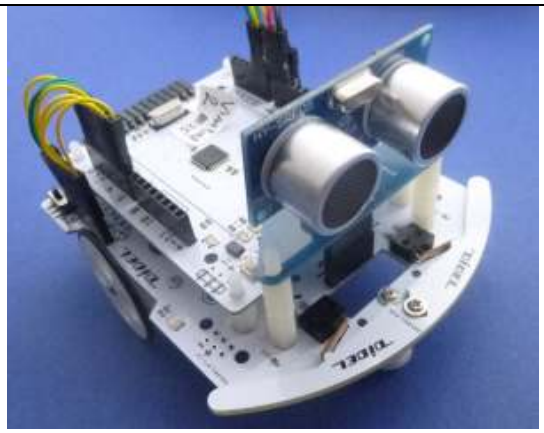
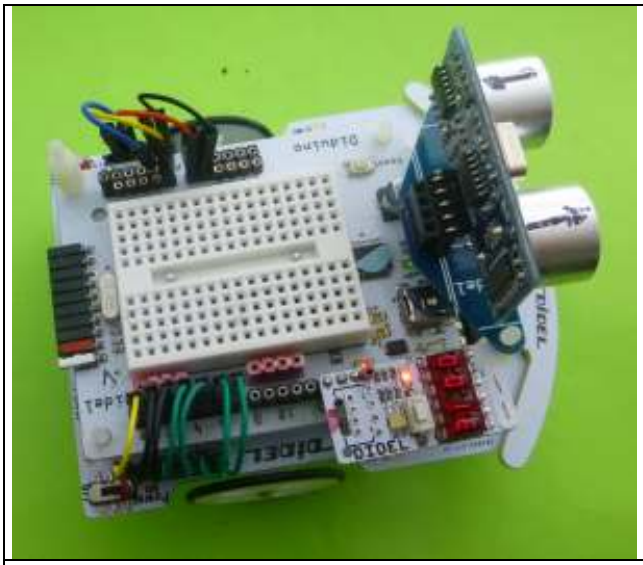
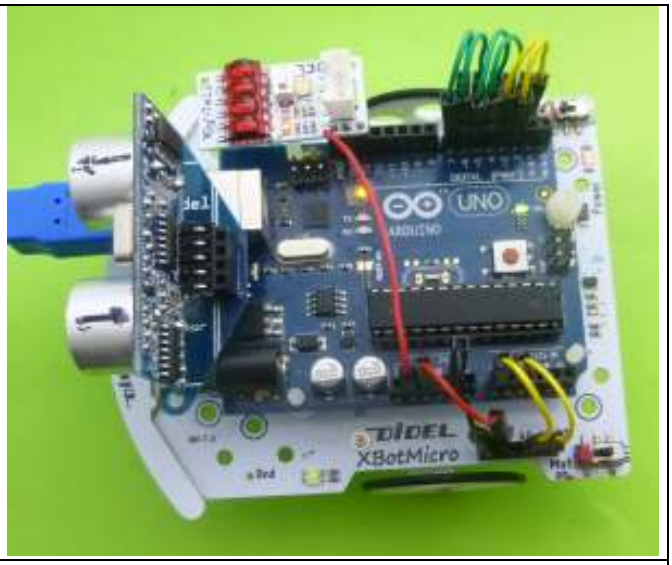
Expériences avec le capteur ultrason

Utilise le Xbot Arduino avec SR05, Tell et la libraire LibX

Le capteur de distance à ultrason SR05 est décrit en détail sous www.didel.com/xbot/DistSonar.pdf Cette documentation montre comment utiliser le SR05 "à la Arduino", dans un mode bloquant qui limite les applications. Elle explique aussi pour les avancés le mécanisme de la librairie Uson.h. Le but ici est de démontrer la facilité de mise en oeuvre de la librairie LibX et l'intérêt de l'afficheur Tell pour avoir la distance affichée en temps réel sur le robot en déplacement. Avec des adaptation mineures, le test peut se faire sur la plupart des plateformes robot pilotées par Arduino.	

Câblage sur la plateforme XbotMicro
Le câblage d'une carte Diduino ou Arduino-Uno est expliqué sous www.didel.com/xbot/XBotMicro.pdf
Le logiciel LibX n'est pas compatible avec la carte Leonardo, mais pourrait être adapté.
La documentation du module Tell se trouve sous www.didel.com/diduino/DgTell.pdf
La docuimenta de câblage du module Uson se trouve sous www.didel.com/xbot/UsonMontage.pdf

	
Le Diduino alimente le module Tell sans fil supplémentaire.	L'alimentation de Tell peut se faire en 3.3V, pour éviter un dédoublement de l'alimentation 5V

Une vidéo est prévue pour expliquer le câblage
:

Les librairies utilisées sont
XbotDef.h Défini le câblage des moteurs et moustaches
Pfm.h Converti les variable `char pfmL et pfmR`, valant de –100 à +100
Uson.h Lit le capteur et met à jour `byte uson` qui est la distance mesurée, de 3 à 255 cm. Une distance 0 indique que le capteur n'est pas connecté.
	`typedef uint8_t byte;`
	
	`#include "XbotDef.h"`
	`#include "Pfm.h"`
	`#include "Uson.h"`
	`#include "Debug.h"`
	`#include "Inter.h"`
	
	`void setup() {` // initialisation
	`SetupXbot();`

Trig pin 15/A1 Echo pin 14/A0
Tell.h Pour afficher la variable `int highLow`
Inter.h La routine d'interruption qui gère les tâches précédentes
Les moustaches se lisent dans le programme principal avec des `if(ObsL) {..}` et `if (ObsR) {..}`
	`  SetupUson();`
	`  SetupInter();`
	`}`

Le but du programme est de maintenir le robot à une distance donnée d'une paroi. Le programme compare la distance mesurée avec la distance voulue et active les moteurs pour corriger.

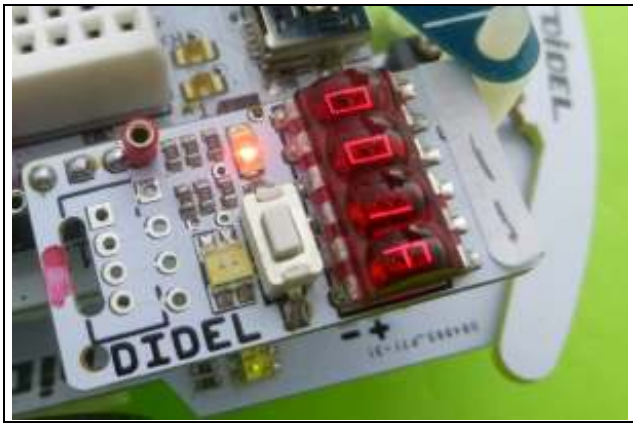
Video provisoire en attendant une version retravaillée
www.didel.com/UsonTellAcouper.MP4

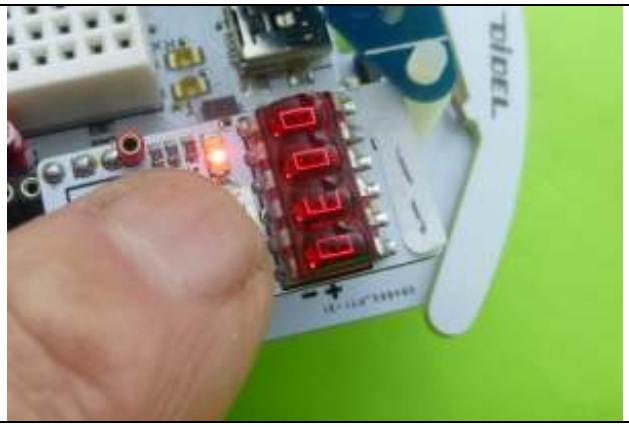
Boucle du programme StopObstacle.ino:

```
//  Le robot reste à une distance donnée de l'obstacle
#define Position 15  // distance stop
#define Ecart 2   // demi-distance stop
#define Vit 40  // vitesse à modifier
void loop() {
    highLow= distUson;
   // On decide, 3 possibilités
   if (distUson > Position+Ecart ) {   // trop loin, on avance
      pfmL=Vit; pfmR=Vit;
   } else if (distUson < Position-Ecart) {  // trop près, on recule
      pfmL=-Vit; pfmR=-Vit;
   } else {   // dans la zone, on stoppe
      pfmL=0; pfmR=0;
   }
   delay (10); // attendre avant de recommencer
}
```

Pour exécuter le fichier StopObstacle.ino il faut ouvrir le zip www.didel.com/xbot/LibXUson.zip et sauver le croquis StopObstacle. Sous Arduino, ouvrir ce fichier et cliquer sur StopObstacle.ino. L'écran montre le programme principal et les fichier de librairies. Télécharger comme d'habitude.

Ne pas enclencher les moteurs et observer la valeur sur les leds bicolores. La distance est en centimètres, la valeur est affichée par défaut en hexa. Presser sut le poussoir pour avoir la valeur convertie en décimal.





Enclenchez les moteurs et observez le comportement
On changera les paramètres pour comprendre l'effet du paramètre vitesse et Ecart.

L'exercice suivant est de corriger proportionnellement à l'écart de distance.
La vitesse PFM est 8 bits signée, entre –100 et +100.
La distance d'expérimentation peut être inférieure à 200.
Le programme le plus .simple est donné ci-contre.
	`void loop() {`
	`   highLow= distUson;`
	`   pfmL= distUson-Ecart;`
	`   pfmL= distUson-Ecart;`
	`}`

Pour créer ce programme, le plus simple est de sauver le croquis du programme précédent sous un nouveau nom, par exemple StopObstacleProp0.ino
Ceci crée le croquis (dossier) StopObstacleProp0. Ne pas oublier de sauver le programme avant de renommer.

La correction est trop lente? Créez StopObstacleProp1.ino en multipliant la vitesse par un facteur. Si la vitesse risque de dépasser 100, il faut saturer

```
if (pfmL>100) {pfmL=100; }  if (pfmL<-100) {pfmL=-100; }
```

Il peut encore y avoir un problème si la vitesse avant saturation a dépassé 8 bits. Il faut alors faire les calculs en 16 bits en déclarer `int pfmL,pfmR; .`

Un réglage proportionnel sans zone morte oscille toujours. Il faut un réglage PI ou PID (pour ceux qui ont fait du réglage automatique) mais la résolution du capteur n'est pas suffisante pour appliquer les formules théoriques.

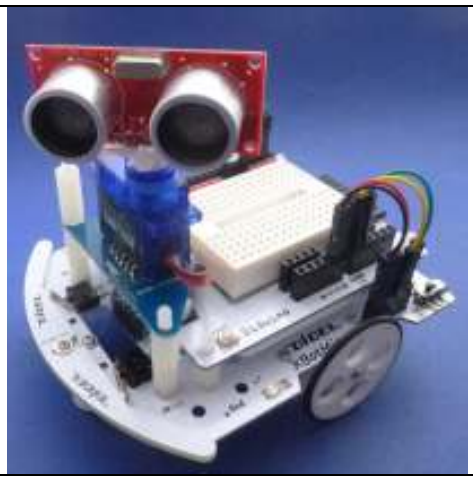
Rappelons que les capteurs ultrasons donnent souvent des valeurs erronées: un fil qui passe à proximité suffit pour créer un echo fixe! La directivité est très mauvaise. Il faut s'écarter de 20 degrés d'un obstacle, même petit, pour que l'écho vienne de plus loin. A partir d'un certain angle, une plaque fait miroir et la distance augmente brusquement.

Bouger le sonar avec un servo

Fixer un servo pour orienter le capteur est un bon exercice.

La librairie "Servo.h" n'offre que 25 angles possibles, mais le servo se connecte sur n'importe quelle pin (modifier les defines et le Setup).
La variable `byte angleServo` accepte les valeurs 0 à 25 (ppm de 0.8 à 2.4ms).

Balayer en faisant tourner le robot sur lui-même est possible, mais moins précis,



Le programme ServoUsonTell.ino balaye et revient sur la position de distance minimum. Il recommence après 5 secondes.

```
// ServoUsonTell.ino  151012  2792 octets
// Utilise LibX
typedef uint8_t byte;

#include "Xbotdef.h"
#include "Pfm.h"
#include "Debug.h"
#include "Uson.h"
#include "Servo.h"
#include "Inter.h"

void setup() {
  SetupXbot();
  SetupDebug();
  SetupUson();
  SetupServo();
  SetupInter();
}
```

```
byte distMin, posDistMin;
void loop () {
  distMin=255;
  for (angleServo=0; angleServo<25;
angleServo++) {
   delay (200);  // attente déplacement
   highLow = (angleServo<<8) +
distUson;
   if (distUson<distMin) {
     distMin=distUson;
     posDistMin = angleServo;
   }
 }
 delay (1000);
 angleServo = posDistMin ;
 delay (500);
 highLow = (angleServo<<8) + distUson;
 delay (5000);
}
```