Commande d'un moteur pas à pas bipolaire sur Arduino

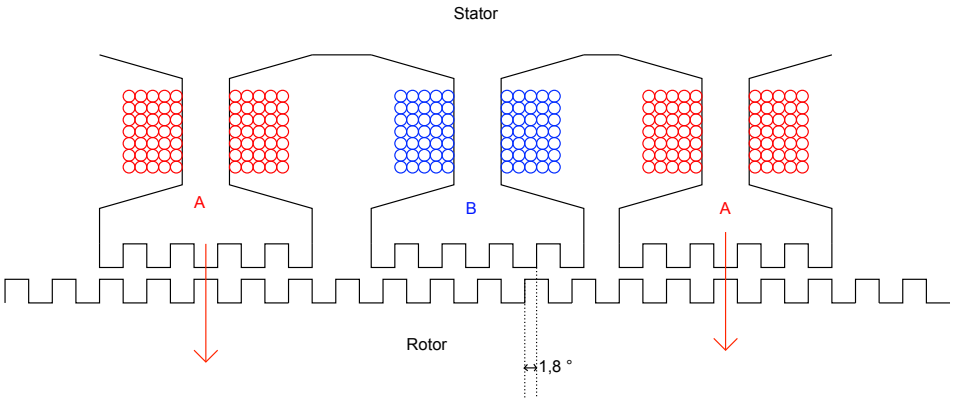
1. Introduction

Ce document montre comment commander un petit moteur pas à pas avec un Arduino. On considère le cas du moteur pas à pas bipolaire. De nombreux modèles au format NEMA17 sont disponibles sur le marché, avec des pas de 1.9 ou 0.8 degrés (respectivement 200 et 400 pas par tour).

2. Principe du moteur pas à pas et de la commande

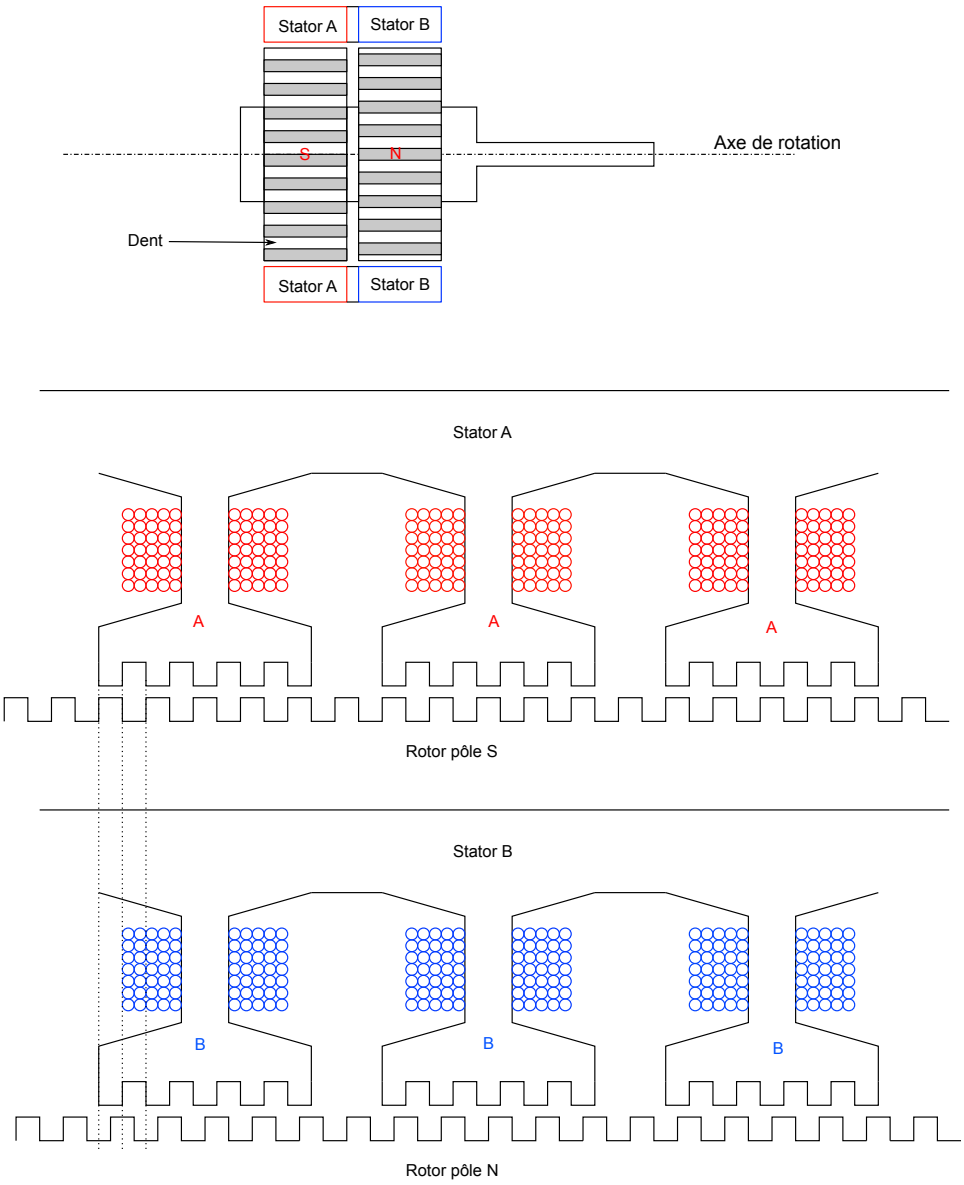
2.a. Moteur à réluctance variable et moteur hybride

Dans un moteur pas à pas à réluctance variable ([1]), les bobines d'excitation sont sur le stator. Le rotor est en acier de forte perméabilité, et n'a pas d'aimantation permanente. Le stator et le rotor sont munis d'encoches. La figure suivante montre un détail avec quelques encoches. La représentation est rectiligne (au lieu de circulaire).



Dans la position représentée, les dents des pôles A du stator sont en vis-à-vis des dents du rotor, alors que celle des pôles B sont décalées. Les pôles A sont excités par un bobinage formant la phase A (en rouge sur la figure). Dans la configuration représentée, un courant circule dans la phase A mais pas dans la phase B. Le champ magnétique généré dans les pôles A induit une aimantation dans le rotor en acier. La configuration stable est celle où les dents du rotor sont alignées avec celles du stator. Si l'on tente d'écarter le rotor de cette position, un couple de rappel s'exerce. Lorsqu'on coupe le courant dans la phase A et qu'on alimente le bobinage de la phase B, le rotor tourne pour obtenir l'alignement avec les dents des pôles B. Si le rotor comporte 100 dents, cela fait une rotation de $1,8$ degrés (moteur 200 pas par tour).

Les moteurs pas à pas sont souvent de type hybride ([2]), avec une aimantation permanente du rotor. Le rotor d'un moteur hybride est constitué de deux roues dentées décalées. La première roue est un pôle sud magnétique et se trouve en vis à vis des pôles A du stator. La seconde roue est un pôle nord magnétique et se trouve en vis à vis des pôles B du stator. La figure suivante représente un moteur hybride dans la configuration où la phase A est alimentée. Les dents du pôle sud du rotor sont alors alignées avec celle des pôles A du stator.



Dans un moteur hybride, il y a un faible couple même en l'absence de courant, qui bloque le rotor dans une position d'alignement, soit avec les pôles A soit avec les pôles B. Ce couple permet souvent de bloquer le moteur au repos sans lui fournir du courant (à condition que la charge au repos soit faible).

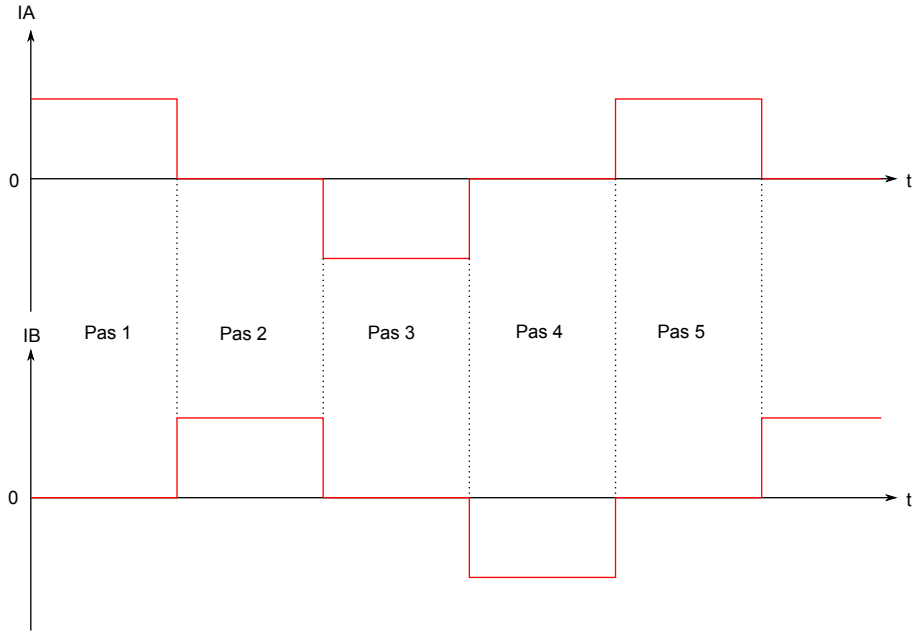
Un moteur pas à pas bipolaire comporte 4 fils d'alimentation, deux pour la phase A, deux pour la phase B. Un ohmmètre permet facilement de repérer les deux paires.

Pour les moteurs unipolaires, qui comportent deux bobines en sens inverse par pôle (deux pour le pôle A, deux pour le pôle B), voir Commande d'un moteur pas à pas unipolaire sur Arduino.

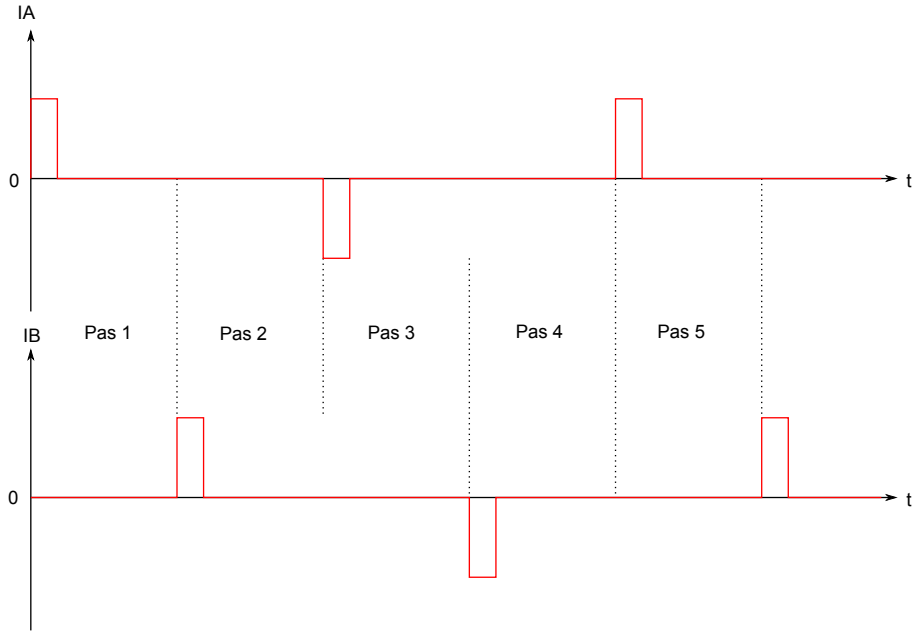
2.b. Excitation des phases

Dans le premier mode d'excitation, la rotation du rotor se fait lorsqu'on coupe le courant dans une phase tout en déclenchant l'alimentation de l'autre phase. Dans ce

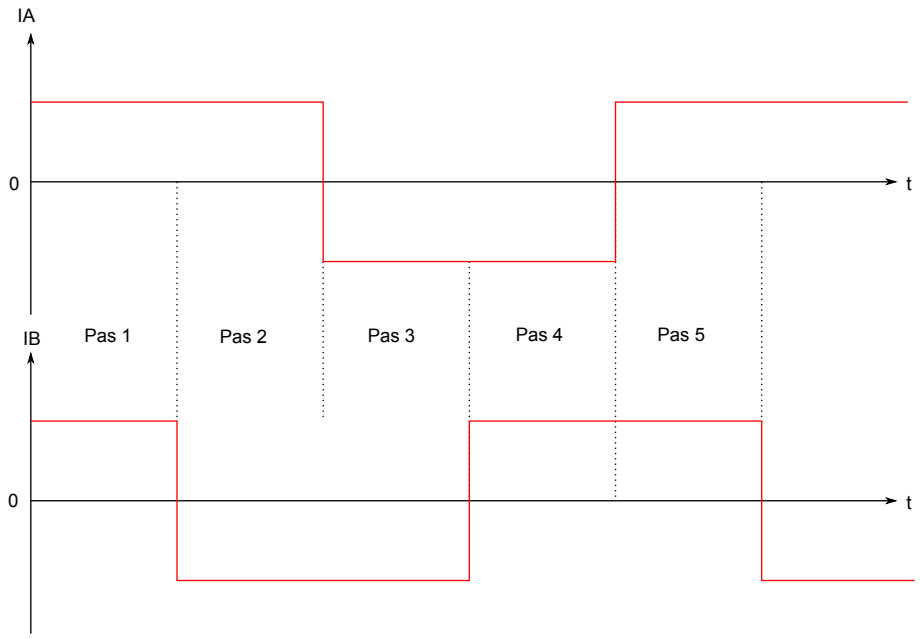
mode, le courant ne circule que dans une phase à la fois (mode One Phase ON, Full Step). Le sens du courant est inversé à chaque alternance. Ce mode est aussi appelé *excitation ondulée*.



