CH-1092 Belmont, Switzerland
Tel +41 21 728-6156, Fax 728-6157
Email info@didel.com, www.didel.com

DiDEL SA

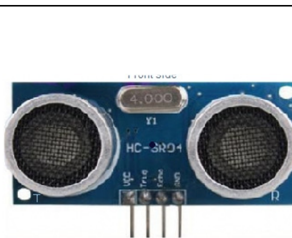
Expérience avec un capteur ultrason

1 Introduction

Ce document s'adresse à des intéressés à la programmation, qui connaissent Arduino et veulent approfondir leur connaissance. Même ceux qui ont suivi le MOOC EPFL (et peut-être surtout ceux-là) peuvent profiter de notre progression pas-à-pas et des explications associées. Arduino est excellent pour voir ce que l'on peut faire, mais c'est comme IKEA: vous câblez comme sur le dessin, vous chargez le programme, et vous êtes content.
Bien utiliser des capteurs suppose beaucoup de connaissances, on va le voir avec l'étude du capteur ultrason SR04 ou SR05 qui ne coûte que quelques francs et est souvent utilisé naïvement.
L'affichage Oled SSD1306 est très utile pour visualiser des signaux, remplaçant le terminal Arduino, avec l'avantage de faire partie de l'application contrairement au terminal série.

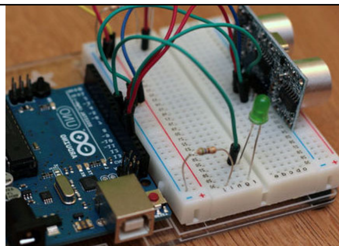
1.1 SR04 et SR05

Le principe d'un capteur ultrason est simple et bien expliqué sur le Web. Les SR04 et 05 ont presque la même électronique, le SR05 a une pin de plus pour dire que le signal est revenu, mais c'est peu utilisable.
C'est le problème des sonar: si le signal ne revient pas, combien de temps faut-il attendre? Le SR05 fixe la limite à une durée équivalente à 3-4 mètres, moins que le SR04 que l'on a vu attendre des dixièmes de secondes.



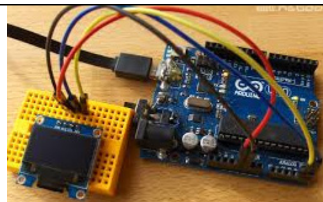
1.2 Câblage

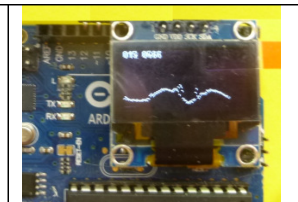
Les exemples de câblage sont nombreux sur le web. Vous êtes libre d'utiliser votre breadboard, mais notre solution est élégante, avec son fichier de définition adapté.
Le SR05 demande une tension supérieure à 4.2V (parfois seulement 3.5V). Le courant moyen est de quelques mA.



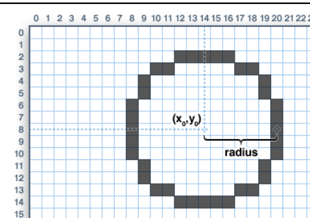
1.3 Oled 1306 I2C

Le composant qui va nous faire plaisir est l'affichage Oled 1306 I2C facile à obtenir pour une dizaine de francs. Il se branche traditionnellement sur les sorties I2C (A4 A5), mais notre soft offre plus de liberté, ce qui permet de le câbler ou cela nous arrange.





Pour utiliser la librairie AdaFruit; chercher "adafruit-gfx-graphics-library". Le graphisme est orienté texte et géométrie.
La documentation Didel 2015 utilise cette libraire et propose des exemples
www.didel.com/diduino/OledI2C.pdf
www.didel.com/diduino/OledI2C.zip



1.4 Librairie compacte LibOledPix

La librairie AdaFruit est volumineuse et nécessite 1 kbyte de mémoire RAM. **LibOledPix** ecrit directement sur le Oled, les programmes sont plus courts, le chargement plus rapide, les fonctions plus simples à mémoriser. De plus, le câblage est possible sur 2 pins quelconques d'un même port.

2 Programmer le capteur ultrason SR04 ou SR05

Ce capteur est bien connu. La documentation dit: courant de repos <2mA, courant pendant l'impulsion d'envoi 15mA, angle mesuré ~30°, distance 2cm à 4m, résolution 0.3cm.

