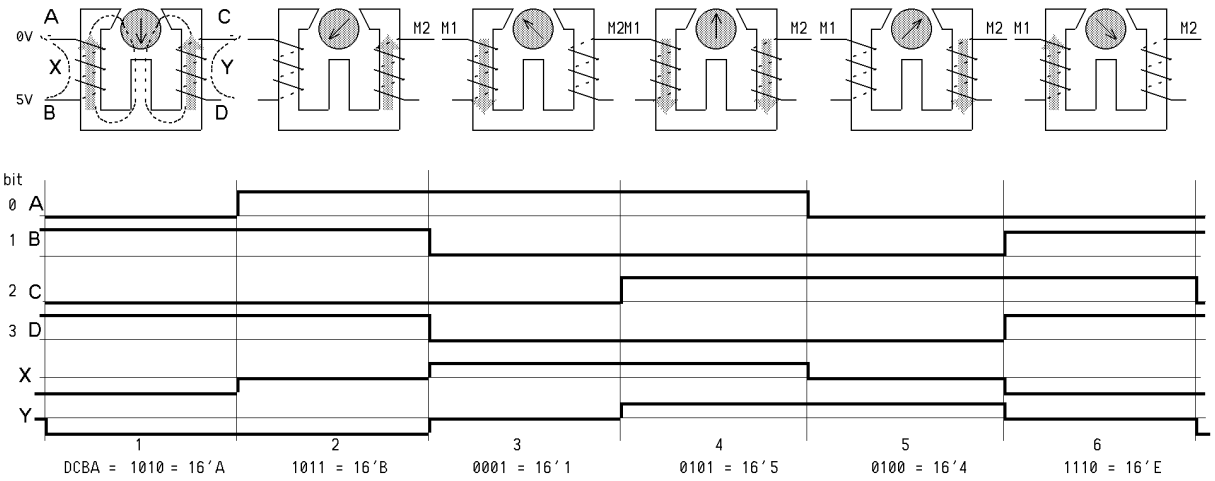
Commande de moteur pas-à-pas

Les moteurs pas à pas industriels ont 4 phases et on les programmes le plus fréquemment par demi-pas. Des circuits intégrés existent avec les amplis de puissance et il y a avantage à les utiliser.
Les moteur d'instrumentation, que l'on trouve entre autre dans les tableaux de bord des voitures n'ont pas besoin d'amplis et permettent plusieurs applications robotiques et domotique.

Les moteurs et horloges Wellgain choisis par Didel existent dans différentes configurations. Ils ont deux bobines et peuvent se commander directement avec 3 ou 4 sorties de microcontrôleur. La séquence à 6 phases est donnée ci-dessous. Des engrenages divisent par 180 et le couple de sortie est de quelques grammes par cm..



Cette séquence se décrit dans un tableau de 6 valeur

```
byte step[6]={0x0A,0x0B,0x01,0x05,0x04,0x0E};
```

que l'on parcourt avec un compteur par 6, avec un `if` pour recommencer

```
i++; if(i==6) i=0;  // compte 0 1 2 3 4 5 0 1 2 ....
```

La valeur qui correspond au pointeur `i` dans le tableau est récupérée par `step[i]`

Le programme qui fait tourner le moteur pas-à-pas à vitesse constante est donc très simple

```
char i=0;
byte etat[6]={0x0A,0x0B,0x01,0x05,0x04,0x0E};
void loop()
{
    i++; if(i==6) i=0;   // compte 0 1 2 3 4 5 0 1 2 ....
    PORTC = etat[i];
    delayMicroseconds(1000); // min ~500 à 5V
}
```

Pour changer de sens, il faut compter dans l'autre sens

```
i--; if(i<0) i=5;  // décompte 0 5 4 3 2 1 0 5
```

On remarque que `i` doit être déclaré comme nombre signé (`int` ou `char`) pour que l'instruction `i<0` soit exécutée correctement. Il y a certainement d'autres solutions.