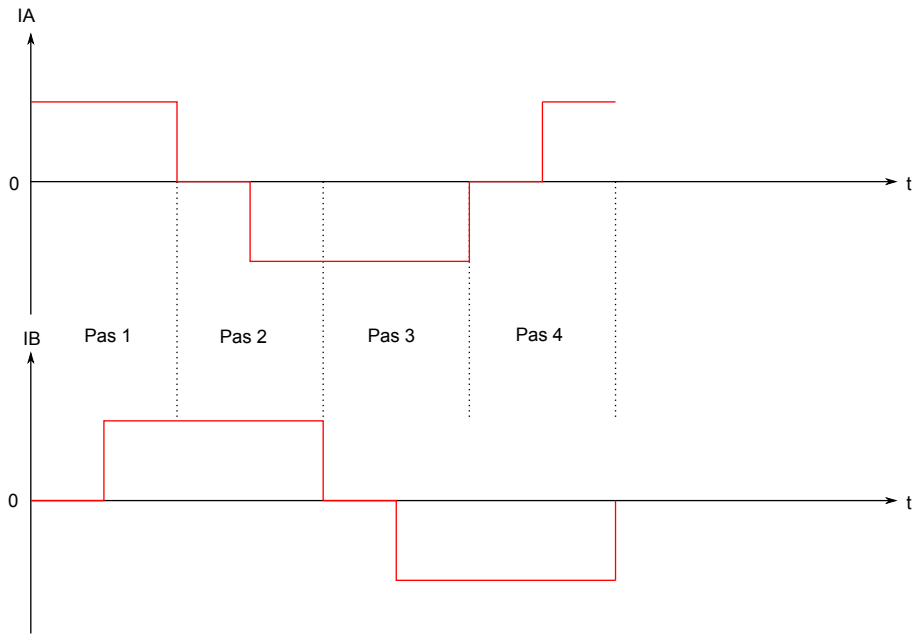
La rotation se fait très rapidement, avec un temps de l'ordre de la milliseconde, juste après le changement des courants (il faudra tout de même tenir compte du temps d'établissement du courant). La durée des pas est généralement de plusieurs dizaines de millisecondes. Le courant est maintenu de manière à bloquer le rotor dans sa position. Si la charge est faible, on peut aussi couper le courant entre les pas, après avoir appliqué une impulsion assez longue pour provoquer la rotation. Par exemple, si les pas sont espacés d'une seconde, on peut appliquer des impulsions de 50 millisecondes suivies de plages de courant nul.



Le second mode (Two Phase ON, Full Step) consiste à alimenter les deux phases en même temps, et à changer le sens du courant à chaque pas. Avec ce mode, le couple moteur est plus important, mais la dissipation est évidemment plus grande (dans le moteur est dans le circuit de commande). Ce mode est aussi appelé *excitation standard*.



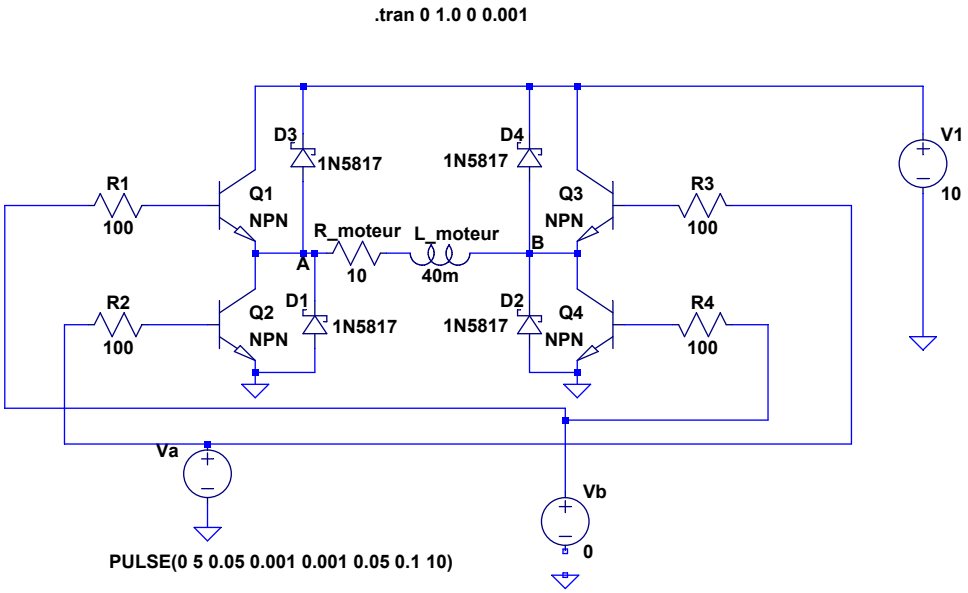
Le troisième mode permet de faire des déplacements par demi-pas (mode Half Step) :



3. Pont en H à transistor bipolaire

3.a. Pont en H

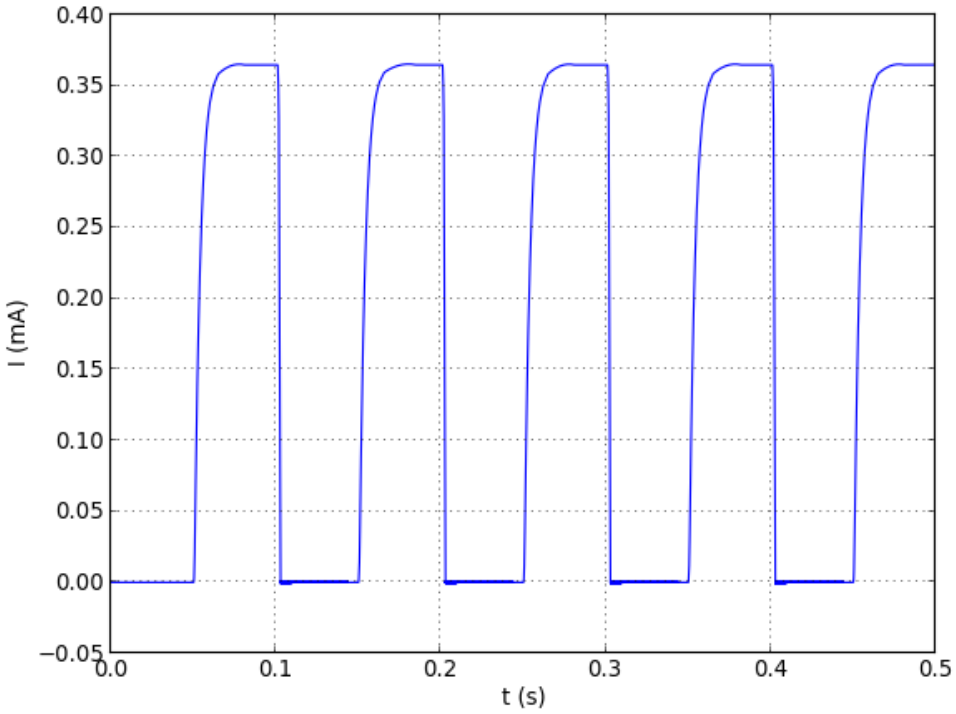
Un pont en H permet d'alimenter un bobinage du moteur (une phase) en permettant de changer le sens du courant. Voici le schéma de principe d'un pont en H :



Le bobinage est représenté par une résistance et une inductance. Lorsque $Va = 5\ V$ et $Vb = 0$, les transitors Q2 et Q3 sont passants et un courant positif circule dans le moteur de B vers A. Inversement, lorsque $Va = 0$ et $Vb = 5\ V$, un courant positif circule de A vers B. Lorsque les deux tensions de commande sont à zéro, le courant dans le bobinage est nul. Les diodes permettent au courant de ce réduire continûment lorsque les transistors sont bloqués (diodes de roue libre)

Le temps de commutation des transistors est négligeable devant le temps d'établissement du courant dans la bobine, qui est de l'ordre de $L/R = 4\ ms$. La simulation LTSPICE effectuée consiste à faire varier Va sous forme d'impulsions carrées, tout en maintenant $Vb = 0$.

