Commande d'un moteur pas à pas unipolaire sur Arduino

1. Introduction

La commande des moteurs pas à pas bipolaires est expliquée dans Commande d'un moteur pas à pas bipolaire sur Arduino. Les deux bobines d'un moteur bipolaire sont alimentées avec des courants qui doivent changer de signe alternativement, ce qui nécessite l'emploi de deux ponts en H.

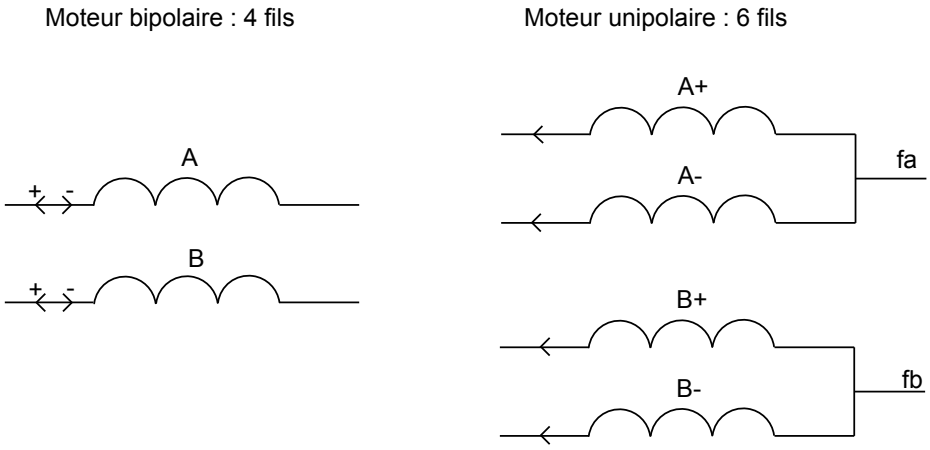
Le moteur pas à pas unipolaire comporte 4 bobines (deux pour chaque pôle du stator), toutes alimentées dans le même sens. La commande de ces moteurs est donc unipolaire (d'où le nom donné à ce type de moteur). Chaque bobine peut être pilotée à partir d'une sortie numérique par un seul transistor. Ce document montre commment commander un moteur pas à pas unipolaire avec un réseau de transistors Darlington.

Les moteurs unipolaires sont plus faciles à piloter que les moteurs bipolaires, mais il y a moins de modèles disponibles sur le marché (pour les petits moteurs).

2. Principe du moteur unipolaire

Pour le principe du moteur pas à pas, voir Commande d'un moteur pas à pas bipolaire sur Arduino.

Dans un moteur bipolaire, chaque pôle du stator est constitué d'une seule bobine, et nécessite donc deux fils d'alimentation. Dans un moteur pas à pas *unipolaire* ([1]), chaque pôle est constituée de deux bobines enroulées en sens inverse sur les pôles du stator. Pour changer le sens du champ magnétique dans un pôle, il faut alimenter l'une ou l'autre de ces deux bobines. Voici le schéma des bobines d'un moteur bipolaire et d'un moteur unipolaire :



Le sens du courant est aussi représenté. Dans les bobines du moteur unipolaire, le sens du courant est toujours le même, alors qu'il change alternativement dans celles du moteur bipolaire. Les deux bobines d'un même pôle (A ou B) ne doivent pas être alimentées en même temps, car elles produisent deux champs opposés.

