PFM et vitesse progressive par interruption

Note: www.didel.com/diduino/PfmParInter.pdf est une première version de ce document.

Le but ici est de tester des routines PFM en commençant par les plus simples et de finir par bien comprendre comment gérer les vitesses des deux moteurs d'un robot, par interruption. Un préalable est une bonne compréhension du C, les exercices inspirés du Coursera "Comprendre les microprocesseurs", documentés pour l'environnement de test que nous allons utiliser, sont recommandés pour se "faire la main" www.didel.com/coursera/LC.pdf.

1. Environnement d'apprentissage

Les programmes agiront sur les pins Arduino 4,5,6,7 utilisées traditionnellement pour commander des moteurs, à cause du analogWrite possible sur les pins 5 et 6. Les pins 2 et 3 sont câblés des poussoirs, ou aux moustaches de votre robot. Le fichier Def2w2m.h est facile à modifier pour un autre câblage, il n'y a aucune contrainte de no de pins.

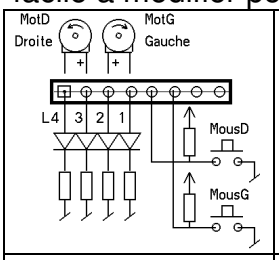
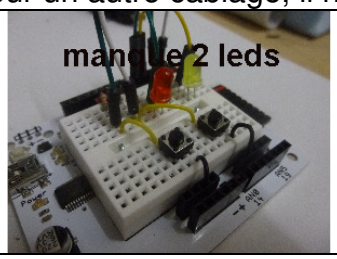
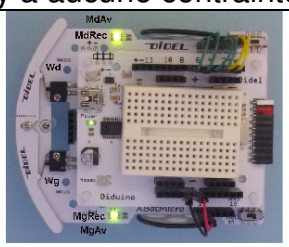
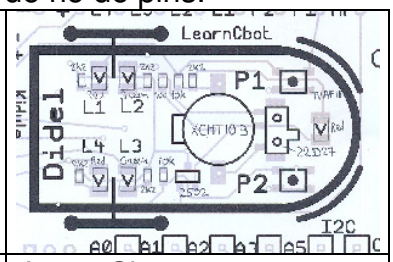
			

Schéma	Diduino	XBot - Diduibot	LearnCbot

Les sorties moteur sont actives à l'état "1". Par contre, les deux poussoirs "moustaches" P1 et P2 qui détectent les **Obs**tacles sont actifs à zéro.
Les noms des moteurs sont G et D, ils **Av**ancent et **Rec**ules
Les bits moteurs ont 4 états 00 stop 10 avance 01 recule 11 pas utilisé
Les commandes sont AvD; RecG; StopG; AvD; RecD; StopD; StopGD; (ne pas utiliser Stop;)
Pour les poussoirs (moustaches) les états sont Pous1On, Pous2On avec des synonymes liées à l'application des moustaches détectant la présence d'obstacles: ObsG et ObsD
Notre but est d'agir sur 2 variables pfmD pfmG dont la valeur entre –MaxPfm et +MaxPfm définit la vitesse de chaque moteur.
On garde le nom vitesse pour une consigne de vitesse. La vitesse réelle doit être mesurée et suppose l'utilisation d'in encodeur.

2. Test en tout ou rien

La première chose à tester, c'est que l'environnement fonctionne. Il faut déclarer les pins, ce que nous ferons avec le fichier XbotDef.h très proche du LcDef.h du MOOC EPFL.

Quelques rappels:
Les constantes et fonctions commencent par une majuscule. Les variables par une minuscule.
Dans un #define la dernière instruction n'a pas de ; L'appel se fait sans ().
Une action définie dans un #define se termine par un ; Par exemple AvG;
Un état défini dans un #define est assigné ou testé. Par exemple if (ObsG) {…