Il faut commencer par le câbler et le tester avec affichage du résultat sur le terminal série d'Arduino.

2.1 Test Arduino

Branchez un capteur ultrason SR04 ou SR05 sur vos pins préférées. (Si vous avez un Xbot, voir http://www.didel.com/xbot/DistSonar.pdf). Utilisons A1 (pin15) en sortie pour le Trig et A0 (pin14) en entrée pour l'Echo. Une impulsion de plus de 10ms sur Trig lance un train d'impulsions sonores à 40 kHz qui dure 0.4ms. Dès que ce train est parti, l'électronique du SR05 active le signal Echo, qui est remis à zéro quand un son revient avec la même fréquence. Comme l'intensité diminue avec le carré de la distance, le capteur ne mesure correctement à 3 mètres que si les conditions sont parfaites. Les réflexions parasites faussent la mesure idéale en face, c'est ce que l'on pourra bien observer avec le Oled.

Le programme TestSonar.ino utilise la fonction Arduino pulsIn() qui a deux paramètres: le numéro de la pin testée et un état logique. Cette fonction bloquante mesure la durée d'une impulsion positive ou négative en microsecondes.
La vitesse du son est 330m/s, donc 0.33mm/µs. Avec l'aller-retour, cela fait 0.165mm pour 1µs, ou 1cm en 60us, 3m en 18ms. Si l'echo n'est pas reçu, ce qui peut arriver avec un obstacle proche qui fait miroir à 45 degrés et envoie le son très loin. il ne faudrait donc pas attendre plus de 20ms. Les SR04 et SR05 sont réglés pour des valeurs supérieures.

2.2 Premier programme de test – *tous les programmes sont dans www.didel.com/OledLibDemo1.zip*

```
    // TestSonar.ino
    #define Trig A1
    #define Echo A0
    void setup () {
      pinMode(Trig, OUTPUT);
      pinMode(Echo, INPUT);
      Serial.begin(9600);
    }
    int dist ; // temps mesuré
    void loop () {
    digitalWrite(Trig,HIGH);
    delayMicroseconds(20);
    digitalWrite(Trig,LOW);
    dist = pulseIn(Echo,HIGH);
    Serial.println(dist);
    delay (1000) ; // une mesure par seconde
}
```

Ce programme est compatible avec le câblage du module Uson sur Xbot.

Charger TestSonar2 pour tester avec l'insertion directe sur le portB (photo en page 3).

Vérifiez les définitions et le câblage si cela ne fonctionne pas.
Avec assez d'espace libre devant le capteur, on peut vérifier que la distance à un grand réflecteur porté par un assistant augmente jusqu'à 2-4 mètres, puis saute à la valeur nettement plus grande correspondant au temps limite quand le signal n'est plus reconnu (voir à une petite valeur si le compteur de temps a dépassé 65536).

2.3 Fonction GetSonar()

Ce dont on aura besoin dans un programme, c'est une valeur donnée en appelant une fonction GetSonar(). Les instructions détaillées doivent être encapsulée dans une librairie; une fois testée. Cette librairie, sous forme d'un fichier inclus, doit initialiser les entrées-sorties utilisées (setup) et définir la fonction.
Déclarations et fonction setup :

```
    #define Trig A1
    #define Echo A0
    #define TrigOn digitalWrite(Trig,HIGH);
    #define TrigOff digitalWrite(Trig,LOW);
    void SetupUson () {
      pinMode (Trig,OUTPUT);
      pinMode (Echo,INPUT);
    }
```

Fonction GetSonar():

```
    int GetSonar () { // bloquant, durée 1 à 100ms
       TrigOn
```

```
    delayMicroseconds(20);
    TrigOff;
    return (pulseIn(Echo,HIGH));
  }
```

Dans la boucle d'un programme de test:

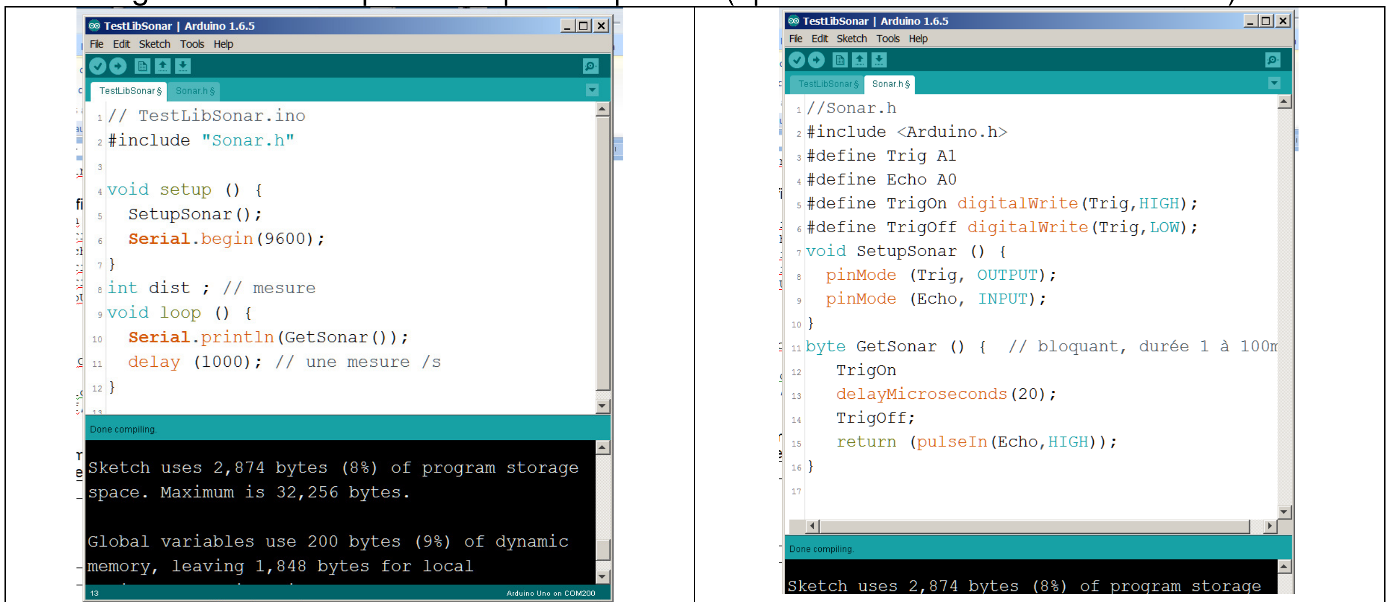
```
Serial.println(GetSonar());
```

2.4 Librairie Sonar.h

Les instructions du setup et la fonction de lecture, une fois définies, n'ont pas de raison de rester avec les instructions de l'application. On les regroupe dans un fichier "de librairie" avec l'extension .h

```
// Sonar.h
#define Trig A1
#define Echo A0
#define TrigOn digitalWrite(Trig,HIGH);
#define TrigOff digitalWrite(Trig,LOW);
void SetupUson () {
  pinMode (Trig,OUTPUT);
  pinMode (Echo,INPUT);
}
byte GetSonar () { // bloquant, durée 1 à 100ms
    TrigOn
    delayMicroseconds(20);
    TrigOff;
    return (pulseIn(Echo,HIGH));
}
```

Ce fichier est appelé comme fichier inclus dans le programme **`TestLibSonar.ino`**.
Arduino montre côte à côte les fichiers inclus, que l'on peut éditer comme le fichier.ino. Le sauvetage est automatique à chaque compilation (option dans le menu "Préférences").



Rappelons qu'Il faut faire des programmes de test à chaque étape, et les conserver.

2.5 Astuce de câblage

Les fils, c'est désordre et peu fiable. Il nous faut 4 signaux pour le SR04/05, puisque le courant est faible, on peut câbler les alimentations sur des sorties, la chute de tension est acceptable jusqu'à 15-20 mA.
Le nouveau groupe set-up et définitions est alors

```
#define Vcc 8
#define Trig 9
#define Echo 10
#define Gnd  12
```

```
#define TrigOn
digitalWrite(Trig,HIGH);
#define TrigOff
digitalWrite(Trig,LOW);
void SetupSonar () {
  pinMode (Trig, OUTPUT);
  pinMode (Echo, INPUT);
  pinMode (Vcc, OUTPUT);
  pinMode(Gnd, OUTPUT);
```

Autre définition en C sur le port D (Sonar3.ino)

```
#define bVcc 0
#define bTrig 1
#define bEcho 2
#define bGnd 4
#define TrigOn bitSet   (DDRB,bTrig)
#define TrigOn bitClear (DDRB,bTrig)
#define EchoHigh PINB & (1<<bEcho)
void SetupSonar () {
```

`digitalWrite (Vcc,HIGH);`
`digitalWrite (Gnd,LOW);`
`}`
	`DDRB |= (1<<bGnd|1<<bVcc|1<<bCk|1<<bDa) ;`
	`PORTB|= 1<<bVcc ;`
	`PORTB&= ~(1<<bGnd|1<<bCk|1<<bDa) ;`
	`}`

Le programme à tester est TestSonar2.ino, qui utilise les fonctions Arduino, que l'on peut comparer avec les programmes TestSonar3.ino et TestSonar4.ino, plus proches du C.

Le 2e groupe d'instructions n'utilise pas de fonctions Arduino. Il est compatible avec tous les processeurs qui ont un compilateur C, et le nombre d'instructions générées est considérablement réduit.

2.6 Curieux par la taille des programmes?

Le programme TestLibSonar.ino est indiqué 2874 bytes. En enlevant le Serial.begin(9600); et le Serial.println(), on a 804 bytes. En enlevant le delay() il reste 642 bytes pour l'initialisation et nos instructions. En les enlevant, il reste le minimum absolu Arduino, 450 bytes. Donc notre librairie Sonar.h prend 200 bytes, ce qui est beaucoup; pulseIn() est général, donc peu efficace. Ecrire une fonction équivalente pour cette application demande 2 while() et prend 20 bytes au lieu de 200!

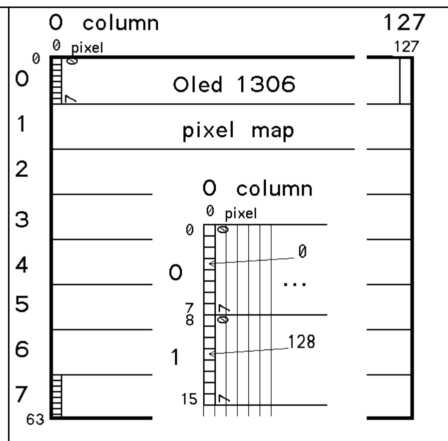
3 Oled a la place du terminal

Pour observer les valeurs d'un capteur sur un robot qui roule, suivre le robot avec un portable qui affiche les valeurs sur le terminal série n'est pas une solution. Transmettre par Wifi ou BT n'est maîtrisé que par des spécialistes, et il faut alors plus performant qu'un Arduino !
Un Oled est simple et sympa, comme vous allez voir.

3.1 LibOledPix

Utilisons la librairie LibOledPix. Si vous connaissez bien la librairie AdaFruit, c'est facile de transposer les noms, les fonctions de base sont similaires.
Cette librairie a la forme d'un ensemble de fichiers inclus. Il n'y a rien à installer.
Chaque exemple que vous trouvez dans le zip contient les fichiers nécessaires. Le croquis contient le .ino et les .h, tout est sauvé avec Tool – ArchiveSketch. Très pratique puisque la date est mentionnée, un back-up sans effort!
Si on modifie un fichier inclus, la modification n'est pas reportée dans les autres croquis qui ajoutent ce fichier. C'est parfois embêtant pour reporter partout une amélioration ou correction. Mais c'est excellent pour apprendre à programmer en modifiant les programmes; ce que l'on modifie ne touche que le programme chargé. Gérer les fichiers .h doit souvent se faire en passant par le gestionnaire de fichier. Arduino ne peut pas renommer ou enlever un .h du croquis. Aller les chercher dans un autre croquis est facile. http://www.didel.com/coursera/FichiersInclus.pdf donne quelques compléments.

Voir http://www.didel.com/OledLib.pdf pour des explications générales en anglais pour le SSD1306. On utilise ici la librairie OledPix, et le but est d'utiliser l'affichage pour aider à la mise au point de programmes utilisant des capteurs et des machines d'état.
Suivez la notice www.didel.com/Oled1306.pdf pour une mise en route progressive dans la compréhension de OledPix.
Rappelons que les nombres et textes ont une coordonnée de début en ligne/colonne et les pixels en x/y. Abscisse y et colonne sont identiques. Les lignes de texte vont de 0 à 7 et les pixels de 0 à 63, origine en haut.
Avant de positionner un caractère ou un nombre, il faut spécifier sa position avec la fonction LiCol(li,co). La fonction Dot(x,y) travaille avec des coordonnées plus fines.



