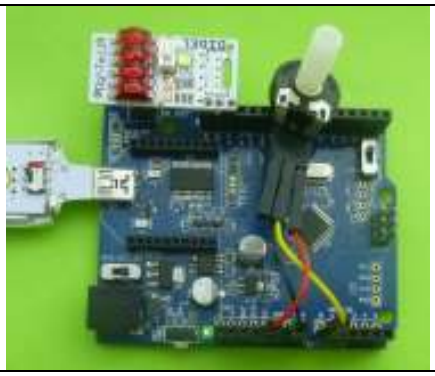
http://www.didel.com/ info@didel.com

www.didel.com/xbot/DemoTell.pdf

Demo Tell
Exemples avec la librairie LibX

Le module Tell est très utile pour aider à la mise au point des applications en visualisant des variables ou la valeur de capteurs pendant que le robot est autonome. Il suffit de 100 bytes en mémoire et la pénalité de temps d'exécution est minime. La première démo affiche la valeur d'un potentiomètre ou capteur analogique branché sur A0.	

Les démos de ce document montrent l'intérêt de LibX et du matériel associé et ne présupposent pas un apprentissage préalable.
Le module Tell utilise les ressources de LibX www.didel.com/diduino/DgTell.pdf
La librairie LibX, facilement portable sur tout microcontrôleur programmable en C est documentée sous www.didel.com/xbot/LibX.pdf
Les programme sont exécutés dans l'environnement Arduino, mais les fonctions Arduino, peu efficaces, ne sont pas utilisées, sauf le delay(), pratique pour la mise au point dans le programme principal puisqu'il est bloquant.
Les programmes sont disponibles sous www.didel.com/xbot/DemoTell.zip .

Note. Avoir pratiqué Arduino via des exemple n'est pas suffisant pour une bonne compréhension de ce document, puisqu'Arduino s'efforce de cacher les aspects temps réel et architecture interne du microcontroleur. Le cours www.didel.com/coursera/LC.pdf est recommandé. Le MOOC EPFL https://www.coursera.org/course/microcontroleurs recommence le 17 octobre 2015.

Principe de la librairie X

Tous les microcontrôleurs ont au moins un compteur interne, appelé timer. Il décompte et quand il arrive à zéro, il active une bascule (appelée flag ou sémaphore). Ce flag coupe alors la parole au programme principal (interrupt) et dans le cas de LibX, ce sont plusieurs activités qui sont surveillées, avec des astuces de programmations qui font que chaque tâche ne dure que quelques microsecondes à chaque interruption. Avec LibX, le programme de l'utilisateur est interrompu pour 10-20 microsecondes toutes les 60 microsecondes. C'est comme si le processeur tournait à 10 MHz au lieu de 16Mhz.
Le travail fait par interruption est de lire des capteurs, ici un potentiomètre, et afficher la valeur sur le module Tell. Le module Oled par contre est bloquant; il a son propre processeur qui n'a pas pu être adapté. Il utilise une librairie Arduino très lourde.
LibX travaille avec des variables globales, rarement avec des fonctions.

Le programme principal

Par interruption, la pin A0 est lue et copiée dans la variable ana0. Par interruption, la variable `highLow` est affichée en hexa ou décimal sur Tell. Afficher la courbe des mesures sur l'écran Oled se fait dans le programme principal: x avance circulairement. Pour y, il faut diviser par 16 (max 1023 --> 63 ou 3FF --> 3F).	``` byte x; void loop() { delay (100); highLow= ana0; } ```

Setup

Il faut naturellement des définitions et un setup avant le programmme principal.

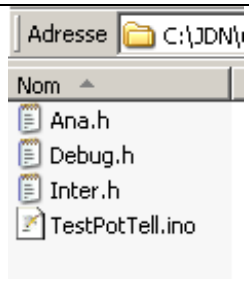
Les librairies Debug, Ana et Inter sont un peu expliquées plus loin.