Test initial: clignoter les leds moteurs

```
// Test0.ino
#include "Xbot.h"
void setup () {
  SetupXbot ();
}
```

```
// XbotDef.h
#include <Arduino.h>  //<avrio.h>
#define bLed 5  // Led13 aide debug
#define LedOn  bitSet (PORTB,bLed)
#define LedOff bitClear (PORTB,bLed)
#define LedToggle PORTB ^= (1<<bLed)

#define bRecG 4
#define bAvG 5
#define bAvD 6
#define bRecD 7
#define AvG   bitSet  (PORTD,bAvG); bitClear(PORTD,bRecG)
#define RecG  bitClear(PORTD,bAvG); bitSet  (PORTD,bRecG)
```

```
void loop () {
  LedToggle;
  AvD; delay (300);
  StopD; delay (300);
  AvG; delay (300);
  StopG; delay (300);
  delay (500);
}
```

```
#define StopG bitClear(PORTD,bAvG); bitClear(PORTD,bRecG)
#define AvD   bitSet  (PORTD,bAvD); bitClear(PORTD,bRecD)
#define RecD  bitClear(PORTD,bAvD); bitSet  (PORTD,bRecD)
#define StopD bitClear(PORTD,bAvD); bitClear(PORTD,bRecD)

#define bMousD 3 // PORTD,3)
#define bMousG 2
#define ObsG  !(PIND&(1<<bMousG))  //!digitalRead(MousG)
#define ObsD  !(PIND&(1<<bMousD))  //!digitalRead(MousD)

void SetupXbot() {  // initialisation 23 in 4567 out
  DDRD |=  0b11110000 ;
  DDRD &=  0b11110011 ;
  DDRB = 1<<bLed ; // Led13
}
```

Tous les programmes se trouvent sous www.didel.com/diduino/PfmPratique.zip

3. PFM simple

Testons avec la Led verte sur la pin 6 la fonction PFM bien expliquées en www.didel.com/kidules/PwmPfm.pdf

Le nombre de valeurs possibles pour la variable pfm est fixé par MaxPfm, égal à 20 dans nos exemples. Une valeur supérieure conduit à des saccades visibles à faibles valeur de pfm. Une valeur plus faible réduit le choix des vitesses; dans la pratique, un robot avec 8 ou 10 vitesses est bien assez précis, surtout s'il n'a pas d'odométrie, donc pas de possibilité d'appliquer des algorithmes de réglage de vitesse mathématiques.

Avec notre premier programme pour vérifier la fonction DoPfm, qui commande un moteur avec des vitesses positives entre 0 et MaxPfm, le pfm est constant et il faut recompiler pour tester une nouvelle valeur. Ceci permet de vérifier des valeurs extrêmes ou anormales, par exemple pfm=0, 19, 20, 21, 32. Une valeur négative est acceptée, le processeur voit le complément à 256 qui est positif.

```
// Test1.ino
#include "Xbot.h"
void setup(){
   SetupXbot();
}
#define MaxPfm 20
void DoPfm (byte pf) {
   volatile byte pfmCnt;
   if (pf > MaxPfm) pf = MaxPfm; // sature
   if ((pfmCnt += pf) > MaxPfm) {
      pfmCnt -= MaxPfm;
      AvG;
   }
   else StopG;
}

byte pfm = 7;  // 0 .. MaxPfm
void loop () {
  delay (2);  // 500 pour voir les impulsions
  DoPfm (pfm);
}
```

Il faut agiter la Led pour voir les impulsions générées par le PFM.
La fonction DoPfm est indépendante du nombre de niveaux, qui est défini avant l'appel.
byte pf donne un nom local au paramètre de la fonction
byte pfmCnt est une variable locale. Le prefixe volatile indique au compilateur de ne pas la modifier en dehors (risque avec des interruptions)

On peut maintenant compliquer le programme pour varier la vitesse automatiquement ou selon des actions sur les moustaches/poussoirs. A noter que l'on ne peux pas utiliser de `delay(), for(,,) ou while()` pour faires des retards. Les instructions du programme doivent être parcourues en moins de 2ms.
La fonction DoPfm est mise dans un #include " Pfm.h" pour alléger le programme.

// **Test2.ino** // Toutes les 100 x 2ms on modifie le PFM

```
// Accélère et recommence
#include "Xbot.h"
#define MaxPfm 20
#include "Pfm.h"
void setup() {
  SetupXbot();
}
byte pfm,cnt;
void loop () {
   delay (2);
   DoPfm (pfm);
   if (cnt++ > 100) {
     cnt=0;
     pfm++;
     if (pfm> MaxPfm) pfm=0;
   }
}
```

```
//Pfm.h  Pfm 0..MaxPfm
volatile byte pfmCnt;
void DoPfm (volatile byte pf) {
   if (pf > MaxPfm) pf = MaxPfm;
   if ((pfmCnt += pf) > MaxPfm) {
      pfmCnt -= MaxPfm;
      AvD;
   }
   else {StopD;}
}
```

Dans tout programme, il faut bien voir les blocs (ici en couleur) et respecter les alignements.

.

// **Test3.ino** // varie selon le poussoirs/moustaches

```
#include "Xbot.h"
#define MaxPfm 20
#include "Pfm.h"
void setup () {
   SetupXbot();
}
byte pfm = MaxPfm/2;  // 0 .. 20
byte cnt=0;
void loop () {
  delay (2);
  DoPfm (pfm);
  if (cnt++ > 100)  {
     cnt=0;
     if (ObsD) {
        if (pfm < MaxPfm) { pfm++;}
     }
     if (ObsG) {
        if (pfm > 0) pfm--; }
     }
  }
}
```

Ce programme mérite plusieurs commentaires.
On est facilement tenté d'écrire

```
if (ObsD) {
   pfm++; if (pfm >20) {pfm=20;}
}
if (ObsG) {
   pfm--; if (pfm <0) {pfm=0;
}
```

Ceci peut poser problème parce que pfm a été déclaré byte, positif. <0 est compris >127.

Si on écrit

```
if (ObsD) {
     if (pfm++ >20) {pfm--;}
}
if (ObsG) {
     if (pfm-- <1) {pfm++;
}
```

il faut bien savoir que le `if` teste la valeur `pfm` avant modification. Faut-il écrire `++pfm` et `--pfm` ?

4. Interruption Timer2

Gérer le PFM par interruption est facile. Le Timer2 est programmé pour une interruption toutes les 60 ou 100 microsecondes (pour servir d'autres tâches rapides). Toutes les 20 interruptions on appelle la fonction DoPfm.
Dans le programme principal, on peut réutiliser les delay() et autres actions bloquantes.

// **Test4.ino**

```
// Test4.ino
#include "Xbot.h"
#define MaxPfm 20
#include "Pfm.h"

volatile byte cnt;  // max 255
ISR (TIMER2_OVF_vect) {
  TCNT2 = 256-200;  //  100 us
  if (cnt++ > 20) { // 2ms
    cnt = 0;
    DoPfm ();
  }
}
void SetupTimer2() {
  TCCR2A = 0; //default
  TCCR2B = 0b00000010; // div 8, 2MHz
  TIMSK2 = 0b00000001;
}
void setup () {
  SetupXbot();
  SetupTimer2():
}
byte pfm;
void loop () {
  pfm++; if (pfm>MaxPfm) { pfm=0; }
  delay (200);
}
```

Si les interruptions n'ont pas été vues, il faut dire que ISR () est la fonction appelée quand le compteur du timer 8 bits passe à zéro. Sa valeur augmente toutes les 0.5 µs (défini par le registre TCCR2B). Il faut réarmer avec le complément à 200, puisque le compteur TCNT2 augmente.

La durée de l'interruption est de ~4 µs.

Un changement très important est que la valeur pfm n'est plus un paramètre, mais une variable globale déclarée avant d'être appelée. La fonction DoPfm() n'a pas de paramètre. Elle lit directement la variable globale pfm qui est modifiée par le programme principal. Dans le programme principal, on modifie cette variable, et à la prochaine interruption, sa valeur est prise en compte.

La prochaine étape est de cacher les instructions Timer dans un #include

// **Test5.ino**

```
#include "Xbot.h"
#define MaxPfm 20
volatile byte pfm;  // variable globale
#include "Pfm.h"
#include "Timer2.h"
void setup()  {
 SetupXbot();
 SetupTimer2();
}
```

```
void loop () {
   pfm++; if (pfm>MaxPfm) pfm=0;
   delay (200);
}
```

5. Vitesse négative

La vitesse doit être déclarée de type char (uint8_t), 8 bit signé, pouvant représenter des nombres de –128 à +127. Nos vitesses seront entre –MaxPfm et +MaxPfm.
Il faut traiter séparément les vitesses positives et négatives.
Nous ne travaillons plus que par interruption avec une vitesse globales pfm.

```
void DoPFM () {
  volatile byte pfmCnt;
  if (pfm > MaxPfm) pfm=MaxPfm; // saturer
  if (pfm < -MaxPfm) pfm= -MaxPfm;
  // on sépare selon le signe
  if (pfm >=0) {
     if ((pfmCnt += pfm) > MaxPfm) {
        pfmCnt -= MaxPfm;
        AvG;
     }
     else { StopG; }
  }
  else {
     pfm = -pfm;  // ou pfm= abs(pfm)
     if ((pfmCnt += pfm) > MaxPfm) {
        pfmCnt -= MaxPfm;
        RecG;
     }
     else { StopG; }
   }
}  // end DoPFM
```

Le **Test6.ino** est similaire au Test5, à part que l'on va de la vitesse négative max à la vitesse positive max. Le **Test7.ino** fait des allers-retours et ressemble plus aux programmes futurs que l'on devra écrire.

6. Pfm sur les 2 moteurs

Il suffit de gérer les deux moteurs dans une fonction DoPfmDG .
Le programme **Test8.ino** qui utilise cette nouvelle fonction copie le Test6, mais varie les 2 moteurs.

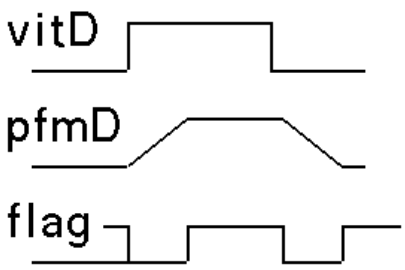
```
void loop () {
  pfmD++; pfmG=pfmD;
  if (pfmD>MaxPfm) { pfmD=-MaxPfm; }
  delay (200);
}
```

7 Résumé

On a maintenant vu comment gérer un moteur par interruption. Les variables du programme principal se déclarent avant le programme. Une variable doit être connue par la fonction ou programme qui suit, une fonction doit être connue par une fonction qui suit. C permet des déclarations pour éviter cette containte.
Un fois les modules en place et testés par un programme simple, la valeur de MaxPfm choisie, on ne voit plus que pfmD et pfmG à gérer correctement.
Pour créer un nouveau programme avec 2 moteurs bidirectionnels, le plus simple est de partir de Test8.ino, renommer ce qu'il faut et modifier le programme principal.

8. Arrêt départ en souplesse

Le programme doit surveiller la valeur du PFM et transformer les changements brusques en rampes. On compare périodiquement la valeur actuelle avec la valeur demandée et on incrémente ou décrémente la vitesse pour de rapprocher de la valeur voulue. Si la période est de 20ms, et on demande de passer de vitesse nulle à vitesse max (20), cela prendra 0.4s.



C'est une tâche supplémentaire à effectuer par interruption, toutes les 20ms. La vitesse voulue par l'utilisateur est interceptée par cette tâche, qui fabrique la valeur pfm pour les moteurs. Pour bien comprendre ce qui se passe,
1) le programme modifie vitD par exemples
2) quand l'interruption à 20ms est appelée, le nouveau pfmD est calculé
3) quand les prochaines interruptions à 2ms sont appelées, le moteur a la nouvelle consigne.
.La fonction SetVit() est exécutée toutes les 20ms et active un flag pour dire que la vitesse voulue est atteinte.

```
void SetVit (char vd,char vg) {
   if (vd>pfmD) {pfmD++; vstable=0;}
   if (vd<pfmD) {pfmD--; vstable=0;}
```

```
    if (vg>pfmG) {pfmG++; vstable=0;}
    if (vg<pfmG) {pfmG--; vstable=0;}
    if ((vitD==pfmD)&(vitG==pfmG)) {vstable=1;}
  }
```

La fonction ISR(Timer2) a été modifiée en ajoutant une tâche qui s'effectue toutes les 20ms.

```
volatile byte cnt1;
volatile byte cnt2;
ISR (TIMER2_OVF_vect)
{
  TCNT2 = 65;  // 256-200+corr <-- 200x8x62.5ns = 100 us
  // on ajoutera ici des timers, des servos, etc
  if (cnt1++ > 20)  // toutes les 2ms
  {
    cnt1 = 0;
    DoPfm (pfmD,pfmG);  //durée 10us
  }
  if (cnt2++ > 20*10)  // toutes les 20ms
  {
    cnt2 = 0;
    SetPfmDG ();  //durée 10us
  }
}
```

Les noms des fichiers insérés ont été changés puisque maintenant on a une base flexible pour beaucoup d'applications. Le programme **Test9.ino** fait osciller le robot.

```
// Test9.ino
#include "LcDef.h"
#include "LCbot.h"
#define MaxPfm 20
char  pfmD=2,pfmG=2;
char  vitD=pfmD,vitG=pfmD;
byte vstable;
#include "LibMoteurs.h"
#include "ISRTimer.h"
void setup()  {
  LcSetup ();
  SetupTimer2();
}
```

```
#define Vit 20;
void loop () {
   vitD = 0; vitG=0;
   delay (1000);    // attente
   vitD = Vit; vitG=-Vit;
   delay (1000);
   vitD = 0; vitG=0;
   delay (1000);
   vitD = -Vit; vitG=Vit;
   delay (1000);
}
```

Les délais dans ce programme doivent permettre au robot ait le temps de finir son mouvement (ou pour vérifier ce qui se passe si on ne lui laisse pas le temps)
En modifiant dans la routine `ISR (TIMER2..` l'instruction
`if (cnt2++ > 20*10)  // toutes les 20ms` → `mettre 20*50 pour 0.1s`
on voit que l'évolution de vitesse est beaucoup plus progressive.

Le test 10 optimise le mouvement en tenant compte du flag `vstable`.
La fonction `NextVit()` a été ajoutée dans le programme principal. Cette fonction est bloquante puisqu'elle attend que la vitesse soit atteint. Le Test 10 peut ainsi enchaîner les mouvements le plus rapidement possible.

```
void NextVit (char vd,char vg) {
  vitD = vd; vitG = vg;
  delay (10);
  while (!vstable) {}
  delayMicroseconds (10);
}
```

La boucle du programme **Test10.ino** s'écrit très clairement

```
#define Vit 10
void loop () {
  // tourne gauche-droite
  NextVit (0,0);
  NextVit (Vit,-Vit);  //tourne g
  NextVit (-Vit,Vit);  // tourne d
  NextVit (0,0);  // retour au
centre
  delay (500);
```

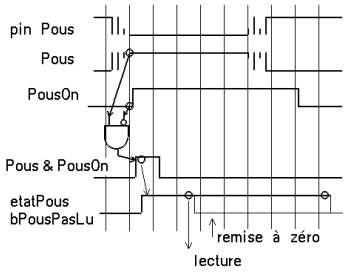
```
// avance-recule
  NextVit (0,0);
  NextVit (Vit,Vit);  //avance
  NextVit (-Vit,-Vit);  // recule
  NextVit (0,0);  // retour
  delay (1000);  // attente
}
```

En résumé, on peut définir la vitesse de trois manières selon l'application:
- En assignant **`pfmD pfmG`**, avec réaction immédiate des moteurs
- Avec la fonction **`SetVit () ;`** qui n'est pas bloquante
- Avec la fonction **`NextVit () ;`** qui est bloquante.

9 Table de vitesses

Pour programmer un ballet, il faut créer une table qui contient les vitesse successives.

