stepper** qui va nous permettre d'utiliser simplement le moteur pas à pas.
- On déclare ensuite une constante définissant le nombre de pas du moteur, à adapter à votre situation. Ici, moteur bipolaire de 200 pas.

Noter que si le pas vous est donné en degrés, faire $360 / \text{valeur pas}^\circ$ pour connaître le nombre de pas. Exemple : pas = $1,8^\circ$ donc $360 / 1,8^\circ = 200$ pas.

- On déclare ensuite toutes les broches utiles de l'interface utilisée. Dans mon cas, j'utilise le Arduino Motor Shield V3. Pour chaque phase, on a :
 - une broche **pwm** qui sera laissée à **HIGH**
 - une broche **dir** fixe la polarité de la phase (la seule utile)
 - une broche de **frein** inactivée au niveau **LOW**
 - une broche **analogique** de mesure de l'intensité.
- Un objet Stepper pour lequel on fixe le nombre de pas, et ici les 2 broches de contrôle de la polarité :

Stepper moteurPAP(nombre_pas, broche1, broche2); // déclare un objet Stepper contrôlé par 2 broches

- Noter que nous n'utilisons pas ici la forme de déclaration avec 4 broches qui est à utiliser seulement si vous utilisez une interface utilisant 2 broches de contrôle par phase.

Stepper moteurPAP(nombre_pas, broche1, broche2, broche3, broche4); // déclare un objet Stepper contrôlé par 4 broches

```
// --- Inclusion des librairies ---
#include <Stepper.h> // librairie pour moteurs pas à pas

// --- Déclaration des constantes utiles ---

const int NombrePas=200; // Nombre de pas du moteur pas à pas

// --- Déclaration des constantes des broches E/S numériques ---

//----- les broches du Motor Shield V3 -----
const int pwmPhaseA=3; // Constante pour la broche pwm phase A
const int dirPhaseA=12; // Constante pour la broche dir phase A
const int freinPhaseA=9; // Constante pour la broche frein phase A
const int intensitePhaseA=A0; // intensité de la phase A

const int pwmPhaseB=11; // Constante pour la broche pwm phase B
const int dirPhaseB=13; // Constante pour la broche dir phase B
const int freinPhaseB=8; // Constante pour la broche frein phase B
const int intensitePhaseB=A1; // intensité de la phase B

// --- Déclaration des constantes des broches analogiques ---

// --- Déclaration des variables globales ---

// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---

Stepper stepper(NombrePas, dirPhaseA, dirPhaseB); // crée un objet
Stepper pour contrôler le moteur pas à pas avec 2 broches
```

Fonction **setup()**

Configuration des broches utilisées

- On commence par configurer en sortie toutes les broches de contrôle de chaque phase avec l'instruction **pinMode**(broche, sens), et ce pour les 2 phases.

Initialisation des broches de contrôle

- On met ici les broches pwm au niveau **HIGH** car elles ne sont pas utilisées pour le contrôle de la polarité. Pour mémoire, ces broches servent à fixer la vitesse dans le cas d'un moteur CC.
- On met au niveau **LOW** :
 - la broche dir de chaque phase,
 - la broche frein de chaque phase qui est ainsi inactive.
- Enfin, on fixe la vitesse angulaire de rotation du moteur en tours par minutes avec la fonction **setSpeed()**. Laisser une valeur moyenne entre 10 et 30 par exemple.

```
// ----- Broches en sorties numériques -----  
pinMode (pwmPhaseA,OUTPUT); // Broche vitesseMotA configurée en sortie  
pinMode (dirPhaseA,OUTPUT); // Broche sensMotA configurée en sortie  
pinMode (freinPhaseA,OUTPUT); // Broche freinMotA configurée en sortie  
  
pinMode (pwmPhaseB,OUTPUT); // Broche vitesseMotB configurée en sortie  
pinMode (dirPhaseB,OUTPUT); // Broche sensMotB configurée en sortie  
pinMode (freinPhaseB,OUTPUT); // Broche freinMotB configurée en sortie  
  
// ----- Broches en entrées numériques -----  
  
// ----- Activation si besoin du rappel au + (pullup) des broches en e  
ntrées numériques -----  
  
// ----- Initialisation des variables utilisées -----  
  
// ----- Codes d'initialisation utile -----  
digitalWrite(pwmPhaseA,HIGH); // inutilisée = laissée au niveau HAUT  
digitalWrite(dirPhaseA,LOW);  
digitalWrite(freinPhaseA,LOW); // frein off  
  
digitalWrite(pwmPhaseB,HIGH); // inutilisée = laissée au niveau HAUT  
digitalWrite(dirPhaseB,LOW);  
digitalWrite(freinPhaseB,LOW); // frein off  
  
// initialise la vitesse de rotation du moteur pas à pas en tour par min  
ute  
stepper.setSpeed(20); // fixe la vitesse d'exécution d'un pas à l'autre
```

Fonction **loop()**

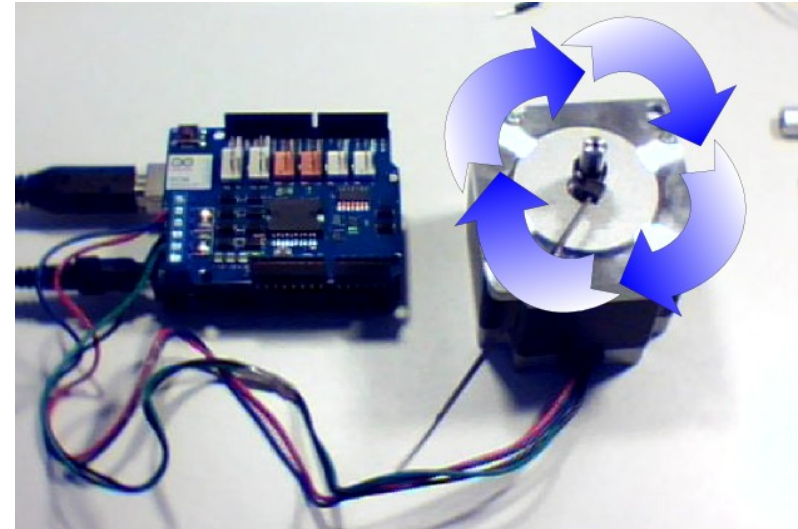
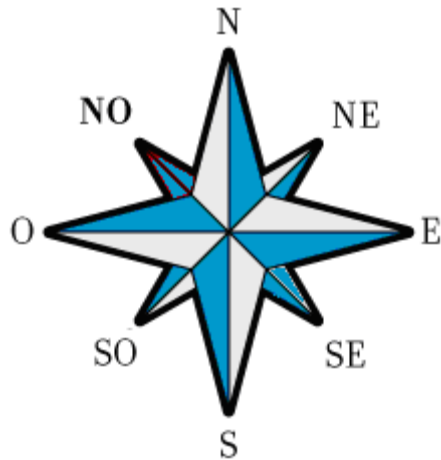
- à l'aide d'une fonction **for()**, on parcourt 4 positions :
- à chaque position la fonction **step()** permet d'avancer du nombre de pas voulu en une fois en sens positif,
- à chaque position, on réalise une pause de 1 seconde avec le fonction **delay()**

Remarquer une nouvelle fois avec quelle simplicité ce code vous permet d'obtenir le résultat voulu : 3 lignes suffisent !

```
void loop(){ // debut de la fonction loop()  
  
for (int i=0; i<=4; i++) { // 4 positions  
  
    stepper.step(NombrePas/4);  
    delay (1000);  
  
} // fin for  
  
} // fin de la fonction loop()
```

Fonctionnement du programme

- Hors tension, mettre le shield en place sur la carte Arduino et connecter les phases du moteur.
- Ensuite, connecter l'alimentation des moteurs
- Puis ensuite seulement connecter le port USB et programmer votre carte. Une fois fait, l'axe du moteur réalise un **positionnement en 4 position de 90° successives**.



Essayez avec 8 positions intermédiaires : voyez comme la précision est au rendez-vous !