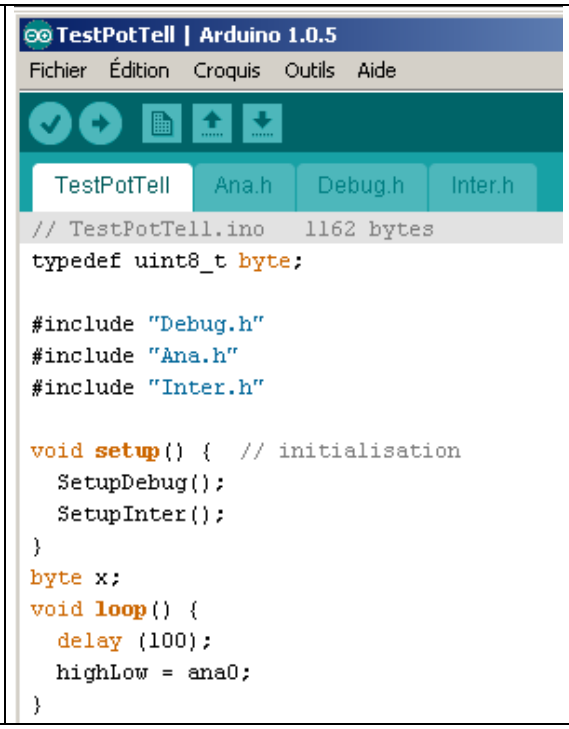
```
// TestPotTell.ino  7898 bytes  1168 sans Oled
typedef uint8_t byte;  //pour le compilateur

#include "Debug.h"
#include "Ana.h"
#include "Inter.h"

void setup() {  // initialisation

  SetupDebug();
  SetupInter();
}
```

Structure du croquis et conseil

Sur le disque, le croquis Arduino contient 3 fichiers.

Sur l'écran, les fichiers sont listés en haut et il n'est pas nécessaire d'être dans le .ino pour compiler. Mais attention, une modification sur l'écran n'est pas sauvée automatiquement sur disque. Sauvez bien vos programme avant de les modifier: sauvez, créez le nouveau nom, modifiez.





Librairie LibX Inter.h

Le fonctionnement et les interruptions du timer sont expliquées sous www.didel.com/coursera/LC5.pdf et www.didel.com/coursera/LC7.pdf
Le setup initialise les registres dans le mode le plus simple qui fixe la vitesse de "rotation" du compteur et autorise les interruptions.
Le flag appelle via une fonction AVR les instructions à exécuter: réinitialiser le compteur pour que la prochaine interruption se fasse dans 60us, appeler les routines DoAna() et SendS1() (affichage).
Ensuite, le compteur cnt2 fait que des tâches supplémentaires ne s'exécutent que toutes les 20ms, et pas toutes en même temps: l'une après l'autre toutes les 2.5 ms.
On reviendra sur la structure switch, essentielle dans les projets.

```
// Inter.h tous les 60us
volatile byte cnt1,cnt2;
volatile byte e6ms;
ISR (TIMER2_OVF_vect) {
 TCNT2 = 141;  //  60 us
 DoAna();
 SendS1();
 if (cnt2++ >= 40) { cnt2=0;
          // toutes les 40x60= =2.4ms   min 10-15  pour Uson
    switch (e6ms) {
       case 0:  // 1 fois sur 8 ~20ms
         flagS1=1;   // lance le transfert
         e6ms=1;   break;
       case 1:
         flagAna=1;  // lance la conversion
         e6ms=2;   break;
     ....  une partie manque
    }  // end switch
 }  // end if cnt2
} // end ISR

void SetupInter() {  // initialisation
  TCCR2A = 0; //default
  TCCR2B = 0b00000010;  // clk/8
  TIMSK2 = 0b00000001;  // TOIE2
  sei();
}
```

Lecture analogique

Arduino a la fonction analog.Read(A0) qui doit être appelée à chaque lecture.
Au lieu d'écrire `ana0 = analogRead(A0);` on lit la variable `ana0`, mise à jour toutes les 20ms.
Une différence importante: analogRead(A0); rend un mot de 10 bits, la variable ana0 est 8 bits, une précision bien suffisante pour des capteurs peu précis dans un environnement aléatoire.

```
//Ana.h    Lecture directe A0 sur AVR 328
volatile byte ana0;

void SetupAna() {
     ADMUX  = 0x60+0 ;  // 8-bit mode, + channel
     ADCSRA = (1<<ADEN) + 5;  // prescale 32
     DDRC &= 0b11111110;
}
void DoAna () {
    ADCSRA |= (1<<ADSC) ;   // start conversion
    while (ADCSRA & (1<<ADSC)) ;  // wait finished
    ana0 = ADCH ;
}
```

Affichage Tell

Un fil peut suffire pour transmettre des mots binaires en jouant sur la largeur des impulsions. Le format S1 utilisé par le module Tell est expliqué sous www.didel.com/diduino/DgTell.pdf
Le transfert, haché menu par les interruptions, prend 3ms et est répété toutes les 20ms.
La pin Arduino 13 est utilisée pour le transfert; sa position à côté d'un Gnd et d'un + sur Diduino facilite le câblage.