10. Moustaches par interruptions



Repartons de zéro. Les moustaches ont des rebonds de contact parfois assez important. Echantillonnons par interruption toutes les 50 millisecondes et activons pour chaque moustache deux flags dans une variable EtatMous qui est une copie idéale des 2 moustaches.
Pour chaque moustache, l'interruption va donc mettre à jour deux bits. Un bit nommé ObOn est l'état actuel de la moustache, que l'on utilise rarement. Le 2e bit ObEdge mémorise que l'on a pressé, c'est un flag (drapeau) que l'on doit remettre a zéro quand on a pris note de l'informations.
Il faut déclarer trois variables et écrire les instructions suivantes, appelées toutes les 20 à 50 ms.

```
byte nowObs, prevObs, flagObs;
#define bObOn 1
#define bObEdge 0
nowObs = Obs;
if (nowObs) {bitSet flagObs,bObOn);}  else {bitClear flagObs,bObOn);}
if (prevObs && !nowObs) { bitSet (flagObs,bObEdge); }
prevObsD = nowObsD;
```

Si on veut savoir si un obstacle est touché, moustache appuyée, on écrira
if (`flagObs & 1<<bObOn) { . . .  }`
Si on veut savoir si un obstacle a été touché, on faisait autre chose et la moustache n'est peut-être plus appuyée maintenant, il faut tester le flag, et en général le quittancer
if (`flagObs & 1<<bObEdge){ bitClear flagObs,bObEdge); . . .  }`
? version simplifiée 1 flag
?? fonctions TestObs → mets à jour
Humm, pas le temps de retravailler cette fin. Faites signes et je m'y remet. jdn

Le routine d'interruption et le programme **TestM1.ino** qui clignote à chaque action sont alors

```
volatile int cnt;
ISR (TIMER2_OVF_vect)
{
  TCNT2 = 68;  // 256-200+corr <--  100 us
  if (c500++ > 500) { // toutes les 50ms
     c500 = 0;
    TestObs();
  }
}
```

```
. . . définitions, set-up, fonctions

void loop () {
  if (flagObsD) {
    flagObsD=0;
    LedToggle;
  }
}
```

