Mouvement périodique d'un moteur pas à pas

1. Introduction

Ce document montre comment piloter un moteur pas-à-pas avec un arduino pour obtenir un mouvement périodique. Il s'agit de faire varier la position angulaire en suivant une fonction périodique, par exemple une sinusoïde, avec une période variable.

Le moteur est commandé par un circuit spécialisé qui se charge de générer la commande d'un pont DMOS. Ce circuit est commandé par une impulsion sur l'entrée STEP pour chaque pas moteur (ou micropas), l'entrée DIR définissant le sens de rotation.

Nous avons utilisé un Allegro A4988, qui peut générer des seizièmes de pas et peut fournir 2 A par phase. La carte pour arduino Dual Bipolar Stepper Motor Shield fabriquée par dfrobot comporte ces deux circuits et permet donc de piloter deux moteurs. Pour le premier moteur, l'entrée STEP est reliée à la borne D5 de l'arduino, l'entrée DIR à la borne D4.

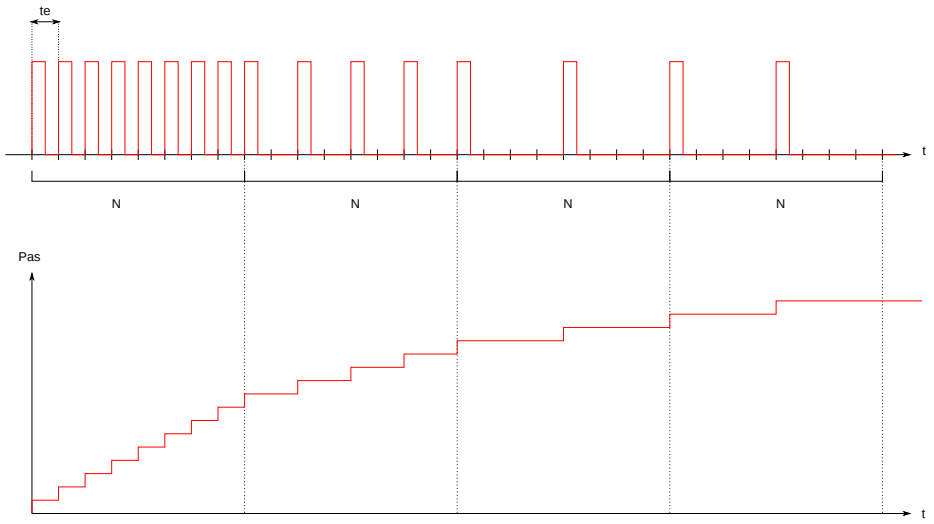
2. Principe

La forme d'onde à générer (position angulaire en fonction du temps) est divisée en quatre quart de période. Soit T la période. Il s'agit de définir une séquence de pas (ou de micropas) à appliquer pendant la durée $T/4$. Cette séquence est répliquée quatre fois pour réaliser la période complète.

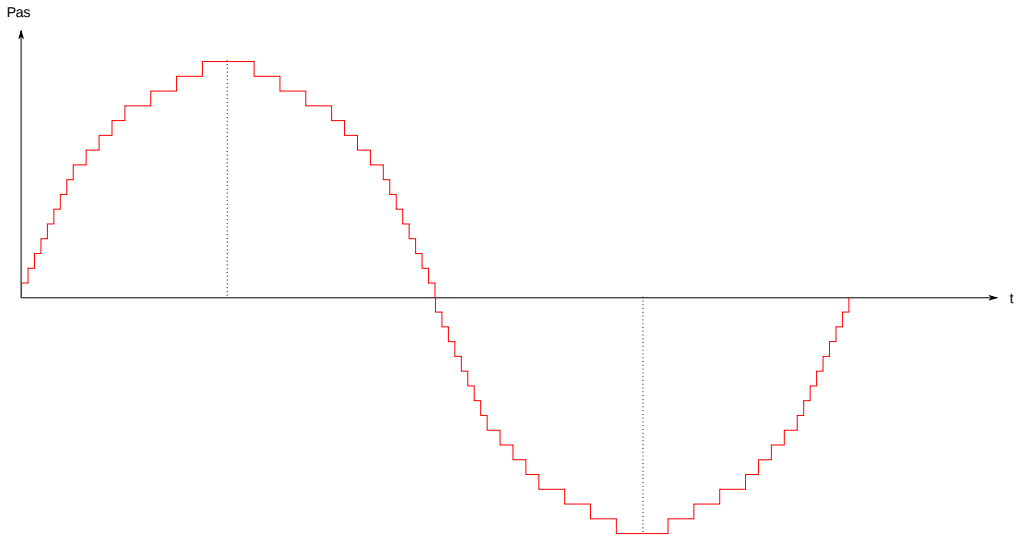
La durée $T/4$ est divisée en P intervalles. Sur chacun de ces intervalles, les pas sont effectués à une cadence constante. Pour obtenir cela, on divise chaque intervalle en N pour définir une période d'échantillonnage :