Exemple `LiCo(2,0); Car('a');` place un a en début de 2e ligne.
Par contre `LiCo(7,100); DecBig(123);` qui cherche à mettre en bas d'écran un nombre décimal de 3 digits en gros caractères ne pourra pas afficher tout le nombre, il a besoin de 30 colonnes.
Si les noms des fonctions de la libraire ne conviennent pas, c'est facile de les redéclarer,

3.3 Câblage

La librairie OledI2Cbb permet de câbler le 1306 sur n'importe quelle pin. L'affichage ne consomme que quelques mA, on va donc faire le même truc qu'avec les SR05. Toutefois, les timings sont critiques et on ne peut pas utiliser les fonctions Arduino. Il faut des définitions en C que l'on trouve dans OledI2Cbb.h

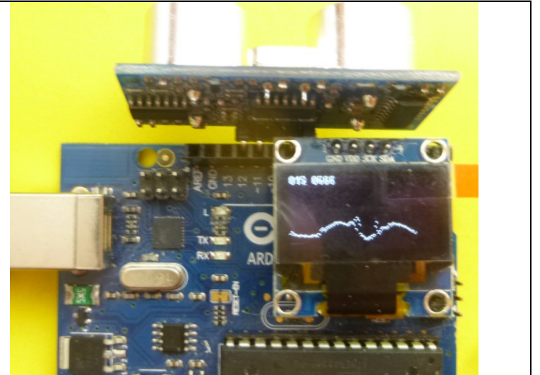
Si les opérations logiques ne vous sont pas familières, http://www.didel.com/C/OperationsLogiques.pdf donne les bases nécessaires. Arduino est très lourd et peu efficace quand on doit agir sur plusieurs bits d'un même mot.

3.2 Test de la librairie OledPix

Les fichiers à inclure sont les suivants:
OledI2Cbb.h (pins quelconques) communications avec le 1306
OledPix.h Fonctions alphanumériques et graphiques
Revérifiez avec les programme dans www.didel.com/Oled1306.zip que tout se passe bien.
Le fichier TestOledPix.ino et ses fichiers inclus permet de tester les fonctions à disposition en mettant et enlevant les // des commentaires.

4 Ajoutons le SR05

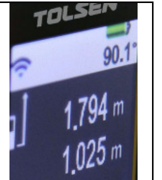
Avec notre approche modulaire, il suffit d'insérer le SR05 sur les pins précédemment définies et d'ajouter dans le dossier TestLibOlexPix le fichier Sonar.h, l'inclure et appeler son setup. On a alors accès à la fonction GetSonar(). La valeur obtenue est affichée avec Hex16() ou Dec9999() et on peut réfléchir comment afficher cette distance sous forme graphique.
En X, il suffira d'incrémenter toutes les 20ms par exemple.
En Y, on voudrait avoir l'origine en bas, et on doit se limiter à 63 pixels. Le problème sera donc de définir quelle gamme de distances pour 0 à 63 pixels.



4.1 Afficher la distance

Comme premier test, on veut afficher la distance sur le Oled.

La "distance" est une valeur en microsecondes. Si on veut que cette valeur soit lue par un humain, on va faire une division en virgule flottante comme on voit facilement des exemples Arduino. Mais notre capteur est imprécis, et on ne parle pas en mètres avec un robot, On veut savoir si l'obstacle est proche, plus simple de travailler avec les nombre bruts. On les transformera quand on les aura bien compris.



La librairie Uson n'utilise plus le pulsin; elle est portable et nettement plus courte.
Le résultat est 16 bits, on utilise la fonction Dec9999(); qui a ce nom parce qu'elle limite l'affichage à 9999.

```
//TestOledUson.ino
typedef uint8_t byte;
#include "OledI2C.h"
#include "LibOledPix.h"
#include "Uson.h"
void setup(){
  SetupI2C();
  SetupOledPix();
  SetupUson();
}
int dist;
void loop() {
  dist = GetSonar ();
  Licol(2,0);  // 2e ligne)
  Dec9999 (dist);
  delay (200);
}
```

4.3 Fonction oscilloscope

L'écran graphique peut afficher la courbe des mesures faites a une fréquence max qui dépend du temps d'écriture d'un pixel (50 µs), et naturellement du temps de préparation de ce pixel.
Dot(x,y) a son origine en haut à gauche. Il faut parfois un peu réfléchir pour mettre cette origine en bas en tenant compte d'un facteur d'échelle.
Pour faire un oscilloscope, la valeur x est incrémentée à la période voulue. En bout d'écran, on peut effacer, ou laisser les traces se superposer. Il y a toutefois un effet expliqué dans www.didel.com/OledI2C.pdf: le Oled demande d'écrire les 8 bits verticaux dont le pixel fait partie. On peut donc en passant avec la courbe effacer un nombre. S'il est reconstruit à chaque cycle, ce qui prend 1ms, on ne voit rien.

