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

Moyenne et moyenne glissante

Cette documentation s'adresse à des débutants. Le but est de bien faire comprendre l'utilité et la facilité d'utilisation de la moyenne glissante. Le résultat est un fichier inséré Moyenne.h aussi facile à utiliser qu'une fonction Arduino. Pour le programmeur un peu expérimenté qui a une application, la dernière page contient toute l'information utile pour moyenner des valeurs 8 et 16 bits. Adapter à d'autres types de données est trivial.

1. Moyenne

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.
Par exemple, on reçoit des valeurs 8 bits d'un capteur on les accumule. Toutes le 4 valeurs, on transmet la moyenne.
La fonction bloquante qui effectue 4 mesures équidistantes est la suivante. Les paramètres sont la période de mesure pp et la valeur mesurée mes , mesure si c'est une variable mise à jour par interruption, ou Mesure() si c'est une fonction qui donne la valeur, comme un analogRead(). Les deux sont déclarées en 16 bits, une durée en ms dèpasse facilement 256 et une mesure avec AnalogRead est 10 bits.

```
uint16_t Moyenne (uint16_t pp, uint16_t mes) {
  int total;
  for (uint8_t i=0; i<4; i++) { // sur 4 mesures
     total  += mes;
     delay(pp);
  }
  return (total/4) ;
}
```

Le total ne doit pas ici dépasser 16 bits. Si on additionne des mots de 10 bits, on peut en additionner 64.
Une division par 4 est nettement plus rapide qu'une division par 3 ou par 5, parce que c'est un décalage, exécuté en 2 instructions. On fera toujours la moyenne sur 2, 4, 8 valeurs pour économiser la mémoire et le temps calcul.

Le programme de test dépend de ce que l'on doit lire. Si on branche un potentiomètre sur A0, on peut afficher les valeurs sur le terminal série.

```
// TestMoyenne.ino
void setup() {
  }
int Moyenne (int pp,int mes) { // sur 4 mesures
  int total;
  for (byte i=0; i<4; i++) {
     total  += mes;
     delay(pp);
  }
  return (total/4) ;
}
byte Ldr;
void loop() {
  Ldr = Moyenne(10, analogRead(A0));
//...  if (Ldr > MaxLight) ...
}
```

2. Moyenne des valeurs d'une table

On a accumulé des valeurs dans une table et on veut la moyenne.

```
void setup()   {
  Serial.begin(9600);
}
byte table [] = {23,43,66,51,11,24,92,7,33,15};
longTable = sizeof (table);
unsigned int total = 0;
```

```
unsigned int moyenne;
byte cnt;
void loop () {
  for (int i=0; i<longTable; i++) {
    total += table [i] ;
  }
  moyenne = total/longTable ;
  if (total%longTable) >= longTable/2) { moyenne++}
  Serial.print (moyenne);
  while (1) {}
}
```

On a besoin de la longueur de la tables dans plusieurs instructions:
- dans la définitions de la table [] (le crochet peut rester vide, le C sait compter les éléments suivants),
- dans le nombre d'itérations dans la boucle for,
- dans la division.
- dans le calcul de l'arrondi

Avec l'opérateur `sizeof` facile à comprendre, le compilateur transfère la longueur de la table dans une variable, et le programme ne dépends plus de la longueur de la table. .

Cet exemple permet aussi de mentionner une fonctionnalité du C appelées conversion de types (*type casting)*: Dans l'instruction `total += table [i] ;` on ajoute un mot de 8 bits à un mot de 16 bits. Le compilateur doit convertir le mot de 8 bits `table [i]` en un mot de 16 bits pour pouvoir l'additionner à total. Dans ce cas, il le fait tout seul, mais parfois, en particulier avec les nombres flottants, il faut bien dire qu'il y a changement de type. Dans notre cas, certains compilateurs peuvent exiger d'écrire `total += (int)table [i] ;`

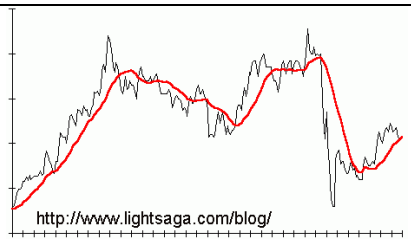
3 Moyenne non bloquante

Le cas normal est que la fonction qui lit le capteur donne une valeur moyennée. La fonction est appelée par interruption synchrone pour lire le capteur, mais la valeur de la variable globale résultat de la mesure n'est mise à jour que toutes les 4 ou 8 interruptions.

```
// TestLectureParInterruption.ino
void setup() {
  ISR (TIMER2_OVF_vect) {
    TCNT2 = 141;  //  58 us (256-141=58x2)
    GetMoyCaptA0();
  }
}
volatile int MoyCaptA0;  //Variable globale
int total; byte i=0;
void GetMoyCaptA0 () { // sur 4 mesures
    if (i++ < 4) {
      total += analogRead(A0)
  }
  else {i=0; MoyCaptA0 = total/4; }
}
void loop() {
//...  if (MoyCaptA0 > MaxLight) ...
}
```

En résumé, toutes les 60 microsecondes (ou une autre période), on lit le capteur, et toutes les 240 microsecondes, on modifie la valeur `MoyCaptA0`..

4 Moyenne glissante

La moyenne glissante ou mobile (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 les variations brusques non filtrées ou les escaliers d'une moyenne simple.	

Evidemment, à la première mesure, la moