Département de Génie Electrotechnique et Automatique
Université de 8 Mai 1945 Guelma

Licence 3ème année Automatique
Module : Bus de communication et réseaux locaux industriels

TP Supplémentaire: Commande des moteurs par Arduino

1. Introduction

Dans ce TP, nous allons contrôler différents types de moteurs dotés de différentes technologies tels que le moteur à courant continu, le servo-moteur, le moteur brushless et le moteur pas-à-pas. Généralement, chaque type de moteur nécessite un driver spécial pour le commander, à l'exception des servo-moteurs qui embarquent le driver à l'intérieur du boitier.

2. Commande d'un moteur à courant continu

Le moteur à courant continu (ou DC réfère à *Direct Current*) est un moteur qu'on trouve un peu partout, notamment dans les jouets des enfants (véhicules et autres) vu son coût très faible. Ce type de moteur est doté de deux fils seulement, et dés qu'on branche l'alimentation sur les deux fils il commence à tourner à une vitesse rapide. Toutefois, lorsque l'alimentation est inversée (le plus + de l'alimentation dans le moins – du moteur et le moins – de l'alimentation dans le plus + du moteur) le moteur tourne dans le sens inverse.

Chaque moteur a un voltage maximum à supporter (sinon il se brûle) et auquel il tourne à sa grande vitesse. Malheureusement, on ne peut pas varier la vitesse de rotation s'il est branché directement dans l'alimentation, ni de changer le sens de rotation en cours de marche. Le seul moyen de contrôler la vitesse est d'abaisser le voltage d'entrée, ce qui requiert une alimentation variable qui est défavorable.

Figure 1. Moteur à courant continu

Donc, pour contrôler le sens de rotation on utilise généralement un circuit électronique qui s'appelle '*pont-H*' qui est constitué de quatre éléments de commutation, i.e. généralement des

transistors. La solution la plus pratique consiste à utiliser un circuit intégré basé sur un *pont-H* et encastré dans un module (appelé *driver* ou *shield*) tel que L298N, L293D, MX1508, etc.

Les modules à base de *pont-H* ne permettent pas seulement le contrôle du sens, mais également la vitesse de rotation grâce au signal PWM[1]. Ce dernier est un signal pseudo-analogique, ce qui veut dire un signal analogique (0 et 1) avec une fréquence fixe. Le principe du signal PWM consiste à générer des impulsions égales ou inférieures à la période. En outre, plus la durée d'impulsion se rapproche de la période plus l'actionneur (i.e. moteur, ampoule, etc.) converge vers un fonctionnement à sa grande puissance (ou vitesse).

Figure 2. Exemple de différents drivers à base de pont-H. A gauche un driver L298N, au centre un driver L293D et à droite un driver MX1508.

En général, le cycle de service (pourcentage) de fonctionnement peut être calculé en divisant la durée d'impulsion T_0 par la période T et en multipliant le résultat par 100. Le signal PWM doit être généré à une fréquence de 20 KHz (i.e. T= 50µs) au minimum pour éviter d'être dans la gamme audible (perçue par l'être humain).

$$\text{Pourcentage} = \frac{T_0}{T} \times 100 \qquad (3.1)$$

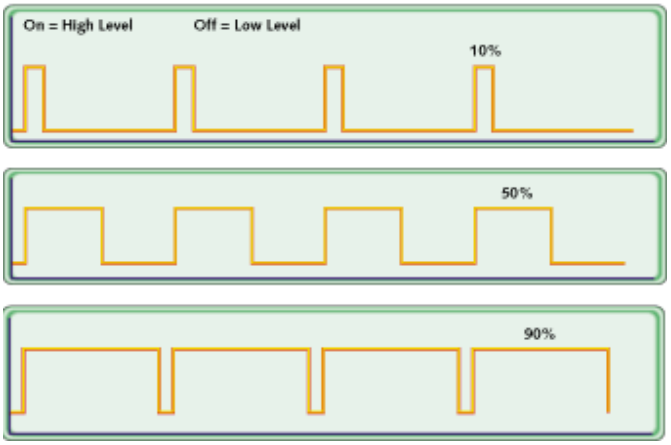

Figure 3. Exemple de différents signaux PWM [1]

[1] PWM est l'abréviation de *Pulse Modulation Width*

Par exemple, pour une fréquence de 20 KHz correspondant à une période de 50µs, si la vitesse est 90% de la vitesse maximale (90% de cycle de service), on envoie une impulsion pendant 45µs (Equation 3.2) et les 5µs restantes on envoie 0 (Figure 3.3).

$$\begin{cases} 50\mu s \rightarrow 100\% \\ x \rightarrow 90\% \end{cases} \Longrightarrow x = \frac{90\% \times 50\mu s}{100\%} = 45\mu s \qquad (3.2)$$

Alors, pour générer un signal PWM en Arduino on a deux solutions, où la première consiste à implémenter le mécanisme softwarement (émulation par programme) en envoyant une impulsion pendant une durée de temps et un repos pour le reste du temps. Par contre, la deuxième solution consiste à bénéficier du PWM implémenté hardwarement dans quelques ports de l'Arduino Uno, et utiliser une fonction qui facilite la programmation. La fonction qui génère le signal PWM est ***analogWrite***, où il est préférable de l'utiliser avec un port doté de PWM. Par exemple si on veut un cycle de service 50%, on passe 127 à la fonction (255/2 = 127) et ainsi de suite. On rappelle que cette fonction accepte une valeur comprise entre 0-255, ce qui exige un mappage du pourcentage dans cet intervalle.

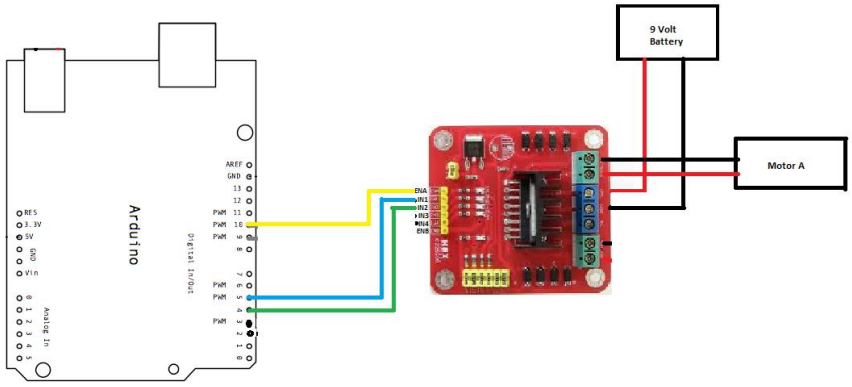

Figure 4. Schéma de branchement du moteur DC, driver L298N et l'Arduino

Dans ce TP, on va commander un moteur à courant continu avec un driver L298N, et celui-ci permet de contrôler deux moteurs dans deux directions ou quatre moteurs dans une seule direction. Donc, on utilise au total trois ports de l'Arduino configurés en tant que sorties (signal sortant de l'Arduino), dont deux entre eux sont destinés pour contrôler la direction et un pour contrôler la vitesse de rotation (PWM).

On branche les deux fils du moteur dans le compartiment A, et les deux fils de l'alimentation dans GND (-) et VMS (+). En outre, on relit les trois fils sortant de l'Arduino avec IN1, IN2 et ENABLE du moteur A (Figure 3.4). Enfin, on tape le code de la Figure 3.5 pour contrôler le moteur dans deux directions, où il est tourné à sa vitesse maximale dans les deux directions, puis à moitié de vitesse maximale.

3. Commande d'un servo-moteur

Le servo-moteur est un cas particulier des moteurs à courant continu, où il peut tourner degré par degré. Il est constitué principalement de moteur à courant continu et d'un ensemble

