Capteur de distance Xdist2Ir

Aspects pédagogiques

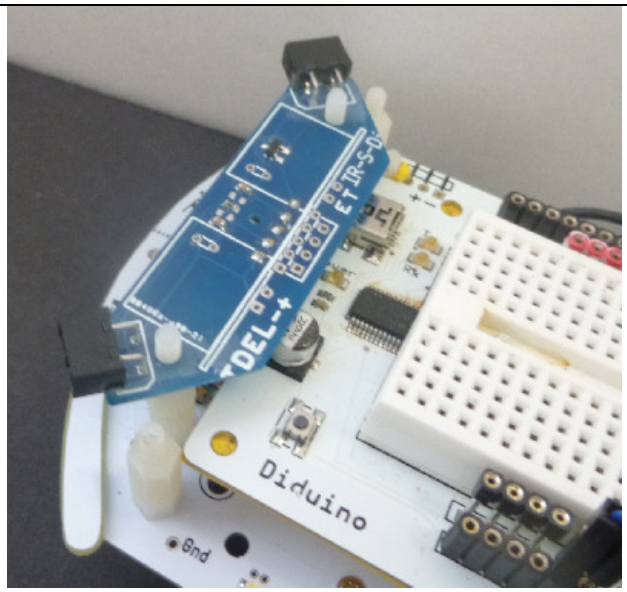
Liens aux docs sous prof/Xdist2Ir.html

Le capteur se fixe sur le connecteur avant du Xbot.
Si les élèves savent ce qu'est un niveau logique et la courbe de décharge d'un condensateur, ou si on a le temps de l'expliquer, le principe de fonctionnement est intéressant et permet de mesurer la résistance d'un composants résistifs non alimentés.

La fonction de lecture rend 2 paramètres et dure ~200ms. Comment vérifier cette durée? On répéte 20 fois et on mesurer la durée totale !

On utilisera le terminal pour afficher les 2 distances. Cela prend aussi du temps. Combien?

On mesure la distance en changeant le réflecteur et la lumière ambiante. L'occasion de faire des mesures comme en physique.
Le premier programme affiche les distances sur le terminal en appelant la fonction `GetDist();`.



Les sources des programmes sont sous www.didel.com/prof/Xdist2Ir.zip

Trois choses caractérisent cette fonction:
- elle est écrite en C portable
- elle est bloquante
- elle met à jour les variables globales
  `mesureG et mesureD`
- les distance rendues sont des bytes
  (il n'est pas réaliste de compter plus loin que 255).

Le programme principal est simple, puisque des variables globales sont utilisées.
Toutes les 10ms, on fait une mesure, mais on n'affiche que toutes les dixièmes de secondes.
Les définitions et fonctions sont en C portables, supposant une bonne familiarité aves les opérations logiques.

Note: sauver les programmes dans un dossier Dist2Ir-Moi

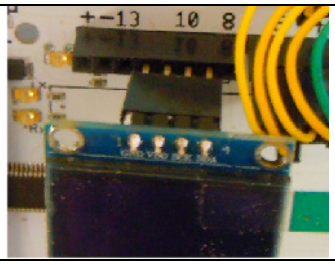
```
//Dist2IrTest1.ino
#include "XBotDist2IrDef.h"

void setup() {
  SetupDistIr ();
  Serial.begin(9600);
}

volatile byte cntAffi;
void loop() {
  GetDist () ;
  delay (10);
  // tous les 10x 10ms, on affiche
  cntAffi ++ ;
  if (cntAffi > 100) {  // on affiche
    cntAffi = 0 ;
    Serial.print("Distances ");
    Serial.print(mesureG);
    Serial.print(" ");
    Serial.println(mesureD);
  }
}
```

On peut réfléchir comment faire sans paramètre global. Une fonction ne peut rendre qu'un paramètre, il faut créer deux fonctions. La fonction `byte GetDistG ( ){..}` se terminera par `return cntG;`. Dans la boucle on écrira `Serial.print(GetDistG);` C'est apparemment plus simple, mais on a dédoublé une fonction qui prend du temps, et les 2 mesures ne sont plus simultanées.