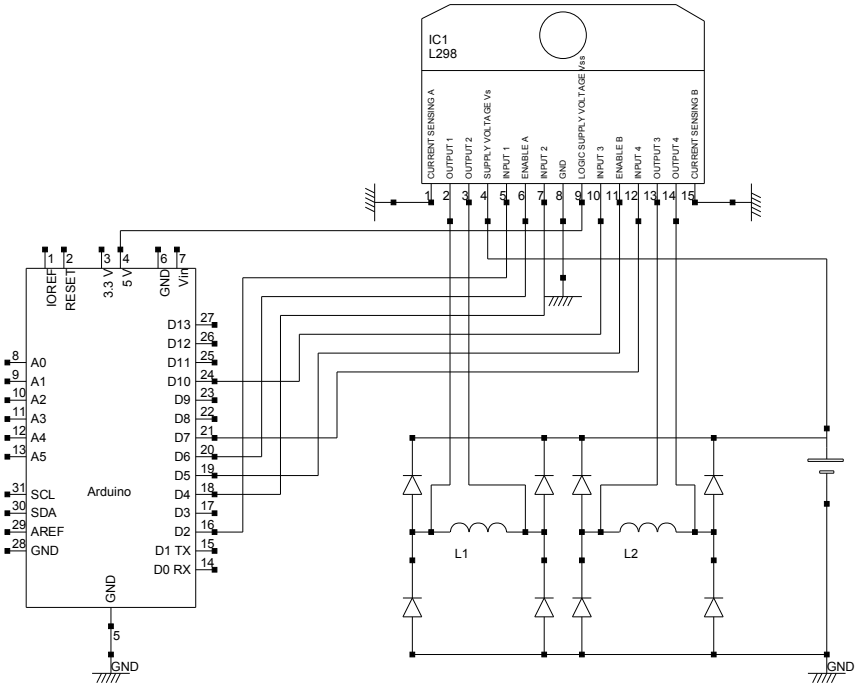
Voici le courant dans la bobine obtenu par cette simulation :



La durée des pas est de 100 ms (en réalité, il y a une inversion du sens du courant qui n'est pas introduite dans la simulation). On voit bien le régime transitoire d'établissement du courant. Pour obtenir le couple maximal, il faut que la durée des impulsions soit supérieure au temps d'établissement du courant.

3.b. Circuit avec arduino

Le circuit L298N comporte deux ponts en H à transistors bipolaires, pouvant en principe délivrer chacun jusqu'à 2 A, sous réserve que la chaleur dissipée soit bien évacuée. Voici un circuit complet avec les connexions à un arduino. Ces connexions permettent d'utiliser conjointement la platine *Arduino Motor Shield*.



Le pont A sert à alimenter la phase A, le pont B à alimenter la phase B. Les deux sorties du pont A sont OUTPUT1 et OUTPUT2. Le pont A est piloté par les trois entrées logiques INPUT1, INPUT2 et ENABLEA. Lorsque ENABLEA=1, un sens du courant est obtenu avec INPUT1=1 et INPUT2=0, alors que le sens opposé est obtenu avec INPUT1=0 et INPUT2=1. Lorsque ENABLEA=0, le pont est inactif (le courant est nul).

Voici la table de correspondance entre les sorties de l'arduino utilisées et les entrées du L298 :

- D2 : INPUT 1
- D4 : INPUT 2
- D6 : ENABLEA
- D10 : INPUT 3
- D7 : INPUT 4
- D5 : ENABLE B

Les entrées ENABLEA et ENABLEB sont pilotées par des sorties PWM de l'Arduino, ce qui permet éventuellement d'appliquer un signal PWM pour limiter le courant moyen, comme on le fait pour les moteurs à courant continu.

3.c. Signaux de commande

On convient de noter les niveaux logiques dans l'ordre INPUT1, INPUT2, INPUT3, INPUT4. Voici les niveaux à appliquer pour effectuer un pas moteur dans un sens (on change l'ordre pour le sens opposé).

Mode à pas entier, une phase alimentée à la fois (One Phase ON, Full Step) :

- 1000

- ▷ 0010
- ▷ 0100
- ▷ 0001

Mode à pas entier, deux phases alimentées en même temps (Two Phase ON, Full Step) :

- ▷ 1010
- ▷ 0110
- ▷ 0101
- ▷ 1001

Mode demi-pas :

- ▷ 1000
- ▷ 1010
- ▷ 0010
- ▷ 0110
- ▷ 0100
- ▷ 0101
- ▷ 0001
- ▷ 1001

À noter qu'il existe des circuits logiques (comme le L297), qui permettent de générer ces séquences en synchronisation avec un signal d'horloge. Dans le cas présent, on utilisera l'arduino pour générer les séquences, ce qui représente une charge de travail minime pour celui-ci. Le véritable intérêt d'un circuit de commande comme le L297 est de permettre la régulation de courant par hachage, que l'on verra plus loin.

3.d. Programmation sur arduino

Le programme suivant montre comment piloter un moteur pas à pas avec le circuit ci-dessus.

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

L298-paspas.ino

```
// commande d'un moteur pas à pas bipolaire avec un L298

#define INPUT1 2
#define INPUT2 4
#define ENABLEA 6
#define INPUT3 10
#define INPUT4 7
#define ENABLEB 5

int etape;
```