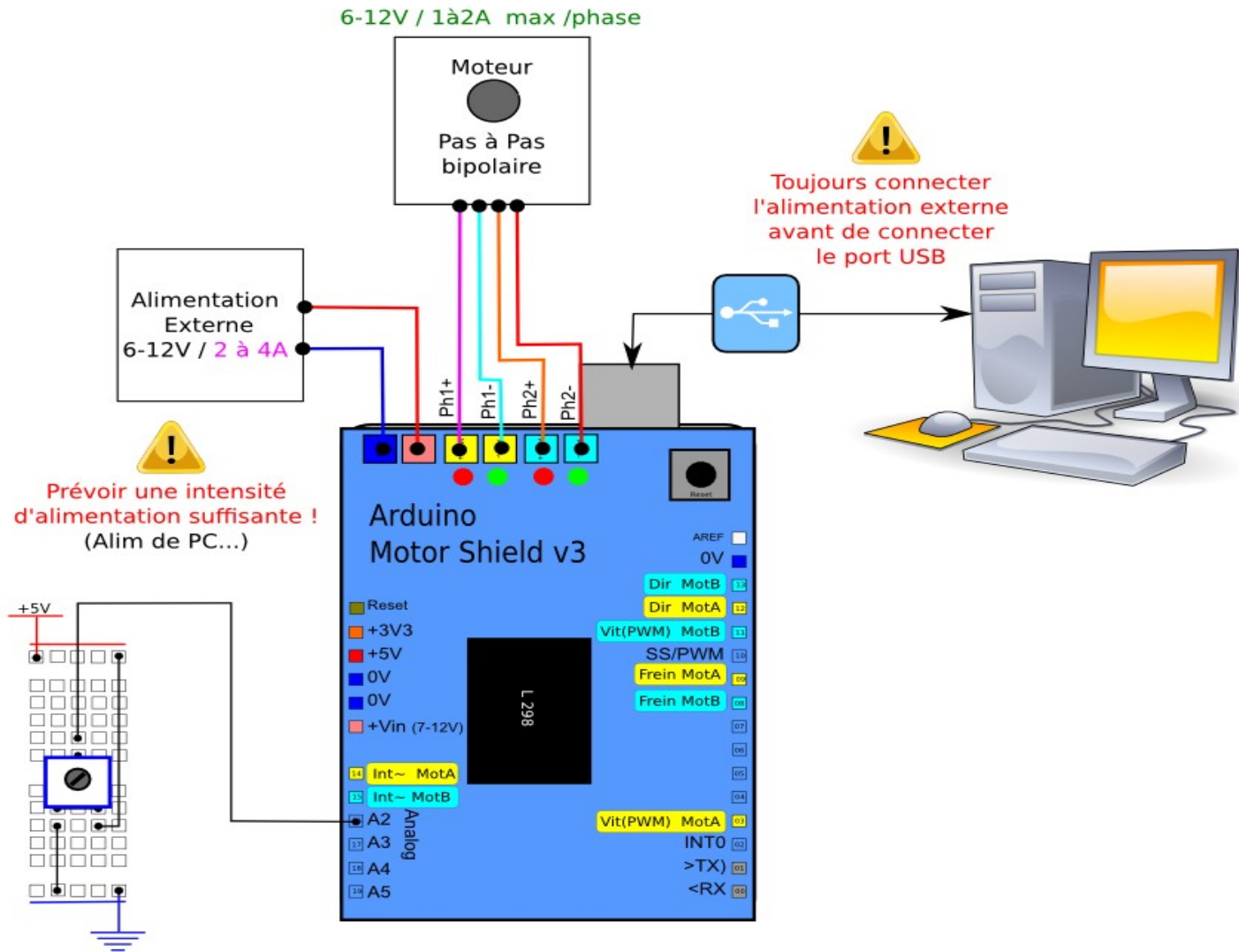
IMPORTANT

IL EST INDISPENSABLE D'ALIMENTER LA CARTE ARDUINO OU LE SHIELD AVEC UNE **ALIMENTATION EXTERNE 6 à 12V / 1 à 2 A (en fonction du moteur que vous utilisez)** à brancher dans le connecteur d'alimentation ou sur le bornier Vin du shield.

25. Contrôler la position d'un moteur pas à pas à l'aide d'une résistance variable : le montage

A présent, nous allons utiliser une résistance variable pour contrôler le moteur pas à pas : pour fixer la position dans un premier temps, pour fixer la vitesse de rotation dans un second temps. Le montage est ici très simple :

- enficher le Arduino Motor Shield V3 sur la carte Arduino
- connecter la sortie de la résistance variable sur la broche analogique A2 (la 3ème car les 2 premières, la A0 et la A1 sont utilisées par le shield)



26. Contrôler la position d'un moteur pas à pas à l'aide d'une résistance variable : le programme

Ce qu'on va faire ici...

- A présent, en gardant toujours le même principe pour le contrôle du moteur pas à pas, nous allons contrôler la position à l'aide d'une résistance variable. Le principe est simple :
 - la mesure de la tension en sortie de la résistance variable réalisée sur la broche analogique va évoluer de 0 (0V) à 1023 (5V) (voir l'atelier dédié à la Conversion analogique-numérique si vous ne comprenez pas cela...)
 - on va corrélérer cette valeur aux pas du moteur avec pas n° 1 pour 0 et pas n°200 pour 5V (si on utilise un moteur 200 pas)
- Nous allons à nouveau utiliser seulement les 2 broches de sens pour contrôler le moteur pas à pas bipolaire et laisser les 2 broches de vitesse (PWM) au niveau HAUT car elles ne servent pas pour le contrôle du moteur pas à pas.

Entête déclarative

- On commence par inclure la librairie **Stepper** qui va nous permettre d'utiliser simplement le moteur pas à pas.
- On déclare ensuite une constante définissant le nombre de pas du moteur, à adapter à votre situation. Ici, moteur bipolaire de 200 pas.

Noter que si le pas vous est donné en degrés, faire $360 / \text{valeur pas}^\circ$ pour connaître le nombre de pas. Exemple : pas = $1,8^\circ$ donc $360 / 1,8^\circ = 200$ pas.

