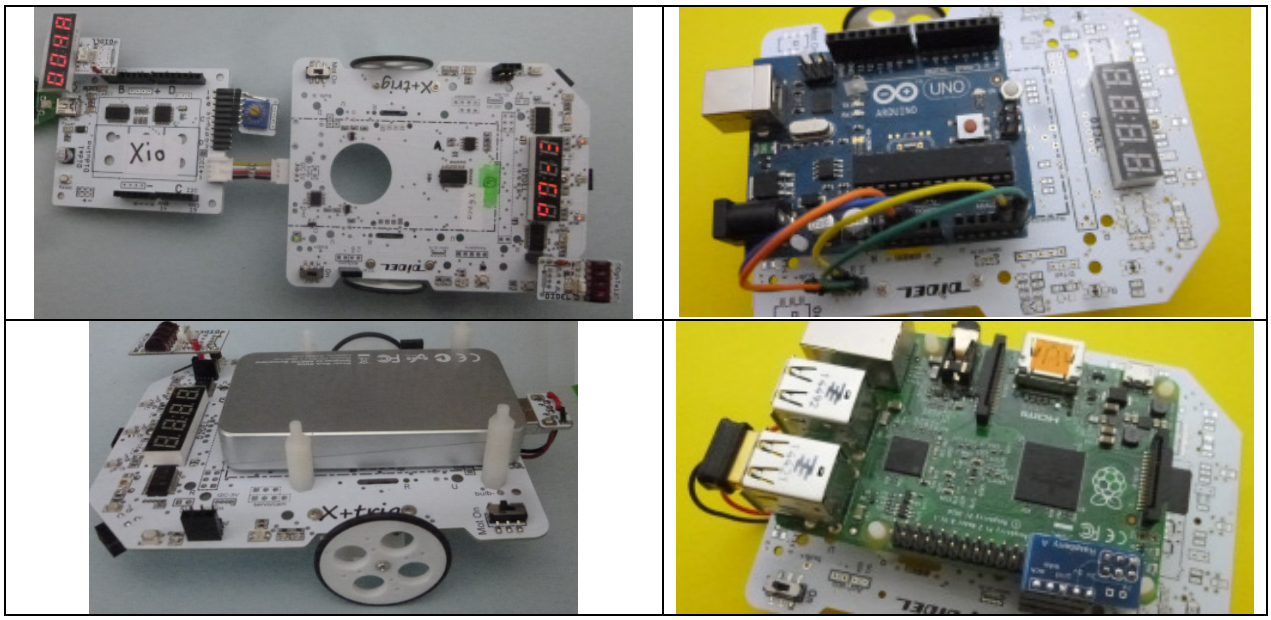
# X+go piloté par Arduino – premier contact

Il y a beaucoup à découvrir avec le X+go. Il comporte 3 microcontrôleurs avec 3 adresses et pour chacun quelques commandes I2C.
Xmot (0x21) gère les moteurs à vitesse variable et l'encodeur
Xtell (0x20) affiche des variables et des textes selon les besoins de l'application
Xio (0x22) gère les capteurs (2 poussoirs, 3 capteurs de distance, 2 de suivi, 2 d'ambiance lumineuse, un capteur ultrason) et actuateurs (2 servos optionnels).

Les extensions prévues sont une caméra linéaire XcamLin (0x23) et un mécanisme de caméra orientable XCamor (0x24). Des modules boussole, accélération sont facilement ajoutés sur les 5 prises I2C à disposition sur la carte, en 5V et en 3V.

Ce document explique un minimum pour voir le robot rouler et afficher des consignes et valeurs de capteurs.
Il faut passer du temps pour bien comprendre les fonctionnalités du Xgo, imaginer des programmes de test et des mini-applications, avant de définir un projet qui peut utiliser des capteurs et afficheurs supplémentaires.

**Câblage**
L'alimentation par le câble USB de programmation permet des déplacements d'une dizaine de centimètres pour vérifier les comportements de base. Des LEDs bicolores et un interrupteur permettent de "voir" la vitesse des moteurs sans que le moteur bouge – pratique pour la mise au point. On verra plus loin des solutions pour donner de l'autonomie au robot




## Un programme démo

Pour se mettre dans le bain, demandons au robot de rester à quelques cm d'un obstacle. Le capteur avant ObsC (obstacle centre) va nous donner une valeur 8 bits utilisée pour décider si le moteur doit avancer ou reculer.
Première chose: lire le capteur. La librairie LibXioWire nous évite de comprendre la librairie Arduino Wire, mais il faudra l'importer et l'initialiser, de même que les 2 autres librairies du Xgo, que l'on étudiera en détail en temps voulu. Comprenons d'abord notre exemple.