$$t_e = \frac{T}{4PN} \tag{1}$$

L'échantillonnage est effectué par des interruptions du microcontrôleur à la période $t_e/2$. Dans l'intervalle de durée Nt_e, les impulsions de commande des pas (STEP) sont espacées d'un délai Dt_e avec $D < N$. Le nombre de pas est donc N/D. La figure suivante montre un exemple avec $P = 4$ et $N = 8$. Les délais sont 1,2,4,4.



La figure suivante montre la réplication des pas moteurs permettant d'obtenir un mouvement périodique :



Lors du changement de direction du moteur (à $t = T/4$ et $t = 3T/4$), le premier pas de l'intervalle doit être enlevé, afin d'assurer la parfaite périodicité de la position angulaire du moteur. Les délais sont stockés dans un tableau. Si 0,1,2,3 sont les indices de ce tableau, la séquence de délais à appliquer est 0,1,2,3,3,2,1,0,0,1,2 etc.

3. Programme arduino

Le programme suivant fonctionne sur les arduinos MEGA, UNO et YUN (ou équivalents). Une séquence de pas est programmé par défaut mais elle peut être modifiée depuis le programme python présenté plus loin.

paspas-periodique.ino

```
#include "Arduino.h"
#define MVT_PERIODIQUE 100

int DIR = 4;  // PG5 (portG sortie 5) sur MEGA, PC4 sur UNO
int STEP = 5; // PE3 (port E sortie 3) sur MEGA, PC5 sur UNO


uint16_t N = 8;
uint8_t P = 4;
uint16_t delais[32] = {1,2,4,4}; // délais entre les micropas

int8_t sens_rotation[4] = {1,-1,-1,1};
uint16_t indice_delai;
int8_t increment_indice_delai;
uint16_t compteur_delai;
uint16_t compteur_n;
uint16_t compteur_p;
uint8_t compteur_quart_periode;
uint16_t compteur_periode;
uint16_t nombre_periodes;
uint8_t flip;

uint16_t diviseur[6] = {0, 1, 8, 64, 256, 1024};
```

La fonction suivante programme le Timer 1 pour qu'il génère des interruptions périodiques. La période est donnée en microsecondes.

```
void timer1_init(uint32_t period) {
  TCCR1A  = 0;
  TCCR1B = (1 << WGM12); // mode CTC avec OCR1A pour le maximum
  uint32_t top = (F_CPU / 1000000 * period);
  int clock = 1;
  while ((top > 0xFFFF) && (clock < 5)) {
    clock++;
    top = (F_CPU / 1000000 * period / diviseur[clock]);
  }
  OCR1A = top;
  TIMSK1 = (1 << OCIE1A);
  TCCR1B |= clock;
}
```

La fonction suivante est appelée à chaque interruption. La période d'interruption est en fait $t_e/2$: il y a deux interruptions pour la durée t_e, la première pour éventuellement faire passe la sortie STEP au niveau haut, la seconde pour la remettre au niveau bas. Le circuit A4988 réagit aux fronts montants. La variable `flip` indique quel cas il faut traiter.

La variable `compteur_delai` contient initialement le délai entre deux pas. Elle est décrémentée à chaque période t_e. Lorsqu'elle atteint 0, une impulsion doit être générée sur STEP. La variable `compteur_n` permet de compter les N intervalles de durée t_e. La variable `compteur_p` permet de compter les quatre quart de période. Lorsque la fin de quart de période est atteint, il faut éventuellement changer le sens de rotation du moteur en modifiant la sortie DIR, et désactiver la génération de l'impulsion STEP dans ce cas. Lorsqu'une période complète est atteinte, on incrémente le compteur de périodes. Lorsque le nombre de périodes demandé est atteint, on stoppe le Timer.