- On déclare la broche utilisée pour la mesure analogique.
- On déclare ensuite toutes les broches utiles de l'interface utilisée. Dans mon cas, j'utilise le Arduino Motor Shield V3. Pour chaque phase, on a :
 - une broche **pwm** qui sera laissée à **HIGH**
 - une broche **dir** fixe la polarité de la phase (la seule utile)
 - une broche de **frein** inactivée au niveau **LOW**
 - une broche **analogique** de mesure de l'intensité.
- On déclare plusieurs variables utilisées par le programme pour gérer la position du moteur pas à pas : une variable de mesure, 2 variables de mémorisation de l'indice du pas courant et cible, une variable de calcul de l'écart entre le pas courant et le pas cible.
- Un objet Stepper pour lequel on fixe le nombre de pas, et ici les 2 broches de contrôle de la polarité :

```
Stepper moteurPAP(nombre_pas, broche1, broche2); // déclare un objet Stepper contrôlé par 2 broches
```

- Noter que nous n'utilisons pas ici la forme de déclaration avec 4 broches qui est à utiliser seulement si vous utilisez une interface utilisant 2 broches de contrôle par phase.

```
Stepper moteurPAP(nombre_pas, broche1, broche2, broche3, broche4); // déclare un objet Stepper contrôlé par 4 broches
```

```
// --- Inclusion des librairies ---
#include <Stepper.h> // librairie pour moteurs pas à pas

// --- Déclaration de constantes utiles
const int NombrePas=200; // Nombre de pas du moteur pas à pas
const int RVar=A2; // broche analogique utilisée pour la résistance variable

//----- les broches du Motor Shield V3 -----
const int pwmPhaseA=3; // Constante pour la broche pwm phase A
const int dirPhaseA=12; // Constante pour la broche dir phase A
const int freinPhaseA=9; // Constante pour la broche frein phase A
const int intensitePhaseA=A0; // intensité de la phase A

const int pwmPhaseB=11; // Constante pour la broche pwm phase B
const int dirPhaseB=13; // Constante pour la broche dir phase B
const int freinPhaseB=8; // Constante pour la broche frein phase B
const int intensitePhaseB=A1; // intensité de la phase B

// --- Déclaration des variables globales ---

int mesure=0; // variable pour mesure analogique

int indicePas=0; // variable d'indice du pas courant
int indicePas0=0; // variable d'indice du dernier pas

int ecartPas=0; // variable de mesure de l'ecart entre les pas

// --- Déclaration des objets utiles pour les fonctionnalités utilisées ---

Stepper stepper(NombrePas, dirPhaseA, dirPhaseB); // crée un objet Stepper pour contrôler le moteur pas à pas avec 2 broches
```

Fonction **setup()**

Configuration des broches utilisées

- On commence par configurer en sortie toutes les broches de contrôle de chaque phase avec l'instruction **pinMode**(broche, sens), et ce pour les 2 phases.

Initialisation des broches de contrôle

- On met ici les broches pwm au niveau **HIGH** car elles ne sont pas utilisées pour le contrôle de la polarité. Pour mémoire, ces broches servent à fixer la vitesse dans le cas d'un moteur CC.
- On met au niveau **LOW** :
 - la broche dir de chaque phase,
 - la broche frein de chaque phase qui est ainsi inactive.
- Enfin, on fixe la vitesse angulaire de rotation du moteur en tours par minutes avec la fonction **setSpeed()**. On utilise ici une valeur de 70 pour un changement rapide de position.

```
// ----- Broches en sorties numériques -----
pinMode (pwmPhaseA,OUTPUT); // Broche vitesseMotA configurée en sortie
pinMode (dirPhaseA,OUTPUT); // Broche sensMotA configurée en sortie
pinMode (freinPhaseA,OUTPUT); // Broche freinMotA configurée en sortie

pinMode (pwmPhaseB,OUTPUT); // Broche vitesseMotB configurée en sortie
pinMode (dirPhaseB,OUTPUT); // Broche sensMotB configurée en sortie
pinMode (freinPhaseB,OUTPUT); // Broche freinMotB configurée en sortie

// ----- Broches en entrées numériques -----

// ----- Activation si besoin du rappel au + (pullup) des broches en e
ntrées numériques -----

// ----- Initialisation des variables utilisées -----

// ----- Codes d'initialisation utile -----
digitalWrite(pwmPhaseA,HIGH); // inutilisée = laissée au niveau HAUT
digitalWrite(dirPhaseA,LOW);
digitalWrite(freinPhaseA,LOW); // frein off

digitalWrite(pwmPhaseB,HIGH); // inutilisée = laissée au niveau HAUT
digitalWrite(dirPhaseB,LOW);
digitalWrite(freinPhaseB,LOW); // frein off

// initialise la vitesse de rotation du moteur pas à pas en tour par min
ute
stepper.setSpeed(70); // fixe la vitesse d'exécution d'un pas à l'autre
```

Fonction `loop()`

- on commence par mesurer la valeur analogique sur la broche analogique où est connectée la résistance variable, à l'aide de la fonction `analogRead()`.
- ensuite on calcule l'indice du pas correspondant tel que pas 0 pour 0V et pas `NombrePas-1` pour 5V, à l'aide de la fonction `map()`,
- on calcule l'écart entre cette valeur et la position courante du pas.
- on teste de l'écart à l'aide de 2 conditions `if()` (en fait une seule aurait pu suffire, mais bon..) puis :
 - on positionne le moteur pas à pas du nombre de pas voulu
 - et on mémorise la nouvelle valeur de l'indice du pas courant.

