Comprendre le PWM et PFM

Liens sous www.didel.com/prof/PwmPfm.html
Programmes sous www.didel.com/prof/PwmPfm.zip

Note pédagogique

Le PFM permet des vitesses lentes, voire très lentes, avec des moto-réducteurs ayant un mauvais rendement, comme les moteurs jouets utilisés pour des robots.
Le PFM n'est pas supporté par les timers de microcontrôleurs et il est inconnu dans la documentation Arduino. Faut-il en parler?

Oui si le but est de comprendre les concepts liés au contrôle de vitesse et de puissance, et pas simplement savoir qu'il y a un digitalWrite et un analogWrite pour commander une sortie.

Une première chose à comprendre est comment le processeur peut générer une tension, un courant, une énergie variable à l'intention d'un dispositif.
Ces notions sont-elles importantes ? Pour programmer, non, pour acquérir une pensée computationnelle, oui.

Mais on n'a pas le temps de parler électro-mécanique ; on veut apprendre à programmer des application temps réel. C'est dans les exemples qu'il faudra élargir sa vision. L'étude du PWM, du PFM et si possible du BCM va montrer des approches différentes, qui existent parce qu'elles répondent à des exigences différentes.

Une difficulté pour l'enseignant est de choisir le niveau de détail permise par le temps à disposition. On peut utiliser des fonctions et agir sur leur paramètre, on peut vouloir comprendre leur algorithme, on peut chercher à écrire proprement ces algorithme en C ou autre language.

Notre choix ici est d'expliquer le principe du PWM et du PFM et montrer comment coder et tester leur algorithme. Dans une 2^{e} étape, on utilise des fonctions et librairies qui n'ont pas été analysées, mais on connaît le mécanisme d'analyse et on fait confiance.

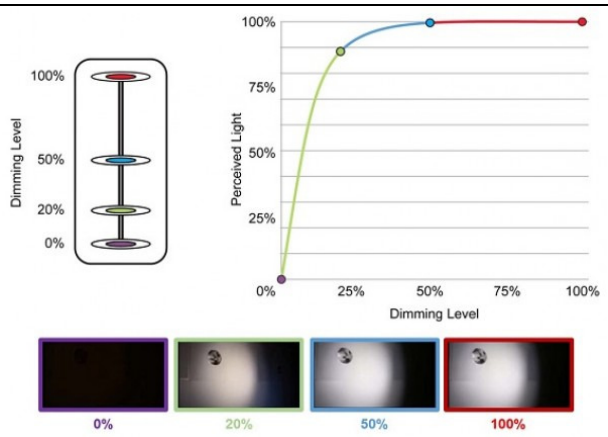
PWM Arduino

Le PWM sur Arduino n'existe que sur 5 pins (dans le cas de l'Arduino Uno) et sa librairie est automatiquement installée. Chargez et compilez le programme "BareMinimum" des exemples Arduino. Il prend 426 octets. Ajoutez l'instruction " analogWrite(6,200); ", cela prend 890 octets.
Ajoutez "delay(1000);" le code monte à 1048 octets.
Ceci montre d'une part qu'Arduino ne charge que les librairies qui sont appelées, et que des librairies très simples prennent beaucoup de place, ce qui ne pose pas de problème.
Le PWM d'Arduino travaille par interruption et n'est pas bloquant. La fonction delay(); mets aussi en route des interruptions (pour les fonctions millis() et micros()) mais elle est bloquante.

Avec Arduino, on programme le PWM "naturellement", sans se poser de question. Avec le PFM, il faut comprendre les options que l'on a, l' objectif de formation est intéressant.

PWM avec une Led

Pour agir sur l'intensité lumineuse ou sur la vitesse d'un moteur, il faut réduire l'énergie fournie en diminuant la tension (ce qui est volumineux et a un mauvais rendement) ou en pulsant la commande de puissance. Un transistor en tout-ou-rien est efficace. Si on le règle pour qu'il laisse passer la moitié de l'énergie, il va dissiper en chaleur l'autre moitié.

Leds et moteurs n'ont pas les mêmes comportements. Les Leds réagissent quasi instantanément (moins d'une microsecondes) et ont une réponse linéaire dans leur domaine d'utilisation (l'énergie lumineuse émise est proportionnelle à l'énergie électrique). Mais notre oeil n'est pas linéaire! L'oeil est très sensible aux faibles intensités lumineuses ; à 50% de PWM ou PFM, on a déjà l'impression d'un 100% d'intensité. Le PWM est bien adapté pour les Leds, mais il faut appliquer une correction de linéarité, comme expliqué sous	

https://www.eldoled.com/support/learning-center/why-you-need-dimming-curves/

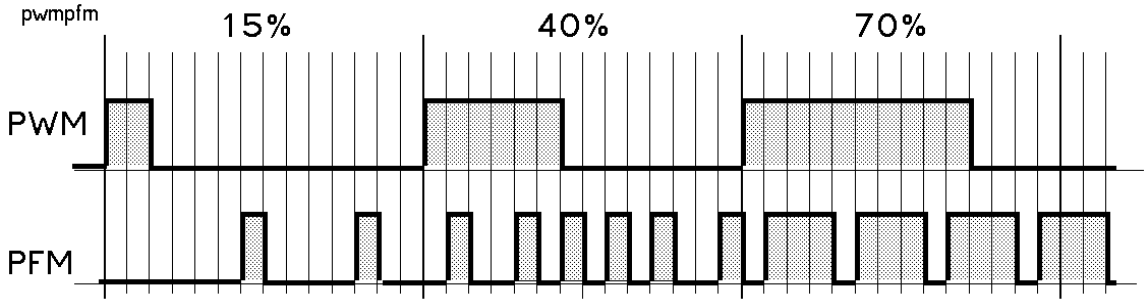
PWM avec un moteur

Un moteur a en comparaison beaucoup de défauts. Il y a du frottement, de l'inertie et une force contre-électomotrice. On ne connaît pas les caractéristiques des moteurs achetés en Chine, les capteurs éventuels ont une trop mauvaise résolution pour envisager une modélisation, et l'objectif est de programmer un robot "intelligemment", et pas d'appliquer la théorie du réglage.

L'inertie peut être négligée. La vitesse de rotation dépend du couple de freinage. Il faut construire proprement pour que ce couple soit régulier et qu'il y ait proportionnalité entre le couple et la vitesse. Mais à faible vitesse, le frottement à l'intérieur de moteur devient important et un moteur bon marché démarre à 30-40% de PWM, et stoppe à 20% de PWM quand on réduit la valeur.

PFM à la rescousse

C'est là que le PFM est intéressant. La figure ci-dessous montre la différence.



Le PFM génère des impulsions de durées constantes et se rapprochent jusqu'à 50%. Ensuite se sont des absences d'impulsions qui se multiplient. Pour qu'un mauvais moteur puisse tourner lentement, il faut que l'énergie contenue dans une impulsion isolée soit suffisante pour faire "décoller" le moteurs. Quelques millisecondes suffisent pour un petit moteur, et c'est facile à calibrer.

La figure peut faire croire que les impulsions PWM sont plus longues que celles en PFM. C'est que le cycle PWM est anormalement lent. Si les impulsions PFM font 2ms, le cycle est de 30ms, donc la fréquence de répétition 33 Hz. L'inertie et le réducteur du moteur vont absorber les a-coup d'avance.

Le PWM d'Arduino a une fréquence de 500 Hz (très lent par rapport à du PWM industriel), et il y a 256 niveaux de puissance possible. L'impulsion minimale dure 8 µs. Donc, avec un PWM de 50=20% de puissance, l'impulsion dure 0.4ms et le moteur ne décolle pas.

Nous ferons du PFM en 20 et 80 pas. Bizarre ces valeurs ? Quatre raisons:

- On veut travailler avec des vitesses positives et négatives. Le PWM qui oblige d'utiliser le type `int`.
- On n'a pas besoin d'une grande précision pour notre robot. Trois, quatre vitesses sont suffisantes, comme en voiture : 30,50,80,120 km/h.

- S'il y a plus de vitesses, elles ne doivent pas être linéaires. Entre 20 et 30km/h, la différence est "plus grande" qu'entre 110 et 120.
- Les vitesses négatives doivent être signées, c'est le plus efficace, et si on code en byte, il faut éviter le dépassement de capacité qui change le signe quand on passe de 127 à 128 (0x7F à 0x80). La vitesse -1 est codée 0xFF; Le processeur travaille en binaire et il faut parler sa langue.

PWM bloquant

Pour comprendre le PWM et faire un exercice "temps réel" avec la Led13 par exemple, on peut utiliser deux délais.

```
byte pwm = 20;      // valeur 0 à 100  fréquence PWM 1kHz période 1ms
void loop() {
   LedOn; delayMicroseconds (10 * pwm);
   LedOn; delayMicroseconds (10 * (100-pwm));
}
```

Dans ce cas on est 0.2ms ON et 0.8ms OFF

On peut faire autre chose dans cette boucle, par exemple surveiller un poussoir pour modifier l'intensité. Le problème est de modifier la valeur PWM lentement (compter les cycles de 1ms) et de saturer ou osciller. On ne peut utiliser que des `if` avec des compteurs ou décompteurs, éventuellement des `for`.

PWM non bloquant

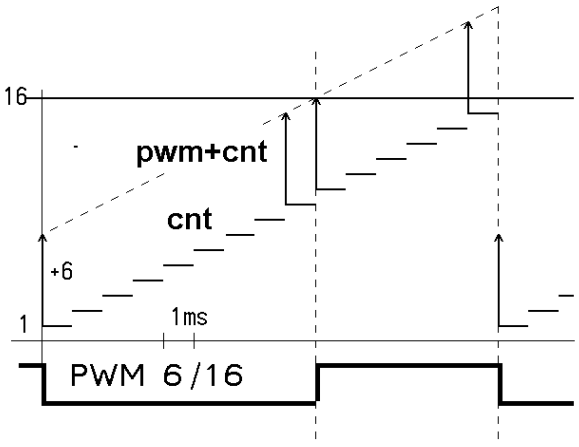
On bloque nécessairement le programme principal, mais pour des temps fréquents et court. C'est du temps partagé.