Pour les deux moustaches, on peut hésiter é mettre 1 nouveau bit dans etatMous et compléter GetEtatMous. Si etatMous=0, il n'y a pas d'action à entreprendre.
Le **TestM2.ino** fait un clignotement différent à droite et à gauche.
Le programme **TestM3.ino** utilise les fonctions du Test10.ino et évite les obstacles.

```
void loop () {
  // obstacle à droite?
  if (etatMous & (1<<bMousDOnPasLu)) {
     bitClear (etatMous,bMousDOnPasLu);
     NextVit (-VitEvite,-VitEvite);  //recule
     delay (500);
     NextVit (VitEvite,-VitEvite);  // tourne g
     delay (200);
     NextVit (VitAv,VitAv);  // retour au centre
  }
```

```
 // obstacle à gauche?
  if (etatMous & (1<<bMousGOnPasLu)) {
     bitClear (etatMous,bMousGOnPasLu);
     NextVit (-VitEvite,-VitEvite);  //recule
     delay (500);
     NextVit (-VitEvite,VitEvite);  // tourne d
     delay (200);
     NextVit (VitAv,VitAv);  // retour au centre
  }
}
```

```
// TestM1.ino
#include "LcDef.h"
#include "LCbot.h"

byte etatMous;
#define bMousDOnPasLu 0
#define bMousDOnNow 1
#include "LibMoustaches.h"

#include "ISRTimer0.h"

void setup()  {
 LcSetup ();
 Timer2Setup();
}

void loop () {
  if (etatMous & (1<<bMousDOnPasLu)) {
     bitClear (etatMous,bMousDOnPasLu);
     LedToggle;
   }
}
```

```
//LibMoustaches.h
static byte mousDPrev;
void GetEtatMous ()  {
   if (MousDOn ^ mousDPrev) {
      bitSet (etatMous, bMousDOnNow) ;
      bitSet (etatMous, bMousDOnPasLu) ;
   } else {
     bitClear (etatMous, bMousDOnNow) ;
   }
   mousDPrev = MousDOn ;
}
```

```
// TestM2.ino
#include "LcDef.h"
#include "LCbot.h"

byte etatMous;
#define bMousDOnPasLu 0
#define bMousDOnNow 1
#define bMousGOnPasLu 2
#define bMousGOnNow 3
#include "LibMoustaches.h"

#include "ISRTimer0.h"

void setup()  {
 LcSetup ();
 Timer2Setup();
}

void loop () {
  delay (1000);
  if (etatMous & (1<<bMousDOnPasLu)) {
     bitClear (etatMous,bMousDOnPasLu);
     LedToggle;
   }
   if (etatMous & (1<<bMousGOnPasLu)) {
     bitClear (etatMous,bMousGOnPasLu);
     LedToggle;
     delay (300);
     LedToggle;
   }
}
```

```
//ISRTimer0.h
volatile int cnt;
ISR (TIMER2_OVF_vect)
{
  TCNT2 = 68;  // 256-200+corr <--
200x8x62.5ns = 100 us
  // on ajoutera ici des timers, des
servos, etc
  if (cnt++ > 500)  // toutes les 2ms
  {
    cnt = 0;
    GetEtatMous ();  //durée 10us
    PORTC=etatMous;
  }
}

void Timer0Setup() {
  cli();
  TCCR2A = 0; //default
  TCCR2B = 0b00000010;
  TIMSK2 = 0b00000001;
  sei();
}
```

```
// TestM3.ino Evite obstacles
#include "LcDef.h"
#include "LCbot.h"

#define MaxPfm 20
char  pfmD=2,pfmG=2;
char  vitD=pfmD,vitG=pfmD;
volatile byte vstable;
#define VitAv 10   // avance
#define VitEvite 5  // recule et tourne

#include "LibMoteurs.h"

byte etatMous;
#define bMousDOnPasLu 0
#define bMousDOnNow 1
#define bMousGOnPasLu 2
#define bMousGOnNow 3
#include "LibMoustaches.h"
```

```
//ISRTimer.h  avec VitD G
volatile byte cnt1;
volatile byte cnt2;
ISR (TIMER2_OVF_vect)
{
  TCNT2 = 68;  // 256-200+corr <--
200x8x62.5ns = 100 us
  // on ajoutera ici des timers, des
servos, etc

  if (cnt1++ > 20)  // toutes les 2ms
  {
    cnt1 = 0;
    if ((vitD==pfmD)&(vitG==pfmG))
{vstable=1;}
    else {vstable=0;}
    DoPfm (pfmD,pfmG);  //durée 10us
  }
  if (cnt2++ > 20*10)  // toutes les
20ms
```

```
#include "ISRTimer.h"
void setup()  {
 LcSetup ();
 Timer2Setup();
 PORTC=0;
 pfmD=0; pfmG=0;
 NextVit (VitAv,VitAv);
}

void loop () {
  // obstacle à droite?
  if (etatMous & (1<<bMousDOnPasLu)) {
     bitClear (etatMous,bMousDOnPasLu);
     NextVit (-VitEvite,-VitEvite);
//recule
     delay (500);
     NextVit (VitEvite,-VitEvite);  //
tourne g
     delay (200);
     NextVit (VitAv,VitAv);  // retour
au centre
  }
  // obstacle à gauche?
  if (etatMous & (1<<bMousGOnPasLu)) {
     bitClear (etatMous,bMousGOnPasLu);
     NextVit (-VitEvite,-VitEvite);
//recule
     delay (500);
     NextVit (-VitEvite,VitEvite);  //
tourne d
     delay (200);
     NextVit (VitAv,VitAv);  // retour
au centre
  }
}
```

```
  {
    cnt2 = 0;
    SetVit (vitD,vitG);
    GetEtatMous ();   // 50ms
nécessaires?
  }
}

void Timer2Setup() {
  cli();
  TCCR2A = 0; //default
  TCCR2B = 0b00000010;
  TIMSK2 = 0b00000001;
  sei();
}
```

```
//LibMoustaches.h
static byte mousDPrev;
void GetEtatMous ()  {
   byte temp = PIND & (1<<bWd|1<<bWg);
   if (temp ^ mousDPrev) {
     if (!(temp & 1<<bWd)) {
        bitSet (etatMous, bMousDOnNow) ;
        bitSet (etatMous, bMousDOnPasLu)
;
     } else {
     bitClear (etatMous, bMousDOnNow) ;
     }
     if (!(temp & 1<<bWg)) {
        bitSet (etatMous, bMousGOnNow) ;
        bitSet (etatMous, bMousGOnPasLu)
;
     } else {
     bitClear (etatMous, bMousGOnNow) ;
     }
   }
   mousDPrev = temp ;
}
```