C'est un petit peu plus compliqué que les programmes précédents, mais rien d'inaccessible à ce stade pour vous normalement...

```
void loop(){ // debut de la fonction loop()

    mesure=analogRead(RVar); // lit la valeur de résistance variable

    indicePas=map(mesure, 0,1023,0,NombrePas-1); // ré-échelonne mesure
    analogique (0-1023) => nombre de pas (0- 199)

    ecartPas=indicePas-indicePas0; // calcul écart entre pas courant et
    pas cible

    //----- si ecart positif -----
    if (ecartPas>0) { // si la différence nombre pas est positive

        stepper.step(ecartPas); // avance du nombre de pas voulu

        indicePas0=indicePas; // mémorise le nouveau pas courant

    } // fin if

    //----- si ecart positif -----
    if (ecartPas<0) { // si la différence nombre pas est positive

        stepper.step(ecartPas); // avance (et donc recule car négatif) du
        nombre de pas voulu

        indicePas0=indicePas; // mémorise le nouveau pas courant

    } // fin if

} // fin de la fonction loop()
```

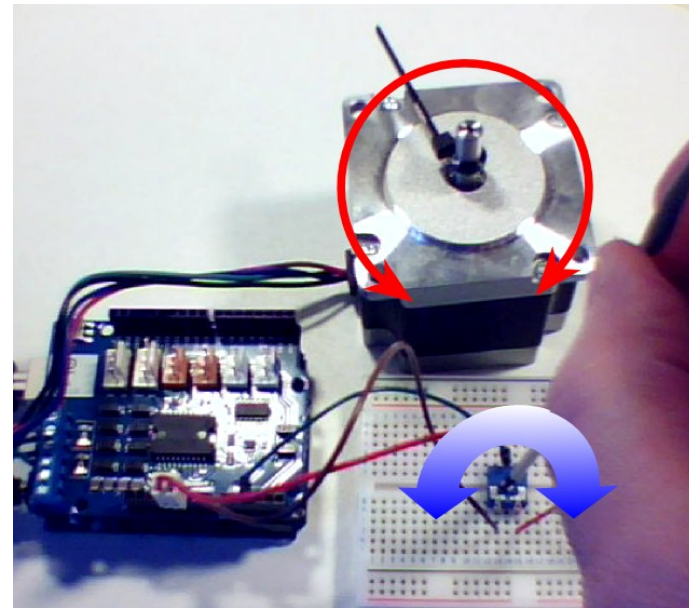
Fonctionnement du programme

- Hors tension, mettre le shield en place sur la carte Arduino et connecter les phases du moteur.
- Ensuite, connecter l'alimentation des moteurs
- Puis ensuite seulement connecter le port USB et programmer votre carte.
- Une fois fait, tourner la résistance variable : l'axe du moteur pas à pas défile du 1er au dernier pas en fonction de la position de l'axe de la résistance variable entre 0V et 5V.

Le moteur pas à pas est ainsi transformé
en une sorte de servomoteur à 360° !



```
if (ecartPas>0) { stepper.step(ecartPas) ...
```



IMPORTANT

IL EST INDISPENSABLE D'ALIMENTER LA CARTE ARDUINO OU LE SHIELD AVEC UNE **ALIMENTATION EXTERNE 6 à 12V / 1 à 2 A (en fonction du moteur que vous utilisez)** à brancher dans le connecteur d'alimentation ou sur le bornier Vin du shield.

27. Contrôler la vitesse d'un moteur pas à pas à l'aide d'une résistance variable : le programme

Ce qu'on va faire ici...

- Bon, parti comme on est parti, ça ne va pas être très difficile de contrôler la vitesse du moteur avec la résistance variable :
 - la mesure de la tension en sortie de la résistance variable réalisée sur la broche analogique va évoluer de 0 (0V) à 1023 (5V) (voir l'atelier dédié à la Conversion analogique-numérique si vous ne comprenez pas cela...)
 - on va corréliser cette valeur à la vitesse du moteur : arrêt pour 0V, vitesse progressivement plus rapide pour un maximum à 5V.
- Nous allons à nouveau utiliser seulement les 2 broches de sens pour contrôler le moteur pas à pas bipolaire et laisser les 2 broches de vitesse (PWM) au niveau HAUT car elles ne servent pas pour le contrôle du moteur pas à pas.