Remarque: La librairie Adafruit n'a pas ce problème, le pixel se dessine dans un tampon de 1kilobytes. Pour voir le pixel, il faut transférer tout le bitmap en 60ms. Cela peut convenir pour le SR04/05, mais pas pour un signal plus rapide, qu'il faut enregistrer et jouer en bloc. LibOled compatible avec LibOledPix travaille aussi avec un tampon.

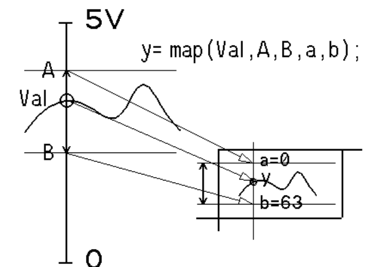
```
// TestOledUsonScope.ino
typedef uint8_t byte;
#include "OledI2C.h"
#include "LibOledPix.h"
#include "Uson.h"
void setup(){
  SetupI2C();
  SetupOledPix();
  SetupUson();
}
int dist; byte x,y;
void loop(){
  dist = GetSonar();  // 0-8000
  LiCol(0,0); BigDec9999 (dist);
  y=(dist/32); if (y>63) {y=63; x+=2;}
//la courbe sature pour dist/32=64, dist=2048=33cm
  Dot (x++,63-y);
  if(x>127) {Clear(); x=0;}
  delay(20);
}
```

4.4 Astuces

Les oscilloscopes à tube cathodique n'avaient qu'un faisceau d'électrons. Comment pouvaient-ils avoir plusieurs traces? Une solution appelée "Alt" changeait de canal à chaque retour de trace. Le "Chop" commutait rapidement entre 2 canaux, en éteignant l'intensité pendant la commutation.
C'est facile à programmer sur notre Oled. On le verra pour afficher la moyenne.
La fonction DDot(x,y) ajoute un 2e point en dessous (si c'est possible). On peut se fabriquer des "dot" en forme de croix ou d'étoile, mais c'est surtout un joli exercice de programmation.

4.5 Adapter l'échelle

On a vu comment diminuer l'échelle et saturer. Un problème plus général est de ramener une gamme de valeurs dans les valeurs affichables.
Par exemple, on mesure une tension avec analogRead(), la valeur obtenue est entre 0 et 5V, 0 et 1023 comme résultat de la mesure. Le capteur que l'on mesure donne des tensions entre A et B. Par exemple le maximum est 4V (analogRead donne la valeur A = 4/5*1024= 820). Le minimum est 2.8V (analogRead donne la valeur B = 2.8/5*1024= 574).
On veut afficher les valeurs lues sur le Oled qui a 64 points de haut, avec la coordonnée 0 en haut. Sur le Oled, le maximum de tension est a=0 et le minimum b=63. Pour avoir la coordonnée y qui correspond à une mesure Val, par exemple 702, il faut appliquer une règle de trois. La fonction Map() d'Arduino fait ceci en 54 microsecondes et utilise 400 bytes de code et une quinzaine de variables.



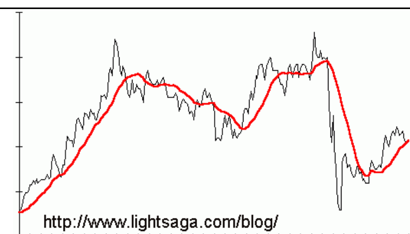
4.6 Simuler un oscilloscope

Un oscilloscope a 2 potentiomètres pour régler la hauteur de la trace, et 2 commutateurs pour régler l'amplification, par exemple pour avoir 4 gammes. Cela utilise 6 entrées et fait un joli projet de programmation. Dot() est utilisé pour une trace, et DDot() pour l'autre.

5. Moyenner

La mesure d'un capteur peut être bruitée. Une solution simple est de regrouper plusieurs mesures et calculer la moyenne. La durée entre les résultats est augmentée.

La moyenne glissante (running average) est beaucoup plus intéressante. On fait la moyenne sur les 4 ou 8 dernières mesures. Le lissage est amélioré, comme le montre la courbe ci-contre. Le signal filtré a naturellement un retard, moins gênant que des variations brusques.



Pour la programmation il faut initialiser une table de mesure, ce qui se fait dans le setup, et faire suivre chaque mesure par la fonction qui moyenne. Le setup remplit de préférence la table avec la mesure du capteur à l'appel. Si on initialise avec des zéros, ce qui est plus simple, il faut attendre 4 ou 8 mesures avant d'avoir rejoint les bonnes valeurs
S'il y a plusieurs capteurs il faut plusieurs tables, pointeurs circulaires et fonctions.

5.1 Algorithme

L'algorithme pour des données 8 bits (plus de précision n'est pas réaliste avec les capteurs grand public) travaille avec un tableau `taGlis[]` et une variable `toTable` qui totalise le contenu de la table. A chaque nouvelle mesure, on remplace la mesure la plus ancienne dans la table et on corrige le total. La mesure filtrée est donnée en divisant le total par la longueur de la table.

```
 #define LonGlis 8
byte taGlis[LonGlis];
byte ptTaGlis; int toTable;
void Glis {
  for (byte i=0; i<LonGlis; i++)  { taGlis[i]=0; }
  toTable=0;
}
byte Glis (byte dd) {
  toTable -= taGlis[ptTaGlis];  // on soustait la plus ancienne valeur
  toTable += dd;           //total à jour
  taGlis[ptTaGlis] = dd;  // table à jour
  return (toTable/LonGlis);
}
```

5.2 Test avec le terminal série

Il faut naturellement un programme de test. Celui-ci aide à comprendre l'algorithme en affichant sur le terminal.

```
//TestMoyGlis.ino avec terminal série
//Taper un chiffre de 0 - 7 dans la ligne du haut suivi d'un enter
#define LonGlis 8   // debut du fichier MoyGlis.h
byte taGlis[LonGlis];
byte ptTaGlis; int toTable;
void SetupGlis () {
  for (byte i=0; i<LonGlis; i++)  { taGlis[i]=0; }
  toTable=0;
}
byte Glis (byte dd) {
  toTable -= taGlis[ptTaGlis];  // on soustait la plus ancienne valeur
  toTable += dd;          //total à jour
  taGlis[ptTaGlis] = dd;  // table à jour
  if (++ptTaGlis==LonGlis) {ptTaGlis=0;}
  return (toTable/LonGlis);
}     // fin du fichier MoyGlis.h

void setup(void) {
  Serial.begin(9600);
  SetupGlis();
}
// Le terminal rend des car Ascii. Parse donnerait des valeurs
byte moy;
void loop () {
  if (Serial.available() > 0) {
    byte car = (Serial.read ())&0x07;   //0-7
    moy = Glis (car);
    Serial.print (car);Serial.print (" ");
    Serial.print (moy);Serial.print (" ");
    Serial.print (ptTaGlis);Serial.print (" tot ");
    Serial.println (toTable,HEX);
  }
}
```

5.3 Fichier MoyGlis.h

Le fichier MoyGlis.h à insérer est utilisable pour des variables 8 et 16 bits: moy8= Glis8(val8) et moy16=Glis16(val16). Le nombre de mesures est dans MoyGlis.h

5-4 Exemple avec le SR05

.On ajoute la librairie MoyGlis.h et son setup dans le programme `TestOledUsonScope.ino`.
On voit que ce "meccano" de fichier inclus est facile à utiliser.

```
//TestOledUsonGlis  170509 Lit un pot et affiche la moyenne
typedef uint8_t byte;
#include "OledI2Cbb.h"
#include "OledPix.h"
#include "Uson.h"
#include "MoyGlis.h"
void setup(){
  SetupI2C();
  SetupOled();
  SetupUson();
  SetupGlis16();
```

```
  }
int dist,moy;
byte x;
void loop(){
  dist = GetSonar();  // 0-8000
  LiCol(1,0); BigDec9999(dist);
  moy = Glis16 (dist);
  LiCol(1,60); BigDec9999(moy);
  Dot (x,dist/16); DDot (x,moy/16);
  if (x++==128) {x=0; Clear();}
 delay(100);
}
```

Il faut naturellement modifiez le délai et essayer différents obstacles et mouvements.

jdn 170601