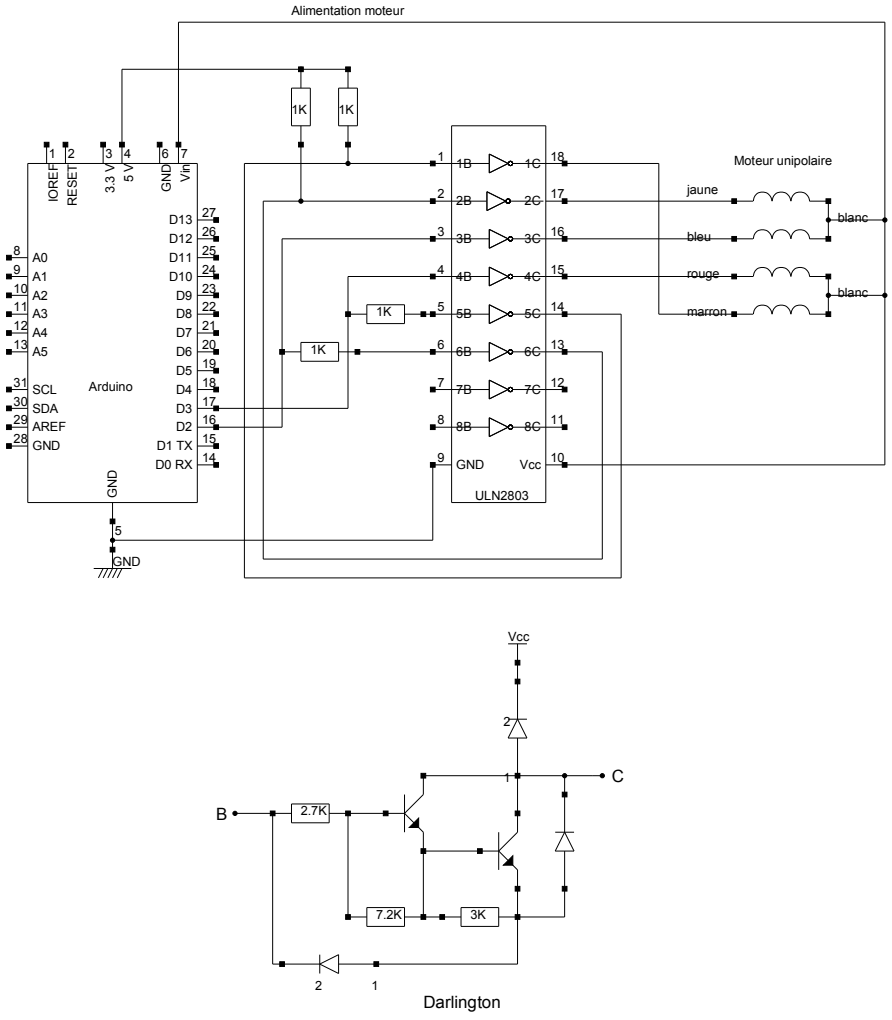
Un moteur unipolaire comporte 6 fils. Les deux fils fa et fb doivent être reliés à la borne positive de l'alimentation. Les 4 autres fils sont généralement reliés au collecteur d'un transistor de commande. Les fils fa et fb ont la même couleur. Pour repérer les autres bornes des bobines, il faut utiliser un ohmmètre.

3. Circuit de commande

La commande d'une bobine est unipolaire (le sens du courant est toujours le même). Elle peut donc se faire avec un seul transistor, par exemple un transistor Darlington, comme la commande d'un moteur à courant continu tournant toujours dans le même sens.

Pour commander les 4 bobines, on utilise 4 transistors d'un réseau de Darlington intégré, par exemple le ULN2803. Ce circuit peut délivrer jusqu'à 500 mA par transistor, soit 500 mA par phase du moteur.