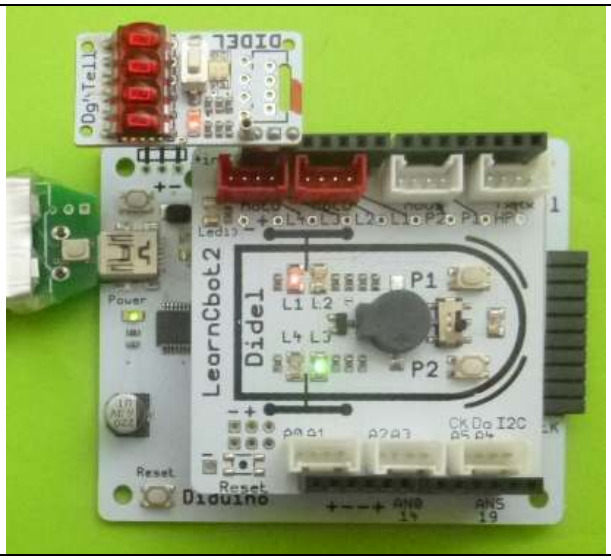
La pin 12 peut créer un signal Tst, utile par exemple pour synchroniser un oscilloscope ou analyseur logique.

La fonction SendS1() sera analysée dans une section prévue sur les machines d'état.

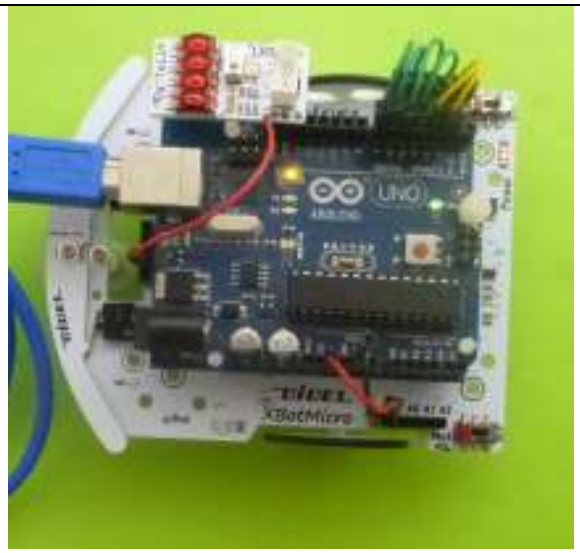
```
// Debug.h
volatile int highLow;
byte flagS1;  // envoi toutes les 20ms

#define bS1 5 // PORTB pin 13
#define S1On bitSet (PORTB,bS1)
#define S1Off bitClear (PORTB,bS1)
#define S1dirOut  bitSet   (DDRB,bS1)
#define S1In  (PORTB & (1<<bS1)) //avec Tell ine pas lire la pin

#define TstOn bitSet (PORTB,4)  // dépannage oscillo
#define TstOff bitClear (PORTB,4)
#define TstToggle PORTB ^= 1<<4
#define TstdirOut bitSet(DDRB,4)

void SetupDebug() {
  S1dirOut;  TstdirOut;
}

#define Nbits 16
enum {AtS1,DebS1,StartS1,BitS1,ContS1,StopS1} eS1 =
AtS1;
  volatile unsigned int hL;
  volatile byte cc, etatLed;
void SendS1 () {   // appel toutes les 58 us
  switch (eS1) {
. . . . .
```

Commande de moteur

Vous avez un LearnCbot ou un Xbot? C'est l'occasion de voir comment commander la vitesse avec un pot ou un joystick.



La vitesse des moteurs est visible sur des leds d'intensité variable. Vert on avance, rouge on recule.



L'interrupteur en bas à droite est Off pendant le test des programmes. Des leds bicolores donnent l'état d'avance des moteurs

Pour une vitesse variable, utilisons le PFM, qui permet d'aller très lentement et se programme sur n'importe quelles pins http://www.didel.com/kidules/PwmPfm.pdf
La librairie LibX Pfm.h traite 2 variables signées pfmL (left) et pfmR (right).
Les vitesses vont de –99 à +99; les valeur en dessous (max –128) et en dessus (max 127) sont saturées, c'est à dire donnent la vitesse maximale, respectivement minimale..
L'utilisation de nombres négatifs est parfois troublante. Il faut bien déclarer les types et faire attention dans les comparaisons

Programme principal joystick --> moteurs

Après avoir déclaré et initialisé les librairies, le programme est très simple. Le pot au milieu donne une valeur proche de 0, correspondant au moteur arrêté. On peut ajouter une zone morte autour du centre (bon exercice de comparaisons signées). Pour afficher les valeurs du pfm on forme le mot de 16 bits en décalant d'un byte.

```
// TestPotMotTell.ino  1444 octets
. . .  includes, setup
// Le joystick commande la vitesse
void loop() {
  pfmL = ana0-128;
  pfmR = ana1-128;
  highLow = (pfmL<<8) + pfmR;
}
```

En assignant chaque potentiomètre à un moteur, il faut bouger le manche en diagonale pour avancer ou reculer. A vous de mélanger et inverser les canaux si nécessaire.

Fichier de définitions XbotDef.h
Ce fichier décrit les bits et les actions relatives des moteurs, moustaches, etc.
En passant par LibX, l'utilisateur n'a pas besoin de travailler avec ces définitions matérielles, sauf quelques-unes:
`LedOn; LedOff; LedToggle;` agissent sur la pin 13, utilisée par Tell. Mais comme Tell agit sur cette ligne une parite du temps seulement, si on allume ou éteint la led par programmation on voit bien la différence d'intensité.
`ObsL ObsR` sont utilisés pour tester l'état des moustaches: `if (ObsL) { }`.
Les Leds associées aux moustaches peuvent être activées par soft en passant par les macros `LedWLOn; LedWLOff; LedWROn; LedWROff;`
Ceci même si le contact est fermé, l'action est de type "collecteur ouvert".
La contrainte est de se mettre Off (leds éteintes) si on veut pouvoir lire les moustaches.

Librairie LibX Ana4.h
Pour lire les 4 canaux, il suffit de déclarer les variables et agrandir la machine d'état qui, sans prendre plus de temps au processeur, va lire un canal après l'autre dans un cycle de 20ms.

Librairie LibX Pfm.h

La fonction pfm est un peu plus compliquée que dans http://www.didel.com/kidules/PwmPfm.pdf parce q'il faut séparer les vitesses positives et négatives, et qu'il y a 2 canaux.

```
// Pfm.h
volatile int8_t pfmL, pfmR;  // -99 .. +99
#define MaxPfm 100
volatile byte pfmCntL;
volatile byte pfmCntR;
void DoPfm () {
  char tempL,tempR;
  if (pfmR > MaxPfm) pfmR=MaxPfm; // saturer
  if (pfmR < -MaxPfm) pfmR= -MaxPfm;

  if (pfmR >=0) {
     if ((pfmCntR += pfmR) > MaxPfm) {
        pfmCntR -= MaxPfm;   ForwRight; }
     else {  StopRight; }
   }
  if (pfmR <0) {
     tempR= -pfmR;  // ou pfR= ABS(pfR)
     if ((pfmCntR += tempR) > MaxPfm) {
        pfmCntR -= MaxPfm;  BackRight; }
     else {  StopRight; }
   }
  if (pfmL > MaxPfm) pfmL=MaxPfm; // saturer
  if (pfmL < . . .
```

Gestion du PFM dans LibX Inter.h

La fonction doPfm() doit être appelée toutes les 2ms environ, pour lui permettre de démarrer à la vitesse la plus lente. Avec 1% de PFM, il y a une impulsion d'avance toutes les 200ms, visible sur les leds moteur. Les engrenages atténuent ces à-coups. Essayez 1% de PWM. Jusqu'à 10-20%, le moteur ne bouge pas..

```
// Inter.h tous les 60us
volatile byte cnt1,cnt2;
volatile byte e6ms;
ISR (TIMER2_OVF_vect) {
 TCNT2 = 141;  //  58 us calibre
 DoAna4();
 SendS1();
 if (cnt1++ > 42) { cnt1=0;  // toutes les
42x58= 2.5ms
   DoPfm();
 }
 if (cnt2++ >= 40) { cnt2=0;
```

Moustaches

Les moustaches se gèrent dans le programme principal. La solution simple pour éviter un obstacle est de recule et tourner, comme dans le programme ci-contre.
Il faut ajuster les vitesses et le délai expérimentalement.
On complètera la boucle si on veut simultanément suivre une trajectoire ou naviguer selon des capteurs.

```
// TestEviteObstacleTell.ino
void loop() {
  pfmL=50; pfmR=50;
  highLow = (pfmL<<8) + pfmR;
  if (ObsL) {  //on recule en tournant
    pfmL=-20; pfmR=-50;
    highLow = (pfmL<<8) + pfmR;
    delay (500);
  }
  if (ObsR) {
    pfmL=-50; pfmR=-20;
    highLow = (pfmL<<8) + pfmR;
    delay (500);
  }
}
```

Obstacles par interruption

Cela peut être gênant de surveiller les obstacles dans toutes les parties d'un programme. Ajoutons un flag déclaré `byte StopIfObstacle;` Si ce flag est à 1, l'interruption surveille les moustache et, s'il y a obstacle, coupe les moteurs et met ce flag à zéro. Le programme lit si le flag est à zéro et intervient. On surveille un flag plutôt que les moustaches ? Quel est le gain? La surveillance du flag ne doit pas se faire toutes les 20ms pour éviter que le robot force sur l'obstacle.
Les instructions à exécuter dans une tâche d'interruption toutes les 20ms sont:

```
if (StopIfObstacle) {
  if (ObsL | ObsR) { pfmL=0; pfmR=0; StopIfObstacle =0;
}
```

On peut encore exécuter l'évitement d'obstacle dans la routine d'interruption, sans que le programme principal s'en rende compte (mais on va naturellement prévoir un compteur d'obstacle que le programme principal peut lire).
A noter que dans la routine d'interruption on ne peut pas avoir une instruction bloquante comme delay(). Il faut écrire une machine d'état avec un compteur d'interruption qui va compter les durées; on verra comment plus loin.

Obstacle devant

La probabilité de détecter une simultanéité est très faible si on échantillonne toutes les 20ms; même en face, une moustache va réagir plus vite que l'autre.
Il faut donc tester l'une ou l'autre des moustaches, et si pressé, couper le moteur, attendre que le robot soit arrêté et retester si c'est les deux ou l'une ou l'autre moustaches (dans cet ordre) qui fait contact.

Gestion du temps

1) `byte` **`hours minutes seconds cents`** `(hundredth)`
Arduino documente millis() et micros(), que nous remplaçons par une horloge de 4 registres qui comptent en binaire, et que l'on peut afficher en décimal sur Tell.

cents	0 à 0x63	= 99 centièmes
seconds	0 à 0x3B	= 59 secondes
minutes	0 à 0x3B	= 59 minutes (min est un mot réservé)
hours	0 à 0xFF	=255 heures = 10 jours 15 heures

2) byte **time20ms time1s**

Ces deux compteur comptent et saturent à la valeur max 0xFF.
Pour mesurer un temps, on met à zéro au début de la mesure et on lit à la fin de la mesure. C'est comme un chronomètre !
Time20ms mesure jusqu'à 5 secondes. Time1s jusqu'à 4 minutes.

3) byte **stopOnTime20ms**

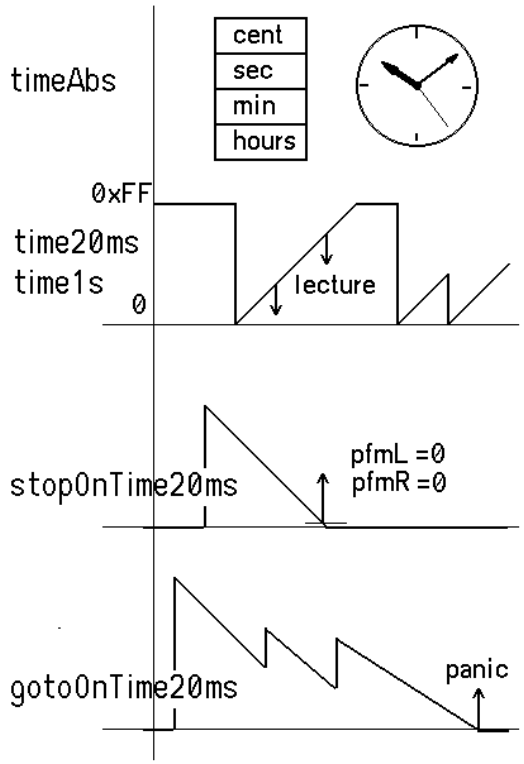
On a vu le StopIfObstacle;. Le compteur StopOnTime20ms est initialisé à une valeur entre 1 et 255 et le moteur s'arrêtera entre 20ms et 4,4 minutes.

4) byte **sotoOnTime20ms**

Le compteur GotoOnTime20ms est initialisé à une valeur entre 1 et 255 et décompte toutes les 20ms. En arrivant à zéro, l'exécution continue à l'adresse indiquée dans la routine d'interruption. Cette solution, simple et efficace, suppose que la suite du programme est la récupération d'un comportement non prévu et sera suivi d'une réinitialisation complète.

Note: Le goto est mal vu, mais pas interdit. C'est comme traverser la route en dehors des passages piéton. C'est déconseillé aux personnes qui n'ont pas assez de jugement !

La routine d'interruption exécute une sous-tâche toutes les 20ms et une tâche toutes les 1ms.




```
if (cnt1++ > 23) { cnt1=0;  // toutes les 23x60= 1.4ms
  DoPfm();
}
if (cnt2++ >= 41) { cnt2=0;   // toutes les2.4ms x 8
 switch (ems) {
    case 0:
      flagS1=1;   // lance SendS1
      ems=1;   break;
    case 1:
      ems=2;   break;
    case 2:
      ems=3;   break;
    case 3:
      if (time20ms) {time20ms--;}
      ems=4;   break;
    case 4:
      if (time1s) {time1s--;}
      ems=5;   break;
    case 5:
      if(stopOnTime20ms==1) {pfmL=0; pfmR=0;}
      if(stopOnTime20ms) {stopOnTime20ms--;}
      ems=6;   break;
    case 6:
       if(!gotoOnTime20ms--) { }  // NA goto Bug;}
        ems=7;   break;
      case 7:
        ems=0;   break;
 } // end switch
}  // end if cnt2 6ms
if (cnt3++ > 170) { cnt3=0; // 10ms for the clock
  centS++; if (cents==100) { cents=0;
    seconds++; if (seconds==60) { seconds=0;
      minutes++; if (minutes==60) { minutes=0;
         hours++;
      }
    }
  }
}  // end cnt3
```

On remarque l'astuce de couper les moteurs lorsque le compteur vaut 1, pour éviter de le bloquer à zéro dans l'état de repos.

Machines d'état

Le principe des machines d'état est expliqué dans www.didel.com/coursera/LC5.pdf et

www.didel.com/coursera/LC7.pdf, ainsi que dans tous les documents C avec des exemples ridicules. Pour nos applications plutôt robotique, il faut considérer 2 cas:
1) L'application passe par une suite d'états bloquants
2) Chaque état ne peut pas durer plus qu'un cerain temps, pour permettre l'exécution en parallèle d'autres tâches.
Le premier cas est utilisé dans le programme principal:
- etat 1 le robot avance, s'il y a obstacle à droite: état 2. S'il y a obstacle à gauche: état 3.
- etat 2 on fait ceci tant que ... ou pendant un certain temps
Le second cas exige un balayage régulier et un temps max dans chaque état. Il n'y a pas de délai, on compte des cycles pour atteindre la bonne durée.

Le zip contient 2 programmes TestEviteObstacleCaseTell.ino et ..Tell2 non commenté ici.

Contourner une île

On a vu comment longer un mur. S'il y a un angle rentrant, il faut améliorer l'algorithme pour éviter de très nombreux aller et retour avant que le virage à 90 degrés soit effectué. Si l'angle est sortant, il faut tenir compte du temps sans obstacles. La variable `StopOnTime20ms` est utilisée dans l'état "avance" . Chaque obstacle à droite réinitialise le décompteur. Si `StopOnTime20ms==0` on passe dans des états de recherche d'un obstacle.

```
// FollowWallOutside.h - Right handed  ignore left obscacles
byte case=0;
#define ASO 50 // attente sans obstacle
while (1)  {
  low = case;   // pour Tell, afficher l'état en cours
  switch (case) {
    case 0:
      pfmL = 70; pfmR = 65;  // turn a little bit right to follow wall
      StopOnTime20ms = ASO ;
        case=1; break;
    case 1:
      if (StopOnTime20ms==0) {case=20;}
      if (ObsL) {case= 10;}
      break;
    case 10: // obstacle left
      pfmL = -70; pfmR = -60;  delay (400);
      case=0;  break; // start again
    case 20: // timeout  Tourne à droite de ~30 degrés
      pfmL = 60; pfmR = 0;  delay (400);
      case=0;  break;
  } // end switch
}   // end while follow
```

Tell est très utile pour visualiser les états; ici, la machine est trop simple.

Machine d'état synchrone pour capteur ultrason

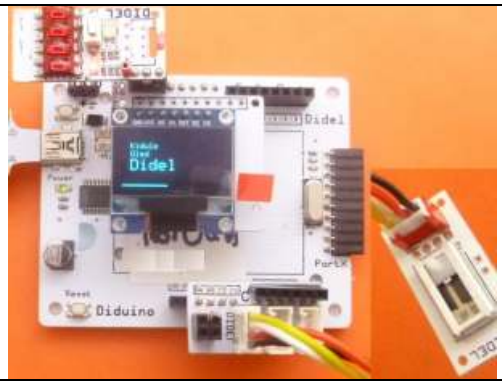
L'explication détaillée se trouve dans www.didel.com/xbot/DistSonar.pdf
L'interruption, toutes les 60 µx, appelle la fonction DoUson pour quelques instructions:
état 0: On attent que flagUson soit actif. Si oui, état =1;
état 1: On active le signal Trig. état = 2;
état 2: On désactive Trig (impulsion d> 10ms) état = 3;
état 3: On attend que le signal Echo s'active état = 4;
état 4: On compte les cycles. Si Echo==0, état = 5; Si nombre de cycle trop grand, etat = 6;
état 5: On copie le compteur de cycles dans distUson
état 6: Il n'y a pas de capteur câblé, on continue ? On signale comment ?

jdn 151005

Doc séparée prévue pour la suite
Capteurs du Xbot: Suivi de lignes et de lumières, Ultrasons. PSD. Infrarouge
Capteurs I2C.
Créer une librairie LibX.

Le module Tell est très utile pour aider à la mise au point des applications en visualisant des variables ou la valeur de capteurs pendant que le robot est autonome. Il suffit de 100 bytes en mémoire et la pénalité de temps d'exécution est minime.
Le module Oled est par contre lourd et lent, mais joli pour les démonstrations.

La première démo affiche la valeur d'un potentiomètre ou capteur analogique branché sur A0.



Le module Oled est documenté sous www.didel.com/xbot/Oled.pdf

Librairie Oled

La librairie Oled doit être installée selon les directives d'Adafruit. Une vestion "light" expurgée des fonctions qui ne nous sont pas utiles ici (dessiner un cercle, un triangle, etc) est accessible via www.didel.com/xbot/Oled.pdf .
Cette documentation montre comment afficher des nombres en hexa et décimal, mais le display demande de rafraichir tout l'écran après écriture dans la mémoire cache, et cela prend un temps parfois excessif.
Comme le Oled utilise la pin 13 de l'affichage, on remarque un cli() sei() qui coupe et réactive les interruptions pendant les mises à jour de l'affichahe.

```
/ Oled.h  1306 avec adaptateur Didel
#define CK   12
#define MOSI 11
#define RZ   10
#define DC    9
#define CS    8
Adafruit_SSD1306 display(MOSI, CK, DC, RZ, CS);

void Welcome () {//Didel.h
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(10,10);
  display.println("Kidule");
  display.setCursor(10,20);
  display.println("Oled");
  display.setCursor(10,30);
  display.setTextSize(2);
  display.print("Didel");

  display.display();
}

 void PutDot (byte pp,byte dd) {
  byte y = 62-dd;
  if (y>62) y=62;
  byte x= pp;
  display.drawPixel(x, y, WHITE);
  display.drawPixel(x, y+1, WHITE);
}

void PutLines (byte np,byte mm) {
     byte y = 62-mm;
     byte x = np ;
   display.drawLine(0, y, 127, y, WHITE);
   display.drawLine(x, 0, x, 63, WHITE);
   display.display();
}
void Update () {
   cli(); display.display(); sei();
}
```

jdn 151018