Le figure a 16 niveaux de PWM. Toute les ms le processeur décide s'il faut activer la sortie PWM suite à une simple addition..

```
void loop() {
   if ((pwm+cnt)>16) {LedOn; }
   if (cnt++>16) {cnt=0;LedOff;}
   //. . .  autres taches
   delayMicroseconds (500);
}
```

Programme complet sous PwmTest0.ino dans le zip

En agitant la carte, on voit le clignotement.



Que se passe-t-il si la valeur pwm donnée est supérieure à 16? Le signal est continuellement à "1", il n'y a pas besoin de saturer.

Evidemment pour être général, on ajoute la ligne #define MaxPwm 16

Les 2 lignes qui génèrent le PWM ne prennent que 2 µs, plus 0.2 à 10 µs pour agir sur la Led, selon le fichier de définition (bitSet rapide, digitalWrite lent).

Les autres tâches doivent aussi durer quelques microsecondes, ce qui est en général le cas avec des machines d'état. On raccourci le délai pour que la boucle dure ~500 µs.

Si MaxPwm est augmenté, la période augmente et le clignotement devient visible.

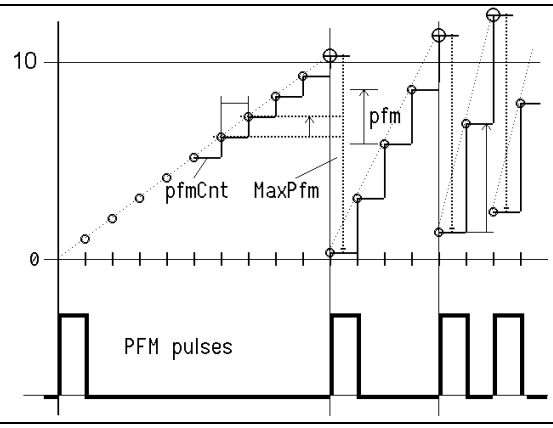
N'est-ce pas un joli exemple ? Algorithme simple, représentation graphique, codage élégant.

PFM bloquant

Supposons que le moteur a besoin d'impulsions de 5ms avec 10 niveaux de PFM. La valeur de la sortie est calculée toutes les 5ms.L'exemple a 10 niveaux de PFM.

La première partie du diagramme montre l'évolution avec une valeur pfm=1. Après 10 fois le temps d'échantillonnage de 5ms, on dépasse la valeur et il faut

1) Activer une impulsion
2) Soustraire la valeur 10



Si la valeur pfm augmente, on arrive plus vite au dépassement et les impulsions de rapprochent. Si le pfm vaut 10, on a chaque fois un dépassement, donc le signal est continu. Il ne faut pas dépasser cette valeur, on saturera la valeur programmée à `MaxPfm`

```
    void loop () {
       if (pfm > MaxPfm) { pfm=MaxPfm; } // sature
        pfmCnt += pfm;
        if (pfmCnt > MaxPfm) { pfmCnt -= MaxPfm ; LedOn; }
        else { LedOff; }
        //. . .   autres tâches
        delay (5);
  }
```

Programme complet sous PfmTest0.ino dans le zip

En comparant avec le PWM, on voit que l'on a beaucoup plus de temps pour exécuter d'autres tâches.

Pour continuer
L'approche ici a été de comprendre deux algorithmes et tester avec le minimum d'instructions. On peut continuer en demandant la création des fonctions, en faisant varier l'intensité, en multipliant les Leds.

Plus sur le PWM
Dans tous les processeurs, le PWM est généré par une logique interne au processeur et la programmation des timer multifonctionnels et délicate. On utilise des librairies qui mettent à disposition des fonctions comme le analogWrite.
Une première difficulté est de travailler avec des vitesses positives et négatives et il faut choisir entre les types char (int8_t) ou int (int16_t). Ne pouvoir agir en PWM que sur l'une des deux sorties du moteur est documenté pour Arduino sous robots/MotorControl.pdf

Plus sur le PFM
Didel a abondamment documenté le PFM, trop méconnu. Il se gère sur Arduino en utilisant le Timer2.
kidules/PwmPfm.pdf explique l'algorithme et l'appel de la fonction pour 2 moteurs par interruption.
La première partie du document diduino/PfmPratique.pdf est très progressive et les programmes du zip sont rapidement chargés si on les demande dans un .ino déjà chargé ou si on a mis le zip en librairie Arduino.
Dans une application, il est important de modifier la vitesse de façon progressive. diduino/PfmPratique.pdf détaille une solution en 2e partie.

Concepts, notions vues ou revisitées
Provisoire - à compléter
Différences entre Leds et moteurs
Conditions de démarrage d'un moteur
Fonction bloquante - non bloquante
Diagramme des temps
Saturation
Nombres négatifs
Interruptions
Variables globales
A compléter

jdn 130215/180829