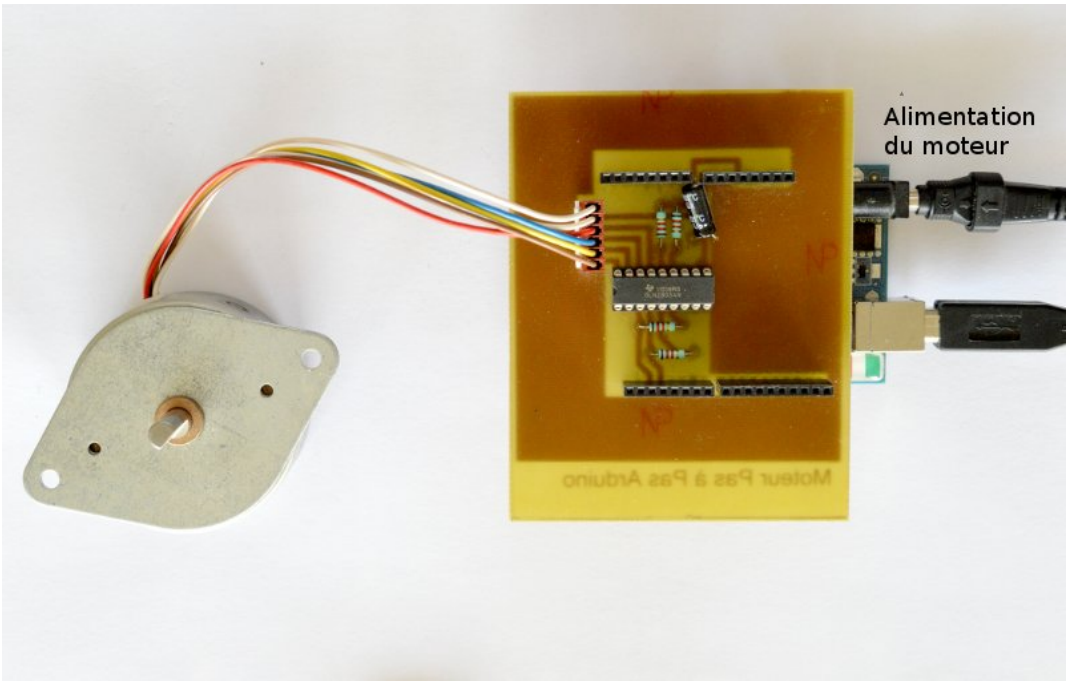
Voici le schéma d'un circuit avec l'arduino ([2]), avec un détail d'un Darlington (entrée B, sortie C).



Un niveau logique haut (5 volts) sur la base B du Darlington fait passer le collecteur à une tension proche de zéro (transistor saturé), c'est pourquoi le Darlington est représenté par

un inverseur logique. Ce niveau haut a pour effet d'alimenter la bobine correspondante, alors qu'un niveau bas bloque le transistor et coupe le courant dans la bobine. On remarque que les diodes de roue libre sont intégrées dans l'ULN2803. Les deux bobines d'une même phase sont commandées par deux signaux inverses, ce qui permet de commander le moteur avec seulement deux sorties de l'arduino, une pour chaque paire de bobines. Cela est à comparer à la commande d'un moteur bipolaire par un double pont en H, qui nécessite 4 sorties de l'arduino. Les transistors Darlington fonctionnent en commutation, ce qui fait qu'ils dissipent très peu d'énergie.

Les couleurs indiquées sur le schéma sont celles du moteur utilisé par l'auteur. La résistance entre les bornes jaune et bleue est de 70 Ω, alors qu'elle est de 35 Ω entre les bornes blanche et jaune. Voici une photographie du montage complet. L'alimentation du moteur est fournie par un bloc secteur de 12 V. Le moteur est alimenté par la platine via la borne Vin de l'arduino. L'alimentation peut donc aussi servir à l'arduino, lorsqu'il n'est pas relié par USB à un ordinateur.



Le moteur utilisé consomme 300 mA par phase pour une tension de 12 V.

Ce mode d'alimentation présente un inconvénient : il ne faut pas brancher l'arduino par l'USB sans avoir branché l'alimentation du moteur, sans quoi celui-ci prendrait son courant dans le port USB de l'ordinateur.