Les trois fonctions suivantes appliquent les signaux logiques pour l'une des étapes, pour les trois modes décrits plus haut :

```
void pas_une_phase(int etape) {
  switch(etape) {
    case 0 : // 1000
    digitalWrite(INPUT1,HIGH);
    digitalWrite(INPUT2,LOW);
    digitalWrite(INPUT3,LOW);
    digitalWrite(INPUT4,LOW);
    break;
    case 1: // 0010
    digitalWrite(INPUT1,LOW);
    digitalWrite(INPUT2,LOW);
    digitalWrite(INPUT3,HIGH);
    digitalWrite(INPUT4,LOW);
    break;
    case 2: // 0100
    digitalWrite(INPUT1,LOW);
    digitalWrite(INPUT2,HIGH);
    digitalWrite(INPUT3,LOW);
    digitalWrite(INPUT4,LOW);
    break;
    case 3: // 0001
    digitalWrite(INPUT1,LOW);
    digitalWrite(INPUT2,LOW);
    digitalWrite(INPUT3,LOW);
    digitalWrite(INPUT4,HIGH);
    break;
  }
}

void pas_deux_phases(int etape) {
  switch(etape) {
    case 0 : // 1010
    digitalWrite(INPUT1,HIGH);
    digitalWrite(INPUT2,LOW);
    digitalWrite(INPUT3,HIGH);
    digitalWrite(INPUT4,LOW);
    break;
    case 1: // 0110
    digitalWrite(INPUT1,LOW);
    digitalWrite(INPUT2,HIGH);
    digitalWrite(INPUT3,HIGH);
    digitalWrite(INPUT4,LOW);
    break;
    case 2: // 0101
    digitalWrite(INPUT1,LOW);
    digitalWrite(INPUT2,HIGH);
    digitalWrite(INPUT3,LOW);
    digitalWrite(INPUT4,HIGH);
```

```
    break;
    case 3: // 1001
    digitalWrite(INPUT1,HIGH);
    digitalWrite(INPUT2,LOW);
    digitalWrite(INPUT3,LOW);
    digitalWrite(INPUT4,HIGH);
    break;
  }
}

void pas_demi_pas(int etape) {
  switch(etape) {
    case 0 : // 1000
    digitalWrite(INPUT1,HIGH);
    digitalWrite(INPUT2,LOW);
    digitalWrite(INPUT3,LOW);
    digitalWrite(INPUT4,LOW);
    break;
    case 1: // 1010
    digitalWrite(INPUT1,HIGH);
    digitalWrite(INPUT2,LOW);
    digitalWrite(INPUT3,HIGH);
    digitalWrite(INPUT4,LOW);
    break;
    case 2: // 0010
    digitalWrite(INPUT1,LOW);
    digitalWrite(INPUT2,LOW);
    digitalWrite(INPUT3,HIGH);
    digitalWrite(INPUT4,LOW);
    break;
    case 3: // 0110
    digitalWrite(INPUT1,LOW);
    digitalWrite(INPUT2,HIGH);
    digitalWrite(INPUT3,HIGH);
    digitalWrite(INPUT4,LOW);
    break;
    case 4: //0100
    digitalWrite(INPUT1,LOW);
    digitalWrite(INPUT2,HIGH);
    digitalWrite(INPUT3,LOW);
    digitalWrite(INPUT4,LOW);
    break;
    case 5: //0101
    digitalWrite(INPUT1,LOW);
    digitalWrite(INPUT2,HIGH);
    digitalWrite(INPUT3,LOW);
    digitalWrite(INPUT4,HIGH);
    break;
```

```
    case 6: //0001
    digitalWrite(INPUT1,LOW);
    digitalWrite(INPUT2,LOW);
    digitalWrite(INPUT3,LOW);
    digitalWrite(INPUT4,HIGH);
    break;
    case 7: //1001
    digitalWrite(INPUT1,HIGH);
    digitalWrite(INPUT2,LOW);
    digitalWrite(INPUT3,LOW);
    digitalWrite(INPUT4,HIGH);
    break;
  }
}
```

Les fonctions suivantes incrémentent ou décrémentent la variable `etape` selon le sens de mouvement choisi et appliquent les signaux pour l'étape atteinte.

```
void pas_une_phase_sens_1() {
  etape++;
  if (etape>3) etape = 0;
  pas_une_phase(etape);
}

void pas_une_phase_sens_2() {
  etape--;
  if (etape<0) etape = 3;
  pas_une_phase(etape);
}

void pas_deux_phases_sens_1() {
  etape++;
  if (etape>3) etape = 0;
  pas_deux_phases(etape);
}

void pas_deux_phases_sens_2() {
  etape--;
  if (etape<0) etape = 3;
  pas_deux_phases(etape);
}

void pas_demi_pas_sens_1() {
  etape++;
  if (etape>7) etape = 0;
  pas_demi_pas(etape);
}
```

```
void pas_demi_pas_sens_2() {
  etape--;
  if (etape<0) etape = 7;
  pas_demi_pas(etape);
}
```

La fonction suivante applique un signal PWM sur les entrées ENABLEA et ENABLEB du L298. Dans la plupart des cas, on utilisera soit la valeur 255 soit la valeur 0.

```
void pwm(int pwm) {
  analogWrite(ENABLEA,pwm);
  analogWrite(ENABLEB,pwm);
}
```

Voici la fonction d'initialisation :

```
void setup() {
    pinMode(INPUT1,OUTPUT);
    pinMode(INPUT2,OUTPUT);
    pinMode(INPUT3,OUTPUT);
    pinMode(INPUT4,OUTPUT);
    pinMode(ENABLEA,OUTPUT);
    pinMode(ENABLEB,OUTPUT);
    analogWrite(ENABLEA,0);
    analogWrite(ENABLEB,0);
    etape = 0;
}
```

Voici un exemple avec des blocs de 100 pas suivi d'un temps d'attente de 3 secondes. La durée d'application du courant est choisie, et la durée totale du pas (en millisecondes). Pour faire tourner le moteur à une vitesse bien précise, on calculera la durée du pas en tenant compte du nombre de pas par tour du moteur. Dans cet exemple, le courant de repos est nul, ce qui suppose que la charge au repos du moteur soit faible. Si une charge importante doit être appliquée, il faut choisir un courant de repos maximal (255). Cela conduit à une dissipation importante dans le L298 (qui doit être muni d'un radiateur). On peut aussi choisir d'appliquer un PWM (par exemple une valeur de 100), ce qui réduit le courant moyen mais a pour conséquence l'émission d'un son à la fréquence du PWM (environ 1000 Hz).

```
void loop() {
  int k;
  unsigned int duree_courant = 10;
  unsigned long duree_pas = 50;
  unsigned long temps;
```

```
  for (k=0; k<100; k++) {
    temps = millis();
    pas_une_phase_sens_1();
    pwm(255); // courant max
    while (millis()-temps < duree_courant) {
          // autre chose à faire pendant le pas moteur
    }
    pwm(0); // courant de repos, en fonction de la charge
    while (millis()-temps < duree_pas) {
          // autre chose à faire pendant le pas moteur
    }
  }
  pwm(0); // courant de repos, en fonction de la charge
  delay(3000);

}
```

Il est possible de faire tourner le moteur avec une durée de pas de l'ordre de la milliseconde. Si la durée est assez courte, le moteur tourne de manière continue tout en restant en synchronisme avec les impulsions : c'est le mode de rotation en *survitesse*. Voici comment faire un tour complet avec un moteur de 200 pas par tour, en utilisant le mode demi-pas :