Si l'algorithme n'a pas été expliqué et les opérations logiques pas familières, c'est quand même un très bon exercice. Les informaticiens doivent sans cesse intervenir dans un programme qu'ils n'ont pas écrit et qu'ils n'ont pas le temps de comprendre dans le détail.

Affichage Tell
S'il est disponible et installé (#include et setup), on affiche les deux valeurs avec l'instruction
`Tell ((mesureG<<8)+mesureD);`

Affichage Oled
L'affichage est connecté sur les pins 12..9. Mettre un prolongateur pour éviter la collision avec le connecteur.




```
//Dist2IrOledTest1.ino On affiche les 2 distances.
// IR connecté sur pins 14 PORTC bit 0
// Oled sur Arduino pins 12 11 10 9
#include "XBotDist2IrDef.h"
#include "OledI2Cbb.h"
#include "OledPix.h"
void setup() {
  SetupDistIr();
  SetupI2Cbb();
  SetupOledPix();
}
byte distG,distD;
byte x,yG,yD;
void loop() {
  GetDist () ;
  LiCol(2,0); BigDec8 (mesureG); BigDec8 (mesureD);
  yG=(mesureG/4); if (yG>=63) {yG=63;}
  yD=(mesureD/4); if (yD>=63) {yD=63;}
  //y 0..63, pour dist 0..255
  Dot (x,63-yG); DDot (x,63-yG);
  if(x++>127) {Clear(); x=0;}
  delay(20);
}
```

Eviter les murs
Si le robot bouge, il faut importer son fichier de définition et se souvenir des solutions en tout ou rien, PWM ou PFM (prof/Xbot.pdf).
Le premier exercice peut simuler la moustache comme dans un premier exercice avec le Xbot.
On remplace le `if(ObsG){..}` par `if (distG<DistMin) {..}`. Renommer et modifier TestDist1.ino en complétant avec ce qu'il faut prendre dans XbotTest0.

Ajouter le PWM
On ne cherche pas à avoir des vitesses lentes et on va en avant seulement. Le PWM Arduino convient bien dans ce cas. Rappelons les instructions à utiliser pour avancer:

```
analogWrite (bAvG, pwmG); digitalWrite (bRecG, LOW);
analogWrite (bAvD, pwmD); digitalWrite (bRecD, LOW);
```

Pour reculer:

```
analogWrite (bAvG, -pwmG); digitalWrite (bRecG, HIGH);
analogWrite (bAvD, -pwmD); digitalWrite (bRecD, HIGH);
```

Il faut avoir spécifié auparavant

```
#define bRecG 4 // bit 4 dans le PortD
#define bAvG 5
#define bAvD 6
#define bRecD 7
```

C'est mieux d'importer `Xbot.h` qui contient ces définitions et les moustaches.
Mais attention, StopGD; ne coupe pas les moteurs ! L'interruption qui a été mise en route par Arduino (on fait des choses derrière votre dos !) remet la vitesse précédente dans la milliseconde.

Comment continuer?

La porte est ouverte pour d'innombrables défis. Utiliser le PFM avec sa librairie, mettre en route des interruptions que l'on maitrise, ajouter des librairies, est essentiel pour maîtriser des applications en C, mais ce n'est la l'objectif d'une première approche de la programmation.

Références – cliquables dans *Xdist2IrProf.html*

Xdist2IrProf.zip Les programmes de cette notice
xDist2Ir.pdf documentation 1401
LibXDist2Ir.pdf utilisation de la librairie par interruption 1510
Oled et capteur de distance IR
Vidéos
SuitCorridor https://www.youtube.com/watch?v=AULplRbHatU

jdn 180813