Entête déclarative

- On commence par inclure la librairie **Stepper** qui va nous permettre d'utiliser simplement le moteur pas à pas.
- On déclare ensuite une constante définissant le nombre de pas du moteur, à adapter à votre situation. Ici, moteur bipolaire de 200 pas.

Noter que si le pas vous est donné en degrés, faire $360 / \text{valeur pas}^\circ$ pour connaître le nombre de pas. Exemple : pas = $1,8^\circ$ donc $360 / 1,8^\circ = 200$ pas.

- On déclare la broche utilisée pour la mesure analogique.
- On déclare ensuite toutes les broches utiles de l'interface utilisée. Dans mon cas, j'utilise le Arduino Motor Shield V3. Pour chaque phase, on a :
 - une broche **pwm** qui sera laissée à **HIGH**
 - une broche **dir** fixe la polarité de la phase (la seule utile)
 - une broche de **frein** inactivée au niveau **LOW**
 - une broche **analogique** de mesure de l'intensité.
- On déclare plusieurs variables utilisées par le programme pour gérer la vitesse du moteur pas à pas : une variable de mesure, 3 variables de vitesse, la vitesse courante, la vitesse cible et la vitesse Max.
- Un objet Stepper pour lequel on fixe le nombre de pas, et ici les 2 broches de contrôle de la polarité :

Stepper moteurPAP(nombre_pas, broche1, broche2); // déclare un objet Stepper contrôlé par 2 broches

- Noter que nous n'utilisons pas ici la forme de déclaration avec 4 broches qui est à utiliser seulement si vous utilisez une interface utilisant 2 broches de contrôle par phase.

Stepper moteurPAP(nombre_pas, broche1, broche2, broche3, broche4); // déclare un objet Stepper contrôlé par 4 broches

```
// --- Inclusion des librairies ---
#include <Stepper.h> // librairie pour moteurs pas à pas

// --- Déclaration de constantes utiles
const int NombrePas=200; // Nombre de pas du moteur pas à pas
const int RVar=A2; // broche analogique utilisée pour la résistance variable

//----- les broches du Motor Shield V3 -----
const int pwmPhaseA=3; // Constante pour la broche pwm phase A
const int dirPhaseA=12; // Constante pour la broche dir phase A
const int freinPhaseA=9; // Constante pour la broche frein phase A
const int intensitePhaseA=A0; // intensité de la phase A

const int pwmPhaseB=11; // Constante pour la broche pwm phase B
const int dirPhaseB=13; // Constante pour la broche dir phase B
const int freinPhaseB=8; // Constante pour la broche frein phase B
const int intensitePhaseB=A1; // intensité de la phase B

// --- Déclaration des variables globales ---

int mesure=0; // variable pour mesure analogique

int vitesse=0; // variable pour la vitesse
int vitesse0=0; // variable pour mémoriser la vitesse courante
int vitesseMax=100; // variable pour la vitesse maximale possible du moteur
//-- à adapter selon moteur... si le moteur "rate" des pas, la valeur est trop élevée...
```

Fonction **setup()**

Configuration des broches utilisées

- On commence par configurer en sortie toutes les broches de contrôle de chaque phase avec l'instruction **pinMode**(broche, sens), et ce pour les 2 phases.

Initialisation des broches de contrôle

- On met ici les broches pwm au niveau **HIGH** car elles ne sont pas utilisées pour le contrôle de la polarité. Pour mémoire, ces broches servent à fixer la vitesse dans le cas d'un moteur CC.
- On met au niveau **LOW** :
 - la broche dir de chaque phase,
 - la broche frein de chaque phase qui est ainsi inactive.
- Enfin, on fixe la vitesse angulaire de rotation du moteur en tours par minutes avec la fonction **setSpeed()**. On utilise ici une valeur de 0 pour avoir moteur à l'arrêt au départ.

```
// ----- Broches en sorties numériques -----  
pinMode (pwmPhaseA,OUTPUT); // Broche vitesseMotA configurée en sortie  
pinMode (dirPhaseA,OUTPUT); // Broche sensMotA configurée en sortie  
pinMode (freinPhaseA,OUTPUT); // Broche freinMotA configurée en sortie  
  
pinMode (pwmPhaseB,OUTPUT); // Broche vitesseMotB configurée en sortie  
pinMode (dirPhaseB,OUTPUT); // Broche sensMotB configurée en sortie  
pinMode (freinPhaseB,OUTPUT); // Broche freinMotB configurée en sortie  
  
// ----- Broches en entrées numériques -----  
  
// ----- Activation si besoin du rappel au + (pullup) des broches en e  
ntrées numériques -----  
  
// ----- Initialisation des variables utilisées -----  
  
// ----- Codes d'initialisation utile -----  
digitalWrite(pwmPhaseA,HIGH); // inutilisée = laissée au niveau HAUT  
digitalWrite(dirPhaseA,LOW);  
digitalWrite(freinPhaseA,LOW); // frein off  
  
digitalWrite(pwmPhaseB,HIGH); // inutilisée = laissée au niveau HAUT  
digitalWrite(dirPhaseB,LOW);  
digitalWrite(freinPhaseB,LOW); // frein off  
  
// initialise la vitesse de rotation du moteur pas à pas en tour par min  
ute  
stepper.setSpeed(0); // fixe la vitesse d'exécution d'un pas à l'autre
```