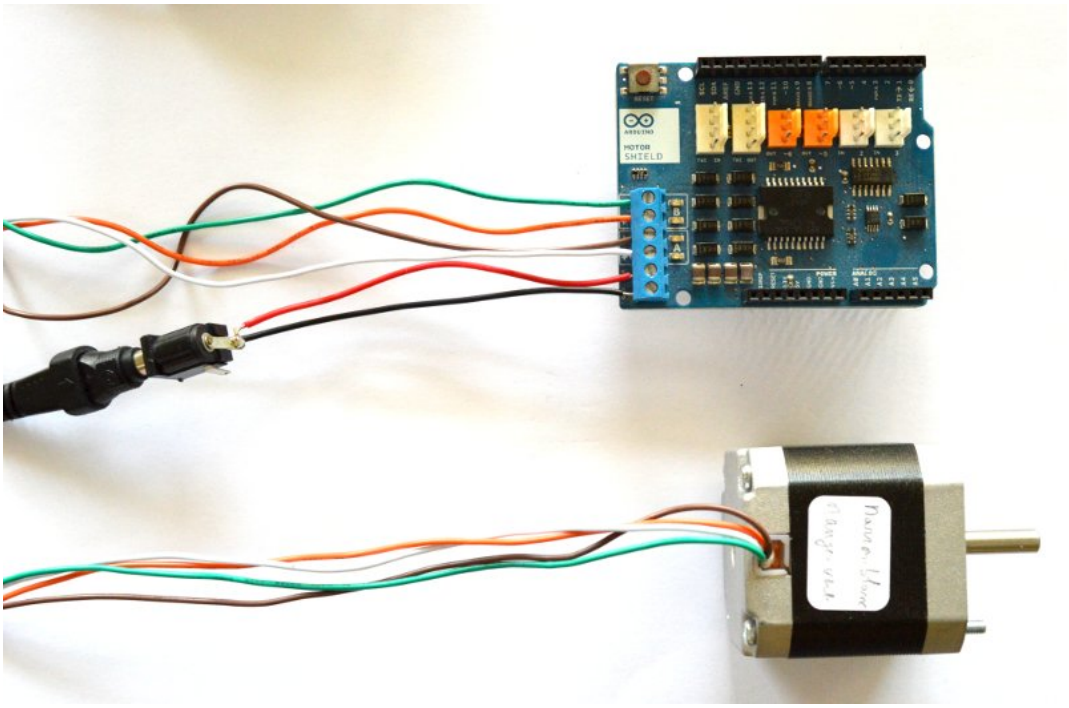
```
duree_pas = 2
pwm(255); // courant max
for (k=0; k<400; k++) {
    temps = millis();
    pas_demi_pas_sens_1();
    while (millis()-temps < duree_pas) {}
  }
```

Dans le cas d'un moteur à fort courant (plus de 1 A), on peut réduire la dissipation en réduisant la durée d'application du courant, tout en restant au dessus de 5 ms. Le reste du temps, on applique un courant nul, ou bien un courant PWM de valeur moyenne faible.

3.e. Utilisation du motor shield arduino

Le Motor-Shield officiel Arduino est plutôt prévu pour piloter un ou deux moteurs à courant continu, mais il peut être utilisé pour un moteur pas à pas de faible puissance de la manière suivante (Attention, la bibliothèque Stepper n'est pas utilisable pour cela).

Voici le branchement d'un moteur pas à pas bipolaire sur ce shield :



Pour déterminer les fils de chaque phase, il faut utiliser un ohmmètre. L'alimentation du moteur est faite directement sur le shield par un bloc secteur 12 V. La liaison Vin a été coupée de manière à éviter une alimentation accidentelle par le port USB.

Le programme ci-dessous reprend les fonctions du précédent pour le motor shield arduino :

motorshieldPasPasBipolaire.ino

```
// commande moteur pas à pas bipolaire avec Arduino-MotorShield
#define DIRA 12
#define DIRB 13
#define BRAKEA 9
#define BRAKEB 8
#define PWMA 3
#define PWMB 11
#define VAL 100

int etape;

void pas_une_phase(int etape) { // pas One Phase ON
  switch (etape) {
    case 0 : // 1000
    digitalWrite(BRAKEA,LOW);
    digitalWrite(BRAKEB,HIGH);
    digitalWrite(DIRA,HIGH);
    break;
    case 1 : // 0010
    digitalWrite(BRAKEA,HIGH);
    digitalWrite(BRAKEB,LOW);
    digitalWrite(DIRB,HIGH);
```

```
    break;
    case 2 : // 0100
    digitalWrite(BRAKEA,LOW);
    digitalWrite(BRAKEB,HIGH);
    digitalWrite(DIRA,LOW);
    break;
    case 3 : // 0001
    digitalWrite(BRAKEA,HIGH);
    digitalWrite(BRAKEB,LOW);
    digitalWrite(DIRB,LOW);
    break;
  }
}

void pas_deux_phases(int etape) { // pas One Phase ON
  switch (etape) {
    case 0 : // 1010
    digitalWrite(BRAKEA,LOW);
    digitalWrite(BRAKEB,LOW);
    digitalWrite(DIRA,HIGH);
    digitalWrite(DIRB,HIGH);
    break;
    case 1 : // 0110
    digitalWrite(BRAKEA,LOW);
    digitalWrite(BRAKEB,LOW);
    digitalWrite(DIRA,LOW);
    digitalWrite(DIRB,HIGH);
    break;
    case 2 : // 0101
    digitalWrite(BRAKEA,LOW);
    digitalWrite(BRAKEB,LOW);
    digitalWrite(DIRA,LOW);
    digitalWrite(DIRB,LOW);
    break;
    case 3 : // 1001
    digitalWrite(BRAKEA,LOW);
    digitalWrite(BRAKEB,LOW);
    digitalWrite(DIRA,HIGH);
    digitalWrite(DIRB,LOW);
    break;
  }
}

void pas_demi_pas(int etape) {
  switch(etape) {
    case 0 : // 1000
    digitalWrite(BRAKEA,LOW);
    digitalWrite(BRAKEB,HIGH);
```

```
    digitalWrite(DIRA,HIGH);
    break;
    case 1: // 1010
    digitalWrite(BRAKEA,LOW);
    digitalWrite(BRAKEB,LOW);
    digitalWrite(DIRA,HIGH);
    digitalWrite(DIRB,HIGH);
    break;
    case 2: // 0010
    digitalWrite(BRAKEA,HIGH);
    digitalWrite(BRAKEB,LOW);
    digitalWrite(DIRB,HIGH);
    break;
    case 3: // 0110
    digitalWrite(BRAKEA,LOW);
    digitalWrite(BRAKEB,LOW);
    digitalWrite(DIRA,LOW);
    digitalWrite(DIRB,HIGH);
    break;
    case 4: //0100
    digitalWrite(BRAKEA,LOW);
    digitalWrite(BRAKEB,HIGH);
    digitalWrite(DIRA,LOW);
    break;
    case 5: //0101
    digitalWrite(BRAKEA,LOW);
    digitalWrite(BRAKEB,LOW);
    digitalWrite(DIRA,LOW);
    digitalWrite(DIRB,LOW);
    break;
    case 6: //0001
    digitalWrite(BRAKEA,HIGH);
    digitalWrite(BRAKEB,LOW);
    digitalWrite(DIRB,LOW);
    break;
    case 7: //1001
    digitalWrite(BRAKEA,LOW);
    digitalWrite(BRAKEB,LOW);
    digitalWrite(DIRA,HIGH);
    digitalWrite(DIRB,LOW);
    break;
  }
}

void pas_une_phase_sens_1() {
  etape++;
  if (etape>3) etape = 0;
  pas_une_phase(etape);
```

```
}

void pas_une_phase_sens_2() {
  etape--;
  if (etape<0) etape = 3;
  pas_une_phase(etape);
}

void pas_deux_phases_sens_1() {
  etape++;
  if (etape>3) etape = 0;
  pas_deux_phases(etape);
}

void pas_deux_phases_sens_2() {
  etape--;
  if (etape<0) etape = 3;
  pas_deux_phases(etape);
}

void pas_demi_pas_sens_1() {
  etape++;
  if (etape>7) etape = 0;
  pas_demi_pas(etape);
}

void pas_demi_pas_sens_2() {
  etape--;
  if (etape<0) etape = 7;
  pas_demi_pas(etape);
}

void pwm(int pwm) {
  analogWrite(PWMA,pwm);
  analogWrite(PWMB,pwm);
}

void setup() {
  pinMode(DIRA,OUTPUT);
  pinMode(DIRB,OUTPUT);
  pinMode(BRAKEA,OUTPUT);
  pinMode(BRAKEB,OUTPUT);
  pinMode(PWMA,OUTPUT);
  pinMode(PWMB,OUTPUT);
  analogWrite(PWMA,0);
  analogWrite(PWMB,0);
  etape = 0;
}
```