Les sorties numériques D4 et D5 sont modifiées directement avec les registres PORTx du microcontrôleur et non pas avec la fonction `digitalWrite`. Pour adapter ce programme à une autre carte de commande moteur (qui utilise d'autres bornes que D4 et D5), il faut consulter la table de correspondance des bornes de l'arduino pour savoir à quel port d'entrée-sortie les bornes utilisées sont reliées.

```
ISR(TIMER1_COMPA_vect) {
  char step = 0;
  char init_delai=0;
  if (flip==0) {
      flip=1;
      compteur_delai--;
      if (compteur_delai==0) step=1;
      compteur_n++;
      if (compteur_n==N) {
          compteur_n=0;
          compteur_p++;
          if (compteur_p==P) {
              indice_delai += increment_indice_delai;
              compteur_p = 0;
              increment_indice_delai = -increment_indice_delai;
              compteur_quart_periode++;
              if ((compteur_quart_periode==1)||(compteur_quart_periode==3)) {
                step=0;
                init_delai = 1;
              }
              if (compteur_quart_periode == 4) {
                  compteur_quart_periode = 0;
                  compteur_periode++;
                  if (compteur_periode == nombre_periodes) {
                      TCCR1B = 0; // fin du mouvement
                    // D5 = LOW
                    #if defined(__AVR_ATmega2560__)
                      PORTE &= ~(1 << PORTE3);
                    #elif defined(__AVR_ATmega32U4__)
                      PORTC &= ~(1 << PORTC6);
                    #else
                      PORTC &= ~(1 << PORTC5);
                    #endif
                      Serial.write(1);
```

```
                }
            }
            if (sens_rotation[compteur_quart_periode]==1) {
                // D4 = HIGH
                #if defined(__AVR_ATmega2560__)
                    PORTG |= (1 << PORTG5);
                #elif defined(__AVR_ATmega32U4__)
                    PORTD |= (1 << PORTD4);
                #else
                    PORTC |= (1 << PORTC4);
                #endif
            }
            else {
                // D4 = LOW
                #if defined(__AVR_ATmega2560__)
                    PORTG &= ~(1 << PORTG5);
                #elif defined(__AVR_ATmega32U4__)
                    PORTD &= ~(1 << PORTD4);
                #else
                    PORTC &= ~(1 << PORTC4);
                #endif
            }
        }
        indice_delai += increment_indice_delai;
    }
    if (step) {
           step=0;
           // D5 = HIGH
           #if defined(__AVR_ATmega2560__)
              PORTE |= (1 << PORTE3);
           #elif defined(__AVR_ATmega32U4__)
              PORTC |= (1 << PORTC6);
           #else
              PORTC |= (1 << PORTC5);
           #endif
           compteur_delai = delais[indice_delai];
    }
    if (init_delai) {
        init_delai=0;
        compteur_delai = delais[indice_delai];
    }
}
else {
    // D5 = LOW
    #if defined(__AVR_ATmega2560__)
            PORTE &= ~(1 << PORTE3);
    #elif defined(__AVR_ATmega32U4__)
            PORTC &= ~(1 << PORTC6);
```

```
      #else
              PORTC &= ~(1 << PORTC5);
      #endif
      flip=0;
  }
}
```

La fonction `setup` initialise les compteurs et programme la séquence par défaut définie dans l'entête. La variable `T` définie la période en millisecondes.

```
void setup() {
    char c;
    Serial.begin(115200);
    Serial.setTimeout(0);
    c = 0;
    Serial.write(c);
    c = 255;
    Serial.write(c);
    c = 0;
    Serial.write(c);
    pinMode(DIR,OUTPUT);
    pinMode(STEP,OUTPUT);
    digitalWrite(DIR,HIGH);
    digitalWrite(STEP,LOW);

    compteur_n = 0;
    compteur_p = 0;
    compteur_delai = 0;
    indice_delai = 0;
    compteur_delai = delais[indice_delai];
    increment_indice_delai = 1;
    compteur_periode = 0;
    flip=0;

    nombre_periodes = 100;
    float T = 1000.0;
    float periode_echant = T/(4*N*P);
    timer1_init(periode_echant*1e3*0.5);
}
```

Afin de piloter le moteur depuis un ordinateur, on ajoute une fonction de communication série pour définir la séquence. La fonction suivante lit les informations suivantes sur le port série et lance la génération :

- ▷ N : entier 16 bits.
- ▷ P : entier 8 bits.
- ▷ Le tableau des P valeurs de délais : entiers 16 bits.