4. Excitation des phases

Pour un moteur unipolaire, les modes d'excitation possibles sont identiques à ceux du moteur bipolaire. Voir Commande d'un moteur pas à pas bipolaire sur Arduino, pour une description complète de ces modes. Cependant, le montage considéré ici ne permet pas d'annuler le courant dans un pôle (une paire de bobines), car il y a nécessairement un courant dans une des deux bobines. Le seul mode d'excitation possible est donc le mode standard (mode Two Phase ON, Full Step), dont voici la séquence :

- 1010

- ▷ 0110
- ▷ 0101
- ▷ 1001

Pour les sorties D2 et D3 de l'arduino, qui pilotent chacune une paire de bobines, la séquence est donc :

- ▷ 11
- ▷ 01
- ▷ 00
- ▷ 10

5. Programmation sur Arduino

5.a. Programmation par boucle simple

Le programme arduino ci-dessous montre comment utiliser le circuit décrit plus haut.

On commence par définir les sorties utilisées et la variable globale `etape`, qui mémorisera l'étape de la séquence d'impulsion en cours.

paspas-unipolaire.ino

```
// commande d'un moteur pas à pas unipolaire avec un ULN2803

#define PHASE_A 2
#define PHASE_B 3

int etape;
```

La fonction suivante applique les niveaux logiques pour une des étapes de la séquence :

```
void pas(int etape) {
  switch(etape) {
    case 0: // 1010
    digitalWrite(PHASE_A,HIGH);
    digitalWrite(PHASE_B,HIGH);
    break;
    case 1: // 0110
    digitalWrite(PHASE_A,LOW);
    digitalWrite(PHASE_B,HIGH);
    break;
    case 2: // 0101
    digitalWrite(PHASE_A,LOW);
    digitalWrite(PHASE_B,LOW);
    break;
    case 3: // 1001
    digitalWrite(PHASE_A,HIGH);
    digitalWrite(PHASE_B,LOW);
    break;
```

```
  }
}
```

La fonction suivante incrémente ou décrémente la variable `etape` selon le sens de mouvement choisi et appliquent les niveaux pour l'étape atteinte.

```
void pas_sens_1() {
  etape++;
  if (etape>3) etape = 0;
  pas(etape);
}

void pas_sens_2() {
  etape--;
  if (etape<0) etape = 3;
  pas(etape);
}
```

Voici la fonction d'initialisation :

```
void setup() {
  pinMode(PHASE_A,OUTPUT);
  pinMode(PHASE_B,OUTPUT);
  etape = 0;
  pas(etape);

}
```

Voici un exemple d'utilisation, avec un tour complet (48 pas pour ce moteur) suivi d'un délai de deux secondes. La durée d'un pas est de 20 ms, ce qui fait une vitesse d'environ 1 tour par seconde.

```
void loop() {
  int k;
  unsigned long duree_pas = 20;
  unsigned long temps;

  for (k=0; k<48; k++) {
    temps = millis();
    pas_sens_1();
    while (millis()-temps < duree_pas) {
          // autre chose à faire pendant le pas moteur
    }
  }
  delay(2000);

}
```

Dans cet exemple, il est prévu de faire d'autres opérations pendant un pas moteur, par exemple la lecture d'un capteur. Cela est particulièrement utile lorsque le pas moteur est long. La bibliothèque Stepper (installée par défaut avec l'environnement Arduino) ne permet pas de faire cela.

Le temps d'un pas pour ce moteur peut être réduit à 10 *ms*. Le moteur tourne alors en régime de *survitesse*, de manière continu.

5.b. Programmation par interruption

Pour les durées de pas de l'ordre de 10 millisecondes, et pour des applications nécessitant une grande précision de vitesse, la méthode précédente n'est peut-être pas assez précise. Il est alors préférable de programmer les pas par interruption. Les interruptions sont déclenchées de manière périodique par un Timer du microcontroleur. Le programme suivant utilise cette méthode, en utilisant la bibliothèque TimerOne.