d’engrenages réducteurs de vitesse, ainsi une carte électronique de contrôle de l’énergie et un capteur de position (capteur rotatif). La plupart des servo-moteurs tournent entre 0° et 180°, mais dans des cas particuliers on trouve des servo-moteurs qui tournent 360°.

```
1.  #define ENABLE_A 10
2.  # define IN1_A 5
3.  # define IN2_A 4
4.
5.  void setup() {
6.      pinMode(ENABLE_A, OUTPUT) ;
7.      pinMode(IN1_A, OUTPUT) ;
8.      pinMode(IN2_A, OUTPUT) ;
9.  }
10.
11. void loop() {
12.     digitalWrite(ENABLE_A, 255)  ;
13.     analogWrite(IN2_A, LOW)  ;
14.     analogWrite(IN1_A, HIGH)  ;
15.     delay(1000) ;
16.     analogWrite(IN1_A, LOW)  ;
17.     analogWrite(IN2_A, HIGH)  ;
18.     delay(1000) ;
19.
20.     digitalWrite(ENABLE_A, 127)  ;
21.     analogWrite(IN2_A, LOW)  ;
22.     analogWrite(IN1_A, HIGH)  ;
23.     delay(1000) ;
24.     analogWrite(IN1_A, LOW)  ;
25.     analogWrite(IN2_A, HIGH)  ;
26.     delay(1000) ;
27. }
```

Figure 5. Code source de contrôle d’un moteur à courant continu avec un driver L298N

Figure 6. Servo-moteur SG90 de la marque TowerPro

Un servo-moteur est équipé de trois fils, dont deux sont dédiés pour l’alimentation et un pour recevoir un signal. En particulier, le fil marron (ou noir) est le GND, le fil rouge (ou orange) est le VCC (ou +5v) et le fil jaune est celui qui reçoit le signal à partir d’un port numérique. Il faut noter que la majorité des servo-moteurs fonctionnent avec 5 volts max (sinon ils se brûlent), sauf les servo-moteurs industriels (gros servo-moteurs).

On note également que plus le couple du servo est élevé plus celui-ci consomme un courant élevé. Néanmoins, il est préférable d'alimenter toujours le servo avec une alimentation externe dotée d'un ampérage élevé, notamment lorsqu'on a plusieurs servos ou un servo à grand couple. En outre, il faut toujours brancher le négatif de l'alimentation externe avec celui de l'Arduino.

Pour manipuler les servos avec l'Arduino, on utilise la bibliothèque <Servo.h>. D'après le référentiel officiel de l'Arduino, l'utilisation de cette bibliothèque désactive le fonctionnement de PWM sur les ports 9 et 10 des Arduino Uno [2], que ce soit des servos branchés sur ces ports ou non.

Pour configurer le port sur lequel le servo est branché, il suffit d'utiliser la fonction dédiée **attach(port)** de l'objet **Servo** au lieu de **pinMode**. Ensuite, pour tourner le servo à un angle donné, on utilise la fonction **write(angle)** de l'objet **Servo**. Malheureusement, cette fonction n'est pas fiable avec tous les servos, car certains constructeurs ne respectent pas les normes, et par conséquent le servo ne sera pas positionné correctement. Par exemple, si on le positionne à 90°, il peut être positionné à moins de cette valeur ou plus (même avec une virgule). Comme une solution alternative, la communauté a mis en disposition une autre fonction qui travaille en bas niveau du signal envoyé, c'est la fonction **writeMicroseconds(microseconds)** de l'objet **Servo**. La majorité des servos qui suivent les normes fonctionnent entre 1000µs (0°) et 2000µs (180°). Donc, on fait la règle des trois pour trouver le nombre de micro secondes à envoyer correspondant à l'angle voulu (e.g. 90° = 1500µ).

```
1.  #include<Servo.h>
2.  #define PIN_SERVO 12
3.
4.  Servo mon_servo ;
5.
6.  void setup() {
7.      mon_servo.attach(PIN_SERVO) ;
8.      mon_servo.write(0) ;
9.  }
10.
11. void loop() {
12.     mon_servo.write(90) ;
13.     delay(1000) ;
14.
15.     mon_servo.writeMicroseconds(2000) ;
16.     delay(1000) ;
17. }
```

Figure 7. Code source de manipulation d'un servo-moteur avec deux fonctions différentes

Après l'exécution du code de la Figure 3.7, nous voyons que le servo se positionne à l'angle 0° (s'il n'est pas dans cet angle par défaut). Ensuite, il se positionne à l'angle 90° et **180**° consécutivement, et comme la fonction *loop* est une boucle infinie, le servo répète les mêmes actions continuellement. Dans le cas où le servo ne tourne pas 180°, on change la valeur 2000 (ligne 15 Figure 3.7) jusqu'à tomber sur le nombre exact.

Nous remarquons que le servo bouge d'un angle à un autre à une vitesse assez rapide (presque non percevable à l'œil), ce qui causera une instabilité de mouvement du corps attaché au servo (un choc peut endommager le montage ou le servo lui même). Donc, il est conseillé de bouger le servo degré par degré jusqu'à atteindre l'angle souhaité. En outre, il est préférable d'ajouter un petit retard (une dizaine de milli secondes) entre deux degrés successifs (Figure 3.8), bien évidemment dans le cas où il n'y a aucune contrainte de temps dans le système réalisé.

```
1.  #include<Servo.h>
2.  #define PIN_SERVO 12
3.
4.  Servo mon_servo ;
5.
6.  void setup() {
7.      mon_servo.attach(PIN_SERVO) ;
8.      mon_servo.write(0) ;
9.  }
10.
11. void loop() {
12.     for(int i=0 ; i<=90 ; i++) {
13.         mon_servo.write(i) ;
14.         delay(10) ;
15.     }
16. }
```

Figure 8. Code source de manipulation d'un servo-moteur degré par degré

4. Commande d'un moteur Brushless

Le moteur Brushless est une évolution du moteur à courant continu dont l'architecture interne est différente. Plus particulièrement, les aimants permanents sont fixes et placés sur le rotor qui tend à suivre un champ magnétique tournant. Le moteur Brushless est équipé de trois fils (trois phases) dont chacun est relié à la tension et à la masse à la fois [3]. En outre, la commande de ce type de moteur nécessite un driver spécial pour passer le courant entre deux fils en alternance, et peut contrôler un moteur triphasé sans capteur en variant la vitesse via le signal PWM.

Figure 9. Exemple de moteur Brushless utilisé dans les drones

Les moteurs Brushless se diffèrent par le nombre de Kv (vitesse) et le voltage de fonctionnement. Généralement, on utilise le driver ESC[2] qui contient deux fils d'alimentation d'entrée, un fils d'acquisition de signal PWM pour contrôler la vitesse et trois fils de sortie pour alimenter le moteur triphasé.

Figure 10. Driver ESC pour commander un moteur Brushless

Le contrôleur ESC étant doté d'une puce interne, il peut être configurable, notamment la plage d'impulsion (en termes de micro secondes). Alors, il faut noter que la plupart des ESC sont configurés pour fonctionner entre 1000-2000µs. Alors, on peut bénéficier de la bibliothèque <Servo.h> pour contrôler le moteur Brushless en utilisant la fonction **writeMicroseconds**.

```
1.  #include<Servo.h>
2.  #define PIN_BRUSHLESS 11
3.
4.  Servo brushless ;
5.
6.  void setup() {
7.      brushless.attach(PIN_BRUSHLESS) ;
8.  }
9.
10. void loop() {
11.     brushless.writeMicroseconds(1200) ;      // fonctionner à 20% de la vitesse max
12.     delay(1000) ;
13.     brushless.writeMicroseconds(1500) ;      // fonctionner à 50% de la vitesse max
14.     delay(1000) ;
15.     brushless.writeMicroseconds(2000) ;      // fonctionner à 100% de la vitesse max
16.     delay(1000) ;
17. }
```

Figure 11. Code source de commande de moteur Brushless en variant la vitesse

5. Commande d'un moteur pas-à-pas

Enfin, le quatrième type des moteurs fonctionnant en courant continu est le moteur pas-à-pas qui est basé sur une technologie totalement différentes des autres moteurs, ainsi il est doté d'une très grande précision de rotation. Il y a trois types de moteur pas-à-pas tels que moteur à

[2] ESC est l'abréviation de *Electronic Speed Controller*

réluctance variable, moteur à aimants permanents et le moteur hybride combinant les deux. Les trois types sont des moteurs à quatre phases, donc équipés de quatre fils d'alimentation et de contrôle ce qui exige l'utilisation d'un driver spécial. Cependant, le but principal est de contrôler le nombre de pas effectués par le moteur et non la vitesse, car ce type de moteur généralement tourne lentement.