Fonction `loop()`

- on commence par mesurer la valeur analogique sur la broche analogique où est connectée la résistance variable, à l'aide de la fonction `analogRead()`.
- ensuite on calcule vitesse correspondante telle que `vitesse=0` pour 0V et `vitesse=vitesseMax` pour 5V, à l'aide de la fonction `map()`,
- si la vitesse change par rapport à la valeur courante, et n'est pas nulle, on change la valeur de la vitesse du moteur pas à pas à l'aide de la fonction `setSpeed()`
- dans tous les cas, si la vitesse n'est pas nulle, on avance d'un pas.

Simple non ?

Vous me direz qu'il n'y a pas forcément d'intérêt à régler la vitesse d'un moteur pas à pas en rotation continue, mais le but ici est essentiellement didactique.

```
void loop(){ // debut de la fonction loop()

    mesure=analogRead(RVar); // lit la valeur de résistance variable

    vitesse=map(mesure,0,1023,0,vitesseMax); // calcule la nouvelle
    vitesse

    if ( (vitesse!=vitesse0) && (vitesse>=1) ) { // si la vitesse est
    modifiée et non nulle

        stepper.setSpeed(vitesse); // fixe la nouvelle vitesse d'exécution
    d'un pas à l'autre si pas négative
        vitesse0=vitesse; // mémorise la nouvelle vitesse courante

    } // fin if

    if (vitesse>=1) stepper.step(1); // avance de 1 pas si vitesse non
    nulle - sinon arrêt

} // fin de la fonction loop()
```

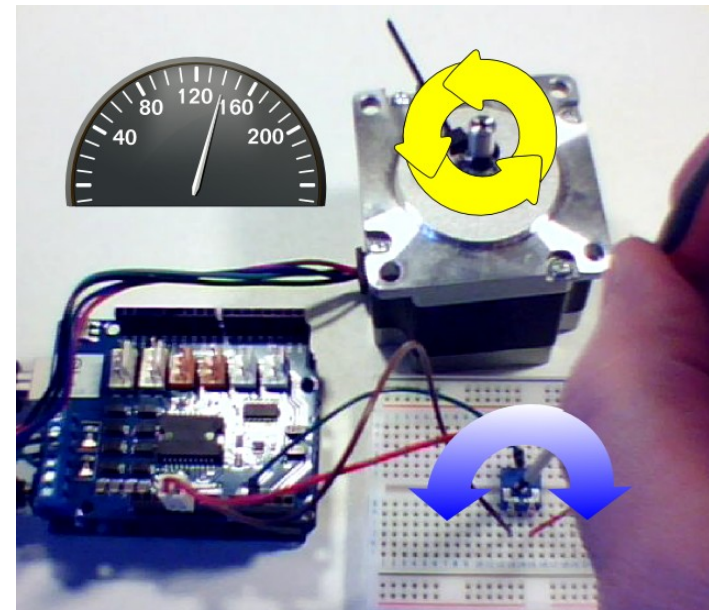
Fonctionnement du programme

- Hors tension, mettre le shield en place sur la carte Arduino et connecter les phases du moteur.
- Ensuite, connecter l'alimentation des moteurs
- Puis ensuite seulement connecter le port USB et programmer votre carte.
- Une fois fait, **tourner la résistance variable** : la vitesse du moteur évolue de l'arrêt à la rotation maximale en fonction de la position de l'axe de la résistance variable entre 0V et 5V.

Le moteur pas à pas se comporte ici finalement comme un moteur CC.



« Un as, vous allez devenir, un as je vous dis !... »



IMPORTANT

IL EST INDISPENSABLE D'ALIMENTER LA CARTE ARDUINO OU LE SHIELD AVEC UNE **ALIMENTATION EXTERNE 6 à 12V / 1 à 2 A (en fonction du moteur que vous utilisez)** à brancher dans le connecteur d'alimentation ou sur le bornier Vin du shield.

28. Contrôler la vitesse d'un moteur pas à pas dans les 2 sens avec une résistance variable : le programme

Ce qu'on va faire ici...

- Je vous propose une variante du programme précédent : exactement la même chose... mais dans les 2 sens. Seule la fonction `loop()` change...