paspas-unipolaire-timer.ino

```
// commande d'un moteur pas à pas unipolaire avec un ULN2803
// programmation des pas par interruption

#include <TimerOne.h>

#define PHASE_A 2
#define PHASE_B 3

int etape;

void pas(int etape) {
  switch(etape) {
    case 0: // 1010
    digitalWrite(PHASE_A,HIGH);
    digitalWrite(PHASE_B,HIGH);
    break;
    case 1: // 0110
    digitalWrite(PHASE_A,LOW);
    digitalWrite(PHASE_B,HIGH);
    break;
    case 2: // 0101
    digitalWrite(PHASE_A,LOW);
    digitalWrite(PHASE_B,LOW);
    break;
    case 3: // 1001
    digitalWrite(PHASE_A,HIGH);
    digitalWrite(PHASE_B,LOW);
    break;
  }
}

void pas_sens_1() {
  etape++;
```

```
  if (etape>3) etape = 0;
  pas(etape);
}

void pas_sens_2() {
  etape--;
  if (etape<0) etape = 3;
  pas(etape);
}


void setup() {
  pinMode(PHASE_A,OUTPUT);
  pinMode(PHASE_B,OUTPUT);
  etape = 0;
  pas(etape);
  // programmation d'une interruption toutes les 20 millisecondes
  Timer1.initialize(20000);
  Timer1.attachInterrupt(timerIsr);
}

void loop() {

}

void timerIsr() { // fonction appelée lors de l'interruption
    pas_sens_1();
}
}
```

Lorsque le moteur fonctionne en survitesse, il faut prévoir des phases d'accélération et de déccélération avec une variation progressive du pas de temps, pour ne pas perdre le synchronisme entre les impulsions et la rotation. Pour programmer une accélération, on commence par envoyer une impulsion motrice seulement une interruption sur dix, puis une sur 9, puis une sur 8 jusqu'à une impulsion par interruption. Voici un exemple :

paspas-unipolaire-timer-accel.ino

```
// commande d'un moteur pas à pas unipolaire avec un ULN2803
// programmation des pas par interruption, exemple avec accélération

#include <TimerOne.h>

#define PHASE_A 2
#define PHASE_B 3

int etape;
int compteur_1, compteur_2;
int nombre_interupt;
```

```
void pas(int etape) {
  switch(etape) {
    case 0: // 1010
    digitalWrite(PHASE_A,HIGH);
    digitalWrite(PHASE_B,HIGH);
    break;
    case 1: // 0110
    digitalWrite(PHASE_A,LOW);
    digitalWrite(PHASE_B,HIGH);
    break;
    case 2: // 0101
    digitalWrite(PHASE_A,LOW);
    digitalWrite(PHASE_B,LOW);
    break;
    case 3: // 1001
    digitalWrite(PHASE_A,HIGH);
    digitalWrite(PHASE_B,LOW);
    break;
  }
}

void pas_sens_1() {
  etape++;
  if (etape>3) etape = 0;
  pas(etape);
}

void pas_sens_2() {
  etape--;
  if (etape<0) etape = 3;
  pas(etape);
}


void setup() {
  pinMode(PHASE_A,OUTPUT);
  pinMode(PHASE_B,OUTPUT);
  etape = 0;
  compteur_1 = 0;
  compteur_2 = 0;
  nombre_interupt = 10;
  pas(etape);
  // programmation d'une interruption toutes les 10 millisecondes
  Timer1.initialize(10000);
  Timer1.attachInterrupt(timerIsr);
}
```

```
void loop() {

}

void timerIsr() { // fonction appelée lors de l'interruption
    if (nombre_interupt==0) pas_sens_1();
    else {
      compteur_1++;
      if (compteur_1==nombre_interupt) {
        pas_sens_1();
        compteur_2++;
        if (compteur_2==10) {nombre_interupt--; compteur_2=0;}
        compteur_1 = 0;
      }
    }
}
```

Références

[1] T. Wildi, *Electrotechnique*, (DeBoeck Université, 2000)
[2] C. Tavernier, *Arduino, maîtrisez sa programmation et ses cartes d'interface*, (Dunod, 2011)