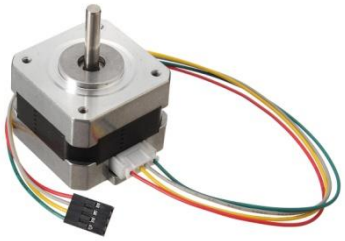

Figure 12. Moteur pas-à-pas Nema17

Il y a une multitude de drivers dédiés qui se diffèrent selon la puissance du moteur (voltage et ampérage), ainsi le nombre de pas par seconde. Dans ce TP, on utilise seulement le driver A4988 qui permet de contrôler des petits moteurs pas-à-pas (typiquement Nema 17), et qui supporte jusqu'à 35v et 2A.

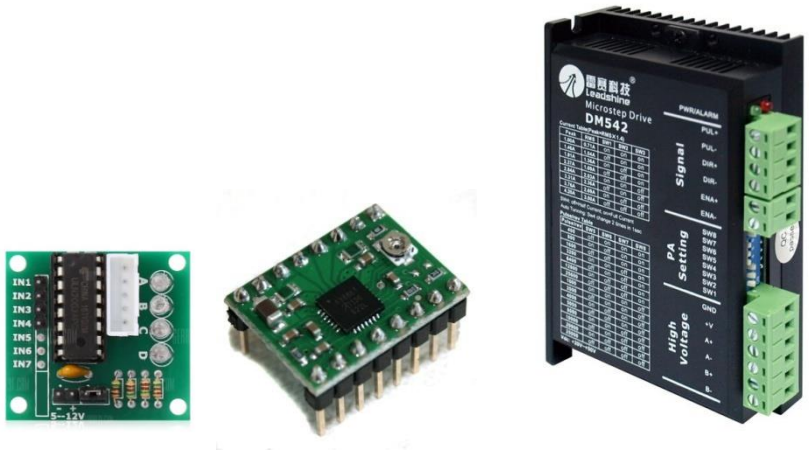

Figure 13. Différentes catégories de driver de moteur pas-à-pas. A droite c'est driver ULN2003, au centre un driver A4988 et à gauche à driver DM542

Pour réaliser ce TP, on a besoin d'une plaque d'essai, des fils de prototypage, un driver A4988, un moteur pas-à-pas, une alimentation 12v et un Arduino Uno. La première chose à faire c'est de brancher le moteur au driver dans les ports A1, A2, B1 et B2. En d'autres termes, on branche le premier pôle du moteur dans A1 et A2, puis le deuxième pôle dans B1 et B2. Ensuite, on connecte le port STP à un port numérique de l'Arduino (exemple port 3) et le port DIR (pour la direction) à un autre port numérique de l'Arduino (exemple port 2). Enfin, la dernière chose à brancher est l'alimentation externe (12v jusqu'à 35v).

Pour faire tourner le moteur à un pas, on envoie une impulsion à STP pendant une durée, puis on remet le port à 0. Donc, si on veut bouger à plusieurs pas, on fait une boucle. En outre, pour contrôler la direction soit on envoie HIGH ou LOW au port DIR.

```
1.  #define PIN_DIR 2
2.  #define PIN_STEP 3
3.
4.  void setup() {
5.      pinMode(PIN_DIR, OUTPUT);
6.      pinMode(PIN_STEP, OUTPUT);
7.  }
8.
9.  void loop() {
10.   digitalWrite(PIN_DIR, HIGH);
11.
12.   for (int i = 0; i < 200; i++) {
13.     digitalWrite(PIN_STEP, HIGH);
14.     delayMicroseconds(2000);
15.     digitalWrite(PIN_STEP, LOW);
16.     delayMicroseconds(2000);
17.   }
18. }
```

Figure 11. Code source de contrôle de moteur Brushless en variant la vitesse

Références

[1] Introduction au signal PWM. https://barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation [consulté en 09/2019]

[2] Référence de la bibliothèque Servo. https://www.arduino.cc/en/reference/servo [consulté en 09/2019]

[3] Modélisation des moteurs BrushLess. https://eduscol.education.fr [consulté en 09/2019]