- ▷ La période T en millisecondes : entier 16 bits.
- ▷ Le nombre de périodes : entier 16 bits.

```
void lecture_mvt_periodique() {
    uint32_t c1,c2,c3,c4;
    while (Serial.available()<2) {};
    c1 = Serial.read();
    c2 = Serial.read();
    N = ((c1<<8) | c2);
    while (Serial.available()<1) {};
    P = Serial.read();
    for (int p=0; p<P; p++) {
        while (Serial.available()<2) {};
        c1 = Serial.read();
        c2 = Serial.read();
        delais[p] = ((c1<<8) | c2);
    }
    while (Serial.available()<2) {};
    c1 = Serial.read();
    c2 = Serial.read();
    uint16_t T = ((c1<<8) | c2); // période en millisecondes
    while (Serial.available()<2) {};
    c1 = Serial.read();
    c2 = Serial.read();
    nombre_periodes = ((c1<<8) | c2);
    digitalWrite(DIR,HIGH);
    digitalWrite(STEP,LOW);
    compteur_n = 0;
    compteur_p = 0;
    compteur_delai = 0;
    indice_delai = 0;
    compteur_delai = delais[indice_delai];
    increment_indice_delai = 1;
    compteur_periode = 0;
    float periode_echant = ((float)T)/(4*N*P);
    flip=0;
    timer1_init(periode_echant*1000*0.5);
}
```

La fonction suivante lit le port série pour savoir si une commande est envoyée :

```
void lecture_serie() {
  char com;
  if (Serial.available()>0) {
    com = Serial.read();
    if (com==MVT_PERIODIQUE) lecture_mvt_periodique();
  }
```

```
}
```

La fonction `loop` effectue une lecture du port série :

```
void loop() {
   delay(100);
   lecture_serie();
}
```

4. Programme python

Le programme python permet de programmer une séquence de pas.
paspas-periodique.py

```
# -*- coding: utf-8 -*-
import serial
import numpy
import time

class Arduino():
    def __init__(self,port):
        self.ser = serial.Serial(port,baudrate=115200)
        c_recu = self.ser.read(1)
        while ord(c_recu)!=0:
            c_recu = self.ser.read(1)
        c_recu = self.ser.read(1)
        while ord(c_recu)!=255:
            c_recu = self.ser.read(1)
        c_recu = self.ser.read(1)
        while ord(c_recu)!=0:
            c_recu = self.ser.read(1)
        self.MVT_PERIODIQUE = 100

    def close(self):
        self.ser.close()

    def write_int16(self,v):
        v = numpy.int16(v)
        char1 = (v & 0xFF00) >> 8
        char2 = (v & 0x00FF)
        self.ser.write(chr(char1))
        self.ser.write(chr(char2))

    def write_int32(self,v):
        v = numpy.int32(v)
        char1 = (v & 0xFF000000) >> 24
```