Post et pré-modification

Le moteur pas-à-pas aide à comprendre une notion importante. Un pointeur (i dans les exemples ci-dessus) mémorise quelle est la position du moteurs. Mais est-ce la position avant l'instruction, ou la position après ? Dans le premier cas, on doit calculer la position suivante avant d'assigner la position. Dans le second, on assigne et on décide quelle sera la position suivante.
Le moteur est dans une position donnée et on ne sait pas si l'ordre suivant le fera tourner dans un sens ou dans l'autre. Donc il faut calculer la position suivante et l'appliquer.

Deux moteurs

Ce qui nous intéresse pour un robot (Wellbot2) ou pour la machine à dessiner KiDelta, c'est de pouvoir piloter deux moteurs indépendamments.

La table peut être complétée et les deux moteurs sur les PORTC et D vont tourner à la même vitesse. Arduino n'a pas un port 8 bits complet que l'on peur utiliser pour 2 moteurs. Il faut répartir les bits sur 2 ports, ce que l'on fait en appelant une fonction (www.didel.com/kidules/PortK.pdf)

```
byte steps[6]={0xAA,0xBB,0x11,0x55,0x44,0xEE};
```

Si, comme pour un robot, les deux moteurs sont symétriques, ils doivent tourner en sens inverse pour que le robot aille droit. On permute alors les 6 états d'un moteur dans la table.

```
byte steps[6]={0xEA,0x4B,0x51,0x15,0xB4,0xAE};
```

Il est alors facile d'écrire un programme qui fait tourner les moteurs et tester la vitesse de démarrage et la vitesse max.

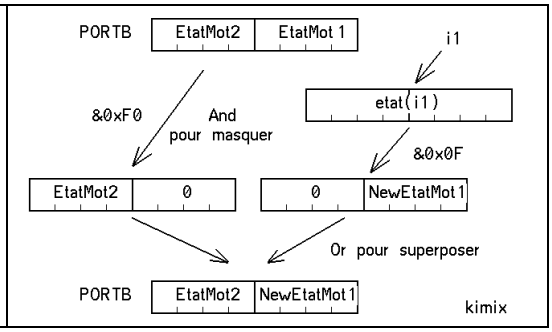
Durées de pas différentes

Le problème se complique si les deux moteurs n'ont pas la même vitesse, ils ont des délais différents et le programme doit gérer deux tâches indépendantes. Cela peut se faire avec deux timers interne au processeur, mais les timers sont rares et on préfère les utiliser pour autre chose. On va faire ces timers par logiciels, c'est à dire programmer deux compteurs qui augmentent en même temps, mais on les teste et remet à zéro séparément pour définit la durée des pas.

```
delayMicroseconds(100);
del1++; del2++ ;
if (dureePas1 = del1)  { del1=0 ; fairePasMoteur1 ;  }
if (dureePas2 = del2)  { del2=0 ; fairePasMoteur2 ;  }
```

Les procédures `fairePas` ont leur pointeurs `i1 i2` et on voit que l'on décide toutes les 100 us seulement si un pas est exécuté.

Le problème maintenant est, lorsque l'on fait un pas sur un moteur, de ne pas modifier l'autre moteur.
La figure ci-contre montre comment modifier le moteur 1 seulement.
C'est facile de compléter ce programme pour qu'il tienne compte de deux variables donnant le sens de rotation des moteurs, mais continuons différemment.

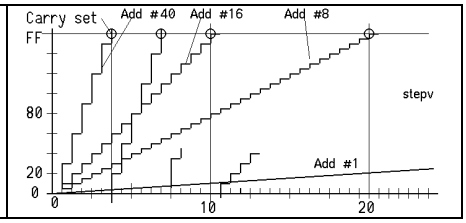


Vitesse variable

Comme pour un moteur continu, raisonner avec une vitesse de rotation plutôt qu'avec une durée de pas et plus naturelle, et elle permet de représenter naturellement les vitesses positives et négatives.

Comprenons le principe avec des vitesses positives seulement.

Tous les 100 microsecondes par exemple, ajoutons une valeur de 0 à 255 appelée vitesse à une variable 8 bits. L'unité arithmétique 8 bit du microcontrôleur déborde si le résultat est égal ou supérieur à 256 (la variable garde ce qui a débordé), et le Carry s'active. Si la Vitesse est 0, la variable ne bouge pas, il n'y a jamais de Carry.