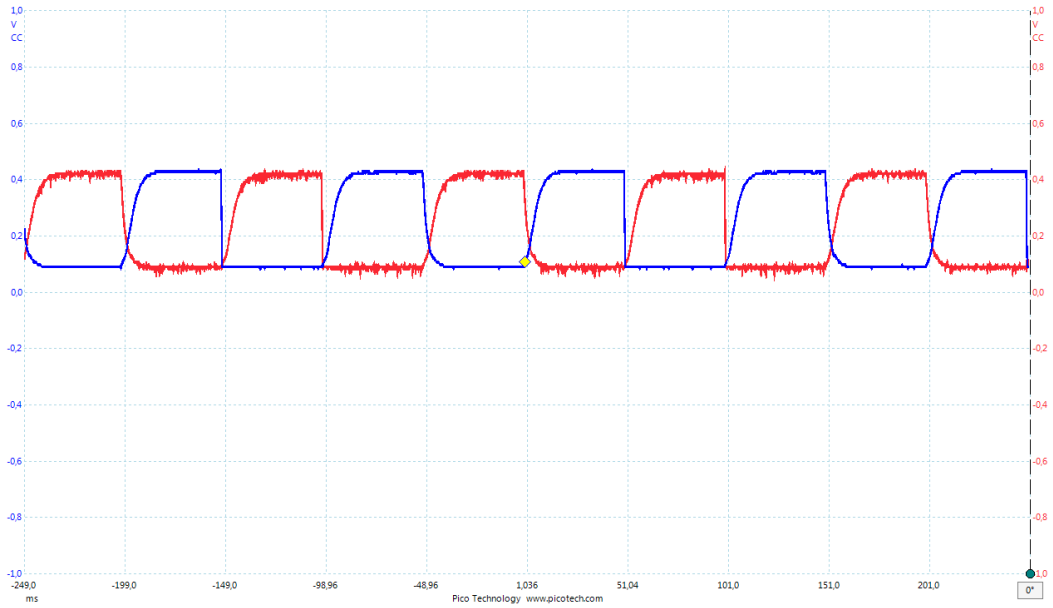
```
void loop() {
  int k;
  unsigned int duree_courant = 10;
  unsigned long duree_pas = 20;
  unsigned long temps;

  for (k=0; k<100; k++) {
    temps = millis();
    pas_deux_phases_sens_1();
    pwm(255); // courant max
    while (millis()-temps < duree_courant) {
          // autre chose à faire pendant le pas moteur
    }
    pwm(0); // courant de repos, en fonction de la charge
    while (millis()-temps < duree_pas) {
          // autre chose à faire pendant le pas moteur
    }
  }
  pwm(0); // courant de repos, en fonction de la charge
  delay(3000);

}
```

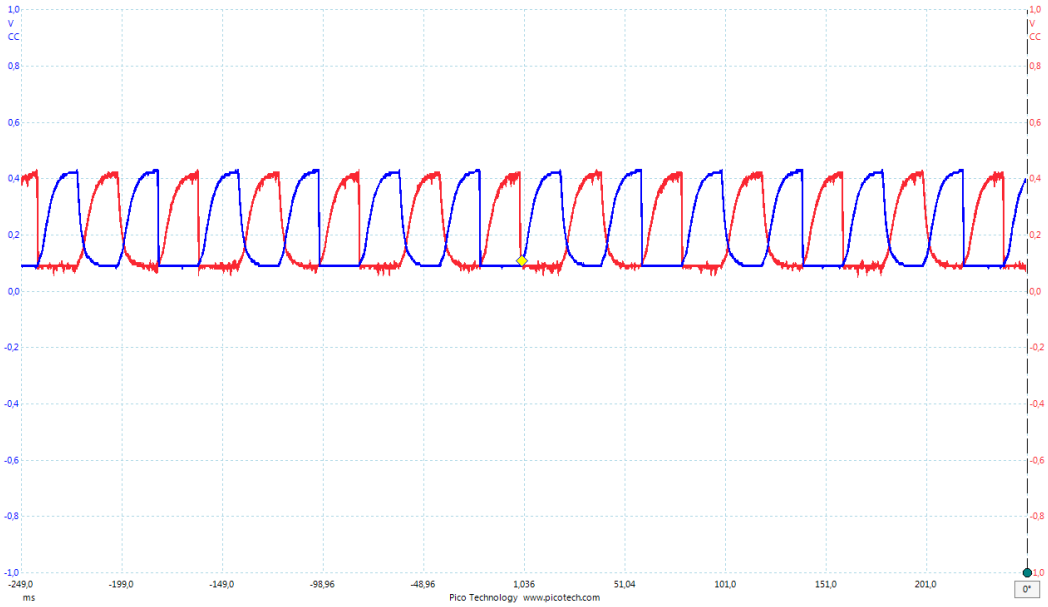
Le L298 du motor shield arduino ne possède pas de radiateur, c'est pourquoi son utilisation pour les moteurs dont le courant est supérieur à 1 A est déconseillée (il y a de toute manière une coupure thermique dans le L298).

Sur ce shield, les bornes CURRENT SENSING A et B du L298 sont branchées sur une résistance de $1,65\ \Omega$, ce qui permet d'obtenir le courant circulant dans les phases A et B (bornes A0 et A1 de l'arduino). Voici l'enregistrement à l'oscilloscope des ces tensions pour des pas de durée 50 ms, avec un moteur consommant 300 mA sous 12 V :