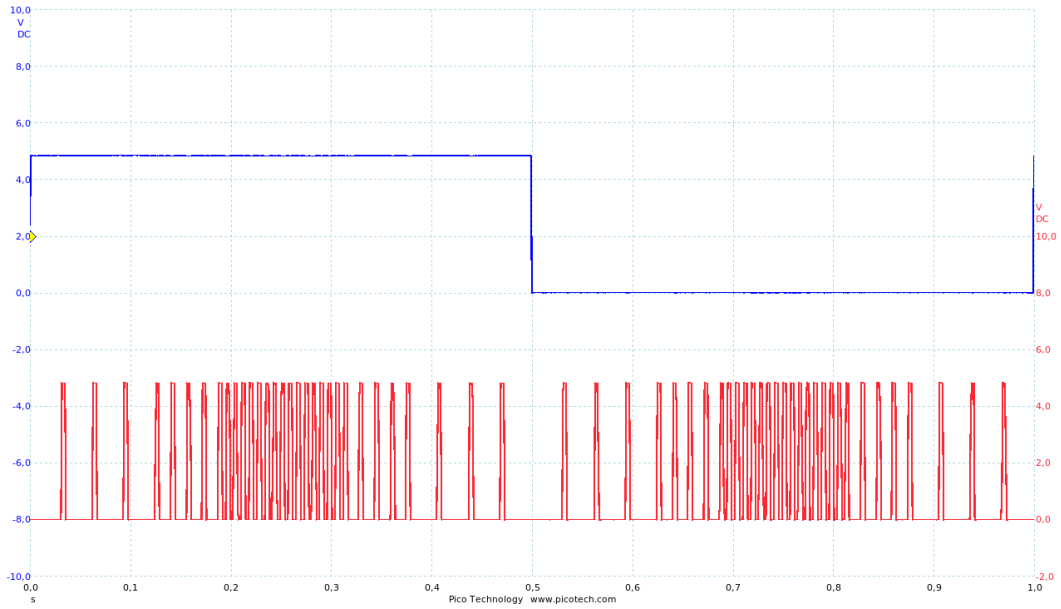
```
        char2 = (v & 0x00FF0000) >> 16
        char3 = (v & 0x0000FF00) >> 8
        char4 = (v & 0x000000FF)
        self.ser.write(chr(char1))
        self.ser.write(chr(char2))
        self.ser.write(chr(char3))
        self.ser.write(chr(char4))


    def mvt_periodique(self,N,delais,T,num_periodes):
        self.ser.write(chr(self.MVT_PERIODIQUE))
        self.write_int16(N)
        P = len(delais)
        self.ser.write(chr(P))
        for p in range(P):
            self.write_int16(delais[p])
        self.write_int16(T);
        self.write_int16(num_periodes)

    def mvt_sinus(self,N,P,A,T,num_periodes):
        dtheta = numpy.pi/2/P
        theta = 0
        delais = []
        for p in range(P):
            num_pas = A*(numpy.sin(theta+dtheta)-numpy.sin(theta))
            theta += dtheta
            delais.append(int(N*1.0/num_pas))
        self.mvt_periodique(N,delais,T,num_periodes)

    def wait(self):
        self.ser.read(1)
```

Voici un exemple d'utilisation :

```
ard = Arduino(4)
ard.mvt_periodique(32,[1,2,4,4],800,20)
ard.wait()
ard.close()
```

5. Tests du générateur de pas

Pour ce type de programme qui génère des impulsions sur des sorties numériques, il est impératif de commencer par un test sur oscilloscope. Voici les signaux des sorties D4 (bleu) et D5 (rouge) pour le cas $N = 8, P = 4$ avec les délais $1, 2, 4, 4$ et une période de 1000 ms.



L'amplitude angulaire du mouvement est égale à l'angle de rotation pour un pas (ou un micropas) multipliée par le nombre de pas générés pendant un quart de période. Ce nombre est proportionnel à N. En doublant N, ou double donc l'amplitude. Pour une période T donnée, l'augmentation de N s'accompagne d'une diminution de la période d'échantillonnage.

Par exemple avec la séquence $1, 2, 4, 4$, l'amplitude est de 16 pas pour $N = 8$. Si le circuit de commande est réglé pour générer des $1/8$ ième de pas et si le moteur comporte 400 pas par tours, cela fait une amplitude de $16/8 = 2pas$, soit $1,8$ degrés. Pour $T = 1000\ ms$, la période d'échantillonnage est $t_e = T/(4NP) = 7,8\ ms$. Avec $N = 32$, on obtient une amplitude 4 fois plus grande mais une période d'échantillonnage 4 fois plus petite (environ 2 ms).

Lorsque la période d'échantillonnage atteint la milliseconde, il faut faire des tests pour vérifier que le moteur suit bien les pas demandés. Pour obtenir une réponse rapide du moteur, il faut augmenter au maximum la tension d'alimentation (30 V) et ajuster le courant moteur avec le potentiomètre prévu pour cela (le courant est régulé par découpage).