Si la Vitesse vaut 1, la variable déborde tous les 256 cycles (25.6 ms).
Si la Vitesse vaut 32, la variable déborde tous les 8 cycles (0.8 us).
Si la Vitesse vaut 255. la variable déborde chaque fois (sauf une fois tous les 256).

Le moteur à vide décroche à 0.5 – 1 ms, donc la vitesse maximale est de 25 environ,
Les pas ne sont pas réguliers, l'espace entre pas peut varier de 100 microsecondes, ce ''jitter'' n'a pas d'influence. Avec le moteur VID29 il faut 6 pas par tour et la réduction est de 180. La durée maximale d'un tour est donc de 100x255x6x180=27540000 microsecondes, soit un tour en 27 secondes. La vitesse maximale est dans les 2 tours à la seconde).

Testons avec un moteur unidirectionnel. Le cœur de la boucle du programme `KiVit1.pde` est

```
delayMicroseconds(100);
total += vitesse ;  // déclaré int 16 bits
if (total > 0xFF) {  // dépasse 8 bits, on fait un pas
      total &= 0xFF ;
      i++; if(i==6) i=0;  // compte 0 1 2 3 4 5 0 1 2 ....
      PORTB = etat[i];
```

Modifions ce programme pour avoir des vitesses positives de 0, 1, 2… 25 et des vitesses négatives de 0, –1, -2, … -25. La variable vitesse est de type char, le test de signe se fait avec l'instruction `vitesse & 0x80`. Il faut encore convertir la valeur –1 en valeur absolue 1, ce qui se fait avec un signe - , mais il faut dans cette opération convertir le type char de la vitesse signée dans le type byte de la valeur ajoutée.
mot pas à pas commandé en vitesse signée

```
// Vitesse max 25 --> 2t/s
byte u8 etat[6]={0xEA,0x4B,0x51,0x15,0xB4,0xAE};
char vitesse = -25; // s8 maintenant -25 .. +25
byte absvit ;
int total ; byte i=0 ;
void loop()
{
    delayMicroseconds(100);
    if (vitesse&0x80)==0) { // positif
          byte absvit = vitesse ;
          total += absvit ;
          if (total > 0xFF) {  // dépasse 8 bits, on fait un pas
            total &= 0xFF ;
                i++; if(i==6) i=0;  // compte 0 1 2 ..
                PORTB = etat[i];
          }
    }
    else { // négatif
          u8 absvit = -vitesse ;
          total += absvit ;
          if (total > 0xFF) {  // dépasse 8 bits, on fait un pas
            total &= 0xFF ;
                i--; if(i<0) i=5;  // décompte 0 5 4 ..
                PORTB = etat[i];
          }
    }
}
```

Pour commander deux moteurs, la solution simple définit deux vitesses de consigne `Vit1,Vit2`, qui sont des nombres 8 bits signés, dans l'intervalle –32 à +32. Ces valeurs sont utilisées dans la routine d'interruption, qui peut saturer les valeurs données à 32, et remplacer les transitions brusques de vitesse par des rampes d'accélération freinage.

L'objectif est souvent d'exécuter un certain nombre de pas, donc faut connaître le nombre de pas effectués, qui correspond à une distance parcourue dans le cas d'un robot à 2 roues. La différence du nombre de pas correspond alors à un angle de rotation.
On définit deux variables signées 16 bits `NbPas1,NbPas2` qui permettent de totaliser ~32000/6/180 = 29 tours. Si le robot a une roue de 30mm de diamètre, cela correspond à un déplacement de 3 mètres.
L'utilisateur souhaite souvent d'aller à une position donnée le plus rapidement possible. Il faut accélérer et freiner à temps, pour éviter de perdre des pas et de glisser.
La fonction `AllerA (Position,Vitesse)` n'est pas simple.

Gestion par interruption
La routine d'interruption gère les moteurs pas-à-pas toutes les 100 microsecondes. Les vitesses sont ajoutées pour définir s'il faut faire un pas et le nombre de pas mise à jour.

```
// byte vit1, byte vit2 sont déclarés dans le pp – non testé
void DoPas (vit1, vit2) {  ! non testé
      if (bitTest(vitesse,7)==0) { // positif
            byte absvit1 = vit1 ;
            int  total += absvit1 ;
            if (total>255) {  // on fait un pas
                  i++; if(i==6) i=0;  // compte 0 1 2 ..
                  PORTB = etat[i];
            }
      }
      else { // négatif
            byte absvit = -vitesse ;
            total += absvit ;
            if (total>255) {  // on fait un pas
```

```
                        i--; if(i<0) i=5;  // décompte 0 5 4 ..
                        PORTB = etat[i];
                    }
                }
            }
```

Vitesse différente pour chaque moteur

La procédure delay() est monotâche ; elle bloque le processeur pendant le délai. Les deux moteurs ont besoin de délais différents. Pour bien faire, il faut une gestion par interruptions, ce que la librairie fournira (celles de l'Arduino ne conviennent pas, elles sont faites pour des moteurs 4 phases).
Une solution à notre portée est de prendre une décision d'évolution très fréquente (100 microsecondes par exemple) et utiliser deux compteurs de période pour décider chaque fois si le délai de chaque moteur est passé, et faire un pas si oui.
La période est calculée à partir de la vitesse par division et la vitesse peut être exprimée en mm/s. c'est le grand avantage du C sur l'assembleur, on mutiplie et divise facilement, sans se préoccuper de la place mémoire et du temps de calcul (mais attention aux 100 microsecondes !).

Donc pour chaque moteur, toutes les 100 microsecondes, on décompte un compteur, et lorsqu'il arrive à zéro, on fait un pas et réinitialise le décompteur Avec 100 microsecondes de cycle et 1200 microsecondes de pas minimum, le compteur est réinitialisé à la valeur 12 à la vitesse maximale. A quel moment faut-il convertir la vitesse en durée) ? Chaque 100 microsecondes ? Le plus facile, mais on ne va pas changer la vitesse si souvent. Toutes les 20 ms ? En même temps que l'on prend des décisions sur les capteurs. Logique, mais c'est une 3^{e} tâche qui va prendre plus que 100 microsecondes et va fausser l'espace entre les pas.

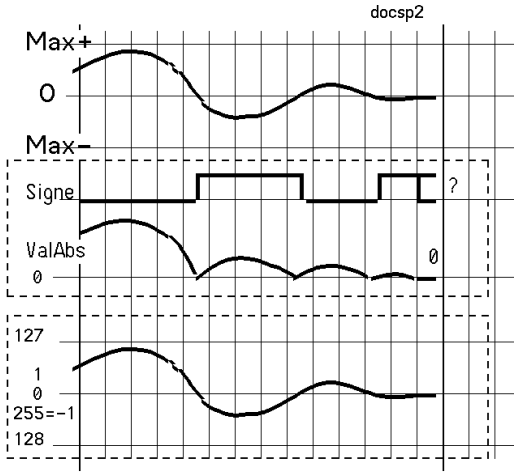
Programmons la solution basée sur des périodes. Définissons les variables PeriodeDroite PeriodeGauche et les décompteurs associés PD et PG.
PeriodeDroite et PeriodeGauche ont des valeurs signées, PD et PG sont initialisés avec leur valeur absolue (voir le programme complet).

```
void  loop ()  {
      delayMicrosecondes (50) ;
      PD--; if (PD==0)  {
            PD = PeriodeDroite ;
            if (PeriodeDroite >= 0)   { id++; if(id==6) id=0; }
            else   { id=id-1;  if(id==-1) id=5; }
            }
      PG--; if (PG==0)  {
            PG = PeriodeGauche ;
            if (PeriodeGauche >= 0)   { id++; if(id==6) id=0; }
            else   { id=id-1;  if(id==-1) id=5; }
            }
      PORTB = AvDroite[id] | AvGauche[ig] ;
}
```

Il n'a a plus qu'à déclarer le début correctement pour tester le programme `Wd2Pas2.pde` qui animait les WellBots2 Pinguino. www.didel.com/Wbot2Pub.pdf