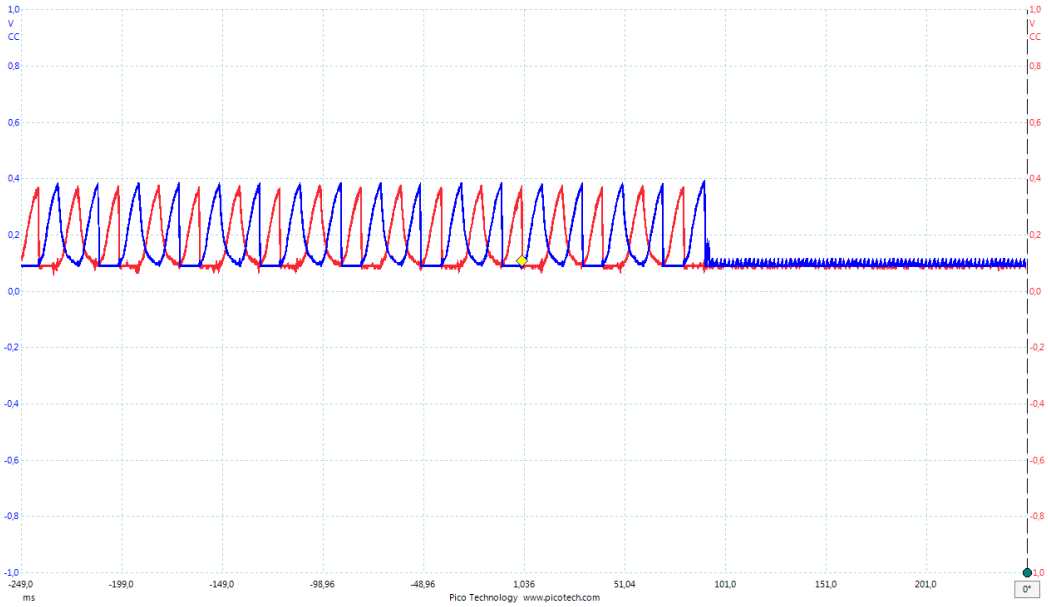


Le mode utilisé est le mode 1 (une phase alimenté à la fois). Le sens du courant n'apparaît pas. On voit que cette durée de pas est largement suffisante pour l'établissement du courant. La fiche du constructeur donne pour une phase du moteur $L = 90\ mH$ et $R = 40\Omega$, ce qui fait un temps $L/R = 2,5\ ms$, en accord avec cet oscillographe.

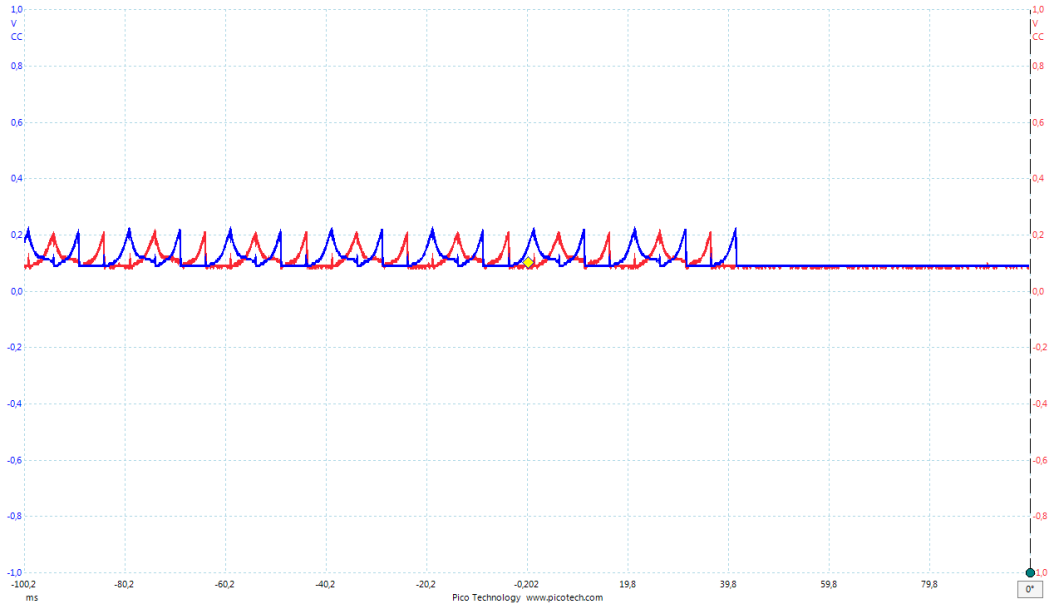
Voici le résultat pour une durée de pas de 20 ms :



Pour une durée de 10 ms :



Pour une durée de 5 ms :



Pour ces deux dernières durées, le courant maximal n'est pas atteint au cours d'un pas, mais le moteur tourne bien. Dans ce régime de fonctionnement, le moteur émet un sifflement dont la fréquence est celle des pas.

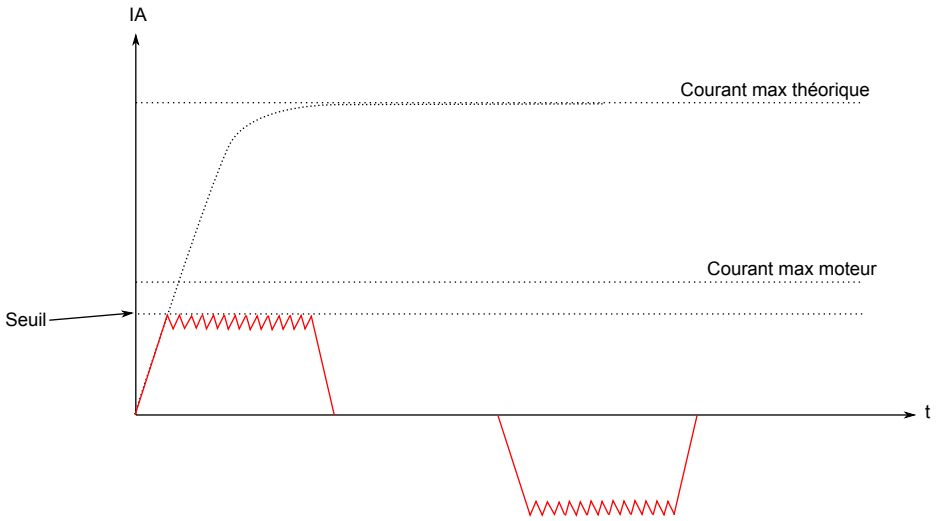
4. Régulation du courant par hacheur

4.a. Principe

Les enregistrements de courant ci-dessus montrent un inconvénient du mode de commande précédent : le temps de montée du courant dans la bobine (rapport L/R) limite l'utilisation à haute fréquence (la limitation n'est pas mécanique). D'autre part, lorsque le courant augmente dans une phase, le courant dans l'autre phase descend à zéro trop lentement, ce qui réduit le couple. Par ailleurs, ce type de commande ne permet pas de réguler le courant pendant l'impulsion.

Pour diminuer le temps de montée du courant, on peut ajouter une résistance en série avec chaque phase et augmenter la tension d'alimentation. Cependant, cette solution n'est pas efficace car une partie importante de l'énergie est dissipée dans cette résistance ajoutée.

La régulation du courant par hachage consiste à couper le courant dès qu'il dépasse un seuil (réglable). Le courant est rétabli peu de temps après la coupure. La base de temps est fournie par un oscillateur dont la fréquence est ajustée par un choix de R et C. Le pont en H est alimenté avec une tension beaucoup plus grande que la tension nominale du moteur, par exemple 36 V pour un moteur de 12 V, ce qui permet d'augmenter la pente de montée du courant. Voici à quoi ressemble la courbe de courant avec la régulation par découpage :



Les variations du courant pendant le découpage ont un période inférieure à celle de l'oscillateur. Plus l'oscillateur est rapide, plus ces variations ont une amplitude faible. En ajustant le seuil de courant, on peut ainsi réguler l'intensité atteinte pendant l'impulsion de commande. Cela permet de réduire notablement la dissipation lorsque la charge appliquée au moteur le permet. Le même dispositif peut d'ailleurs être appliqué à un moteur à courant continu pour réguler le courant.

La commande par découpage permet d'augmenter la fréquence des pas tout en gardant un couple important.

Références

[1] A. Hughes, B. Drury, *Electric motors and drives*, (Elsevier, 2013)
[2] T. Wildi, *Electrotechnique*, (DeBoeck Université, 2000)