Fonction `loop()`

- on commence une nouvelle fois par mesurer la valeur analogique sur la broche analogique où est connectée la résistance variable, à l'aide de la fonction `analogRead()`.
- ensuite on calcule vitesse correspondante telle que `vitesse=-vitesseMax` pour 0V, `vitesse=0` pour 2,5V et `vitesse=vitesseMax` pour 5V, à l'aide de la fonction `map()`,
- si la vitesse change par rapport à la valeur courante, et n'est pas nulle, on change la valeur de la vitesse du moteur pas à pas à l'aide de la fonction `setSpeed()` en utilisant ici la valeur absolue de la vitesse (qui peut être ici soit négative, soit positive...)
- dans tous les cas, si la vitesse n'est pas nulle :
 - on avance d'un pas, si la vitesse est positive
 - on recule d'un pas, si la vitesse est négative

```
void loop(){ // debut de la fonction loop()

    mesure=analogRead(RVar); // lit la valeur de résistance variable

    vitesse=map(mesure,0,1023,-vitesseMax,vitesseMax); // calcule la
nouvelle vitesse

    if ( (vitesse!=vitesse0) && (abs(vitesse)>=1) ) { // si la vitesse
est modifiée et non nulle

        stepper.setSpeed(abs(vitesse)); // fixe la nouvelle vitesse
d'exécution d'un pas à l'autre si pas négative
        vitesse0=vitesse; // mémorise la nouvelle vitesse courante

    } // fin if

    if (vitesse>=1) stepper.step(1); // avance de 1 pas si vitesse non
nulle - sinon arrêt
    if (vitesse<=1) stepper.step(-1); // recule de 1 pas si vitesse non
nulle - sinon arrêt

} // fin de la fonction loop()
```

Fonctionnement du programme

- Ici, tourner la résistance variable : la vitesse du moteur évolue de la vitesse maximale dans un sens en passant par l'arrêt à la rotation puis jusqu'à la vitesse maximale dans l'autre sens en fonction de la position de l'axe de la résistance variable entre 0V et 5V.

29. Les éléments du langage Arduino étudiés dans cet atelier

Les instructions de la librairie Stepper pour le contrôle des moteurs pas à pas :

- Constructeur `Stepper`
- `setSpeed ()`
- `step()`

La documentation complète du langage Arduino en français est disponible ici :
http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.ReferenceMaxi

30. *A présent, vous devriez être capable :*

- D'expliquer le fonctionnement d'un moteur pas à pas bipolaire
- D'utiliser une interface pour contrôler un moteur pas à pas bipolaire en mode « full step »

Table des matières

Moteurs : Apprendre à utiliser un moteur pas à pas bipolaire (en mode « full-step ») avec une carte Arduino.

Intro |

Matériel nécessaire pour les ateliers Arduino |

Matériel spécifique nécessaire pour cet atelier |

Rappel : Technique : l'alimentation de la carte Arduino |

Les moteurs pas à pas : concrètement |

Les moteurs pas à pas bipolaires : fiche technique. |

Pour comprendre : Les moteurs pas à pas bipolaires : structure interne et principe de fonctionnement |

Les moteurs pas à pas bipolaires : schéma électrique type d'utilisation avec Arduino |

Les moteurs pas à pas bipolaires : montage type avec une carte Arduino |

Les moteurs pas à pas : exemples d'utilisation |

Les moteurs pas à pas bipolaires : principe de contrôle à l'aide d'un « double-pont en H » |

Les moteurs pas à pas bipolaires : principe de contrôle des pas |

Mouvement du moteur pas à pas réel |

Les moteurs pas à pas bipolaires : séquences de contrôle des pas |

Moteurs pas à pas bipolaires : interfaces de contrôle |

Langage Arduino : Introduction aux bibliothèques |

Langage Arduino : la bibliothèque Stepper pour le contrôle des moteurs pas à pas |

Contrôle simple d'un moteur pas à pas (la trotteuse) : le montage |

Contrôle simple d'un moteur pas à pas (la trotteuse) : note technique sur les interfaces « directes » |

Contrôle simple d'un moteur pas à pas (la trotteuse) : la séquence de contrôle direct avec 2 broches |

Contrôle simple d'un moteur pas à pas (la trotteuse) : le programme |

Truc technique... : une alimentation régulée de « labo » à pas cher ! |

Mouvement de va-et-vient avec un moteur pas à pas (« l'essuie-glace ») : le programme |

Positionnement par quarts de tours : « la « boussole » |

Contrôler la position d'un moteur pas à pas à l'aide d'une résistance variable : le montage |

Contrôler la position d'un moteur pas à pas à l'aide d'une résistance variable : le programme |

Contrôler la vitesse d'un moteur pas à pas à l'aide d'une résistance variable : le programme |

Contrôler la vitesse d'un moteur pas à pas dans les 2 sens avec une résistance variable : le programme |

Les éléments du langage Arduino étudiés dans cet atelier |

A présent, vous devriez être capable : |

Bravo !
vous avez terminé cet atelier Arduino !



Prêt pour la suite ? Retrouvez de nombreux autres thèmes d'ateliers Arduino ici :

http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS