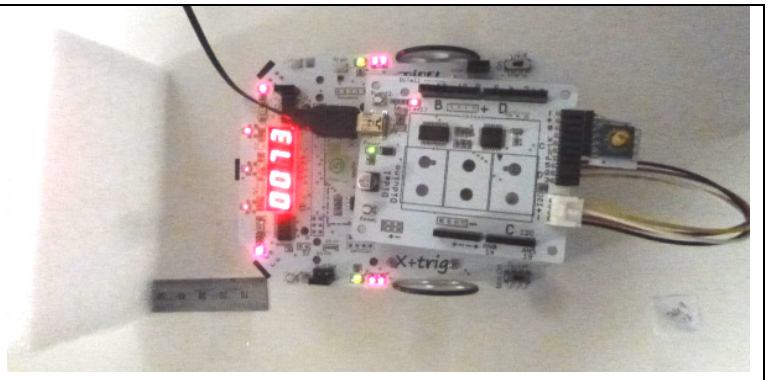
La fonction qui lit le capteur est ReadObsC (). Elle rend une valeur 8 bits, 0xFF pour une distance supérieure à 10cm (détails plus loin) et une valeur presque nulle si l'obstacle est presque contre le capteur.
La fonction qui fait avancer les moteurs est WritePfm (vitGauche,vitDroite) avec une vitesse de -128 à +127 (8 bits signés).
Le programme est alors aussi simple que cela:

```
byte dist;
#define MinDist 0x58
#define MaxDist 0x60
#define Vit 30  // max 127
void loop () {
  dist = ReadObsC (); WriteHex (dist);
  if (dist < MinDist) { WritePfm (-Vit,- Vit); }
  if (dist > MaxDist) { WritePfm (Vit, Vit); }
  if ((dist < MaxDist)&(dist > MinDist)) { WritePfm (0,0);}
  delay(50);
}
```

Ce programme XkeepDistance.ino est précédé par les #include des librairies et le set-up, et il est accompagné par les librairies expliquées plus loin. Ce programme et les suivants se trouvent sur www.didel.com/xbot/XplusGo.zip.

On a défini deux distances et une vitesse que l'on va corriger en observant la valeur du capteur et le comportement du robot.
On voit l'instruction `WriteHex (dist);` qui affiche la distance à chaque boucle, toutes les 50ms.

Sans mettre en route les moteurs, on peut voir sur les LEDs bicolores qu'il y a réaction.

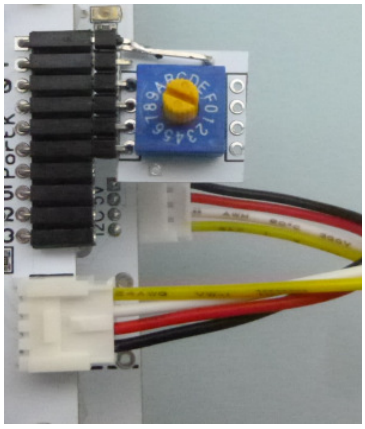


Avec les moteurs en route, on voit que des oscillations apparaissent selon la vitesse et l'intervalle. C'est là que cela devient intéressant. Quelles sont les limites du tout-ou-rien, comment faire en proportionnel?

## Test de la partie moteur et encodeur

Les programmes de test utilisent l'afficheur Tell et le switch à 16 positions sur la carte Diduino. Le même affichage Tell est aussi utilisé avec Xmot et Xio.

### Programme TestXmotLib.ino



Le commutateur permet de choisir un test. S'il n'est pas disponible, on remplace sa lecture par une valeur choisie et on recompile.
Les tests sont faits de préférence avec un module Tell sur la carte Arduino/Diduino pour afficher les valeurs lues.
L'interrupteur moteur permet de comprendre la plupart des tests sans faire tourner les moteurs, en tournant les roues à la main si on veut observer l'effet de l'encodeur.

Les tests affichent l'information sur l'affichage du XGo. En plus, (pour aider notre mise au point), il y a affichage sur l'affichage Tell de la carte Diduino ou Arduino et sur l'affichage de test Tell éventuellement inséré sur la carte sur le bon connecteur.

Test 0 Lit le code d'identification 0B0
Test 1 Inutilisé (ajoutez votre propre test)
Test 2 Avance-recule en mode PFM avec une vitesse +127 -127
Test 3 Avance-recule en mode Speed avec une vitesse +20 -20
Test 4 Avance-stoppe le moteur droit en mode Speed et affiche sur DiTell le nombre de pas en 16 bits (le robot tourne sur lui-même).
Test 5 Avance-stoppe le moteur gauche en mode Speed et affiche sur DiTell le nombre de pas.

Le fichier **LibXmotWire.h** définit des fonctions qui facilitent le développement des applications qui utilisent la partie moteur.

| | | |
|---|---|---|
| v8=ReceiveXmotId (); | uint8_t (byte) | Lit l'identificateur = 0xB0 [1] |
| v8=ReadXmotStatus (); | uint8_t (byte) | |
| WritePfm (pfmLeft,pfmRight); | int8_t (char) | -128 à +127 |
| WriteSpeed (speedL,speedR): | int8_t (char) | -20 à +20 (saturé si différent) |
| v16= ReadEncoL(); | int16_t (int) | max +/-32'000 = 45 mètres |
| v16 = ReadEncoR() | int16_t (int) | 1.5mm par incrément |
| WriteEncoL(v16); | int16_t (int) | |
| WriteEncoR(v16); | int16_t (int) | |
| WriteXmotTell (v16); | uint16_t (uns int) | Option sur connecteur |
| WriteXmotTell88 (h8,l8); | uint8_t (byte) | Modes décimal au poussoir |

Autres document: www.didel.com/xbot/XplusGo.pdf- expliquera les commandes

## Test de la partie affichage

L'affichage Xtell du X+go est identique à l'affichage I2C DgTell, Comme périphérique I2C, Xtell répond à des commandes décrites sous www.didel.com/grove/DgTell.pdf

Pour la mise au point des applications, on va essentiellement utiliser l'affichage de nombres. Pour les tests, c'est pratique d'utiliser le switch à 16 positions. Comme Xtell n'envoie pas de paramètres, l'affichage Tell est inutile.

**Programme TestXtellLib.ino**
Le commutateur permet de choisir un test. S'il n'est pas disponible, on remplace sa lecture par une valeur choisie et on recompile.

Test 0 Lit le code d'identification 0A0
Test 1 Affiche 0x1000 en hexa
Test 2 Affiche 0x1000 en décimal
Test 3 Affiche 1000 en décimal
Test 4 Affiche ABCD fixe
Test 5 Affiche les 4 codes donnés en mode Ascii
Test 6 Affiche les 4 codes donnés en mode Segments
Test7 Fait défiler l'alphabet

Le fichier **LibXtellWire.h** est le fichier à utiliser pour les développements qui utilisent l'afficheur.

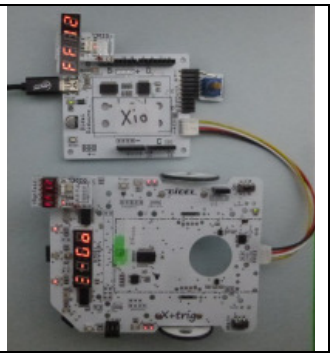
| | | |
|---|---|---|
| v8=ReceiveXtellId (); | uint8_t (byte) | Lit l'identificateur = 0xA0 |
| (WriteMode(v8); pour utilisations spéciales) | | |
| WriteHex (v16); | uint16_t (unsigned int) | Affiche le nombre en hexa |
| WriteDec (v16); | uint16_t (unsigned int) | Affiche le nombre en décimal |
| Write4Car (v8,v8,v8,v8); | uint8_t (byte) | Ecrit 4 car ascii |
| Write4Seg (v8,v8,v8,v8); | uint8_t (byte) | Ecrit 4 codes segment |
| WriteCar (v8); | uint8_t (byte) | Scroll mode, on injecte un caractère à droite |
| | | |
| WriteBlockAscii | *Mode block SMbus pour Raspberry/Python* | |
| WriteBlockSegments | | |

**Comparaison avec Serial.print**

| | | |
|---|---|---|
| WriteHex (0x1000);<br>*(pas de point décimal)* |  | Serial.print (0x1000,HEX); |
| WriteDec (0x1000);<br>*(avec point décimal)* |  | Serial.print (0x1000,DEC); |
| WriteDec (1000);<br>*(avec point décimal)* |  | Serial.print (1000,DEC); |

## Test de la partie capteurs

Le 3e processeur commande quantité de capteurs et leds, que l'on peut lire individuellement ou en bloc.

**Programme TestXioLib.ino**

Le commutateur permet de choisir un test. S'il n'est pas disponible, on remplace sa lecture par une valeur choisie et on recompile. Certains tests affichent sur le DiTell optionnel, à ne pas confondre avec le module Tell sur la carte Diduino.
On peut afficher en I2C sur l'affichage Xtell. C'est fait dans le programme

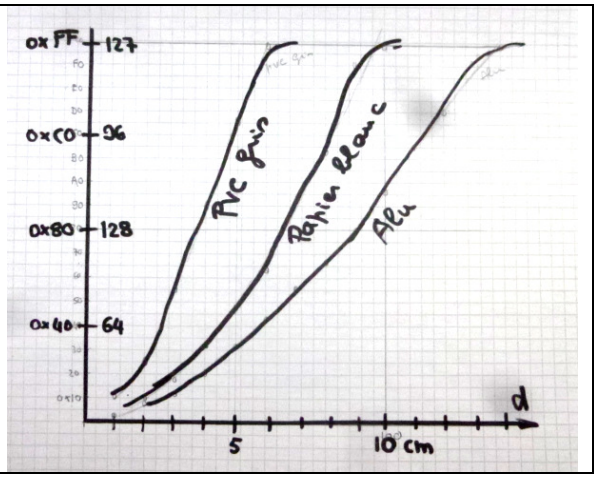
Test 0 Lit le code d'identification 0C0
Test 1 Clignote les 5 leds avant
Test 2 Lit les poussoirs, affiche la valeur lue (1 poussoir gauche, 2 poussoir droite)
et allume/éteint la lampe arrière si elle est câblée (affiche 0x10 pusque bBulb=4)
Test 3 Lit le capteur d'obstacles devant, affiche la valeur analogique lue
Test 4 Lit les capteurs d'obstacle G et D, affiche la valeur analogique lue
Test 5 Lit les capteurs de sol G et D, affiche la valeur lue
Test 6 Lit les capteurs de lumière G et D, affiche la valeur lue
Test 7 vide
Test 8 Met les leds en mode Echo, l'intensité dépend de la mesure du capteur
Test 9 Teste l'intensité variable des leds
Test A Oscille les 2 servos, si connectés. Valeur 0 à 22 saturé

Le fichier **LibXioWire.h** est le fichier à utiliser.

| | | |
|---|---|---|
| v8=ReceiveXioId (); | uint8_t (byte) | Lit l'identificateur Id = 0xC0 |
| v8= ReadButton (); | uint8_t (byte) | **bit 0** **bButtonL**<br>**bit 1** **bButtonR**<br>**bit 4** **bBulb** |
| BulbOn (); BulbOff (); | | |
| v8= ReadObsC (); | uint8_t (byte) | distance devant |
| v16= ReadObsLR (); | uint16_t (gau<<8)+dte | distance gauche-droite |
| v16= ReadSolLR (); | uint16_t (gau<<8)+dte | distance gauche-droite |
| v16= ReadBrightLR (); | uint16_t (gau<<8)+dte | intensité lumière gauche-droite |
| WriteModeLeds(v8); | WriteIoTell88 | **0** **bCopy**<br>**1** **bSetLeds**<br>**2** **bOnOff** |
| WriteLedObsC (v8); | uint8_t (byte) | |

| | | |
|---|---|---|
| WriteLedObsLR (v16); | uint16_t (gau<<8)+dte | |
| WriteLedSolLR(v16); | uint16_t (gau<<8)+dte | |
| WriteServo1(v8); | uint8_t (byte) 0..22 | |
| WriteServo2(v8); | uint8_t (byte) 0..22 | |
| WriteXIoTell (v16); | uint16_t | |
| WriteIoTell88 (v8,v8); | uint8_t | |
| *Mode block SMbus pour Raspberry/Python* | | |
| Pas sur Arduino | *Mode block SMbus pour Raspberry/Python* | |

## Capteur infrarouge de distance

| | |
|---|---|
| Les capteurs de distance ont une diode infrarouge qui éclaire devant et un photo-transistor qui mesure la lumière reçue. L'éclairage ambiant perturbe naturellement les mesures, mais la nature des surfaces réfléchissantes est plus importante, comme le montre les mesures ci-contre.<br>Le capteur de sol est également sensible à la couleur IR du sol et à la distance. Si l'application est de suivre une piste, il faut bien choisir la piste pour avoir un bon contraste. |  |

## Autonomie

Un carte Diduino comme pilote se contente d'un accu Lipo de 250 mAh et sa tension de 3.7V. Pour un Raspberry3, un powerbank de portable convient. Le modèle Arctic PowerBank2000 tient entre les potelets qui fixent la carte Raspberry3 ou Arduino.
Voir pour plus de détails www.didel.com/xplus/XgoConnect.pdf

jdn 160717