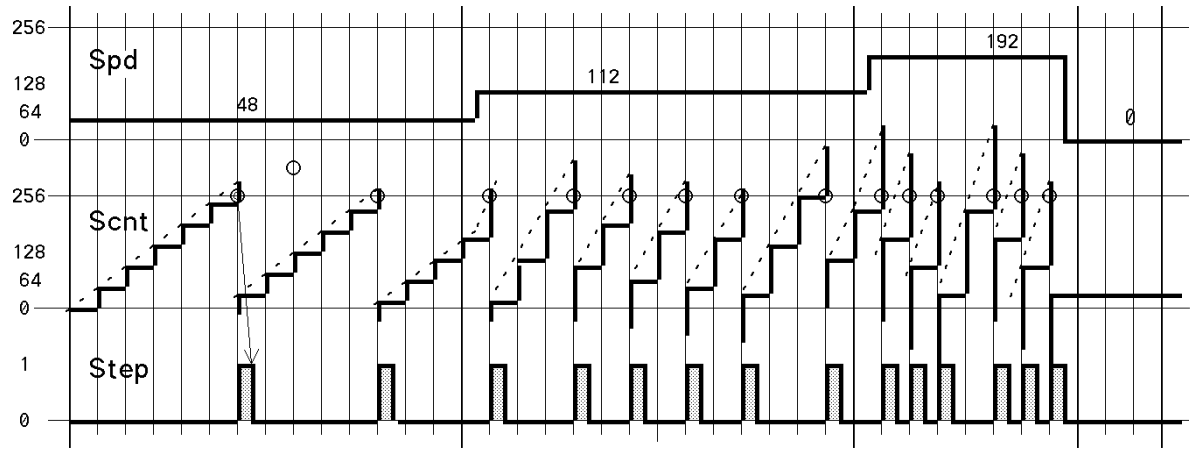
Vitesse

La vitesse de rotation de chaque moteur peut être positive, négative ou nulle. Ne parlons pas encore de la vitesse du robot, qui dépend des deux vitesses des moteurs.
Nous pensons vitesse absolue et sens, ce qui peut se coder avec deux variables, ou en assembleur sur un variable 8 bits avec un bit de signe (bit 7) et une valeur absolue 7 bits. La représentation logique est toutefois en complément à 2, avec une valeur absolue max de la vitess égale à 128 (mots de 8bits), ce qui est bien assez précis.



Il faut maintenant exécuter des pas avec une vitesse signée différente pour chaque moteur.

La période des pas est inverse de la vitesse. En C, on peut diviser facilement (et oublier le temps de calcul et la taille mémoire) et calculer le délai selon la vitesse. Ce n'est ensuite pas évident puisque deux délais se superposent. Une solution est d'utiliser deux timers réinitialisés selon la vitesse lorqu'ils ont appelé l'avance d'un pas par interruption. Avec une seule interruption de période x courte, on peut arrondir la durée entre pas à un multiple de x et décider à chaque interruption s'il faut faire un pas sur un moteur, sur l'autre ou sur les deux. On peut encore s'inspirer de la solution préférée en assembleur.
En assembleur, on utilise une base de temps qui correspond à la période minimum, et une vitesse max 8 bits qui va conduire à une vitesse linéaire de 0 à 255. La vitesse signée Speed est convertie en valeur absolue et doublée (Spd) . Cette valeur est ajoutée à la variable Scnt dont le débordement est signalé par le Carry, qui fait passer au pas suivant. La décision est prise toutes les 1000 microsecondes puisque la période minimale des pas est de 1 ms.



On remarque toutefois que pour les vitesses élevées, il manque parfois une impulsion pour arriver à une moyenne juste. l'inertie du moteur essaye de compenser, mais le rendement est mauvais. Il faut limiter la vitesse à 64 pour que l'impulsion suivante ait au maximum 25% d'errreur, donc échantillonner 4 fois plus rapidement (250 microsecondes).
Dans la pratique, on échantillonne à une période de 100 à 200 us fixée par d'autres contraintes (Petra, Pfm, servos) et on documente la vitesse maximale des moteurs pour en tenir compte dans l'application. On trouve donc les instructions suivantes dans la routine d'interruption.

SpeedDroite est la vitesse signée, de –MaxVit à +MaxVit (Maxvit est < 127, dans la pratique < 60 environ). SPD est la valeur absolue , éventuellement divisée par 2.
Attention à l'extension du signe (donc en C au type de donnée). Il faut prendre le complément avant de décaler.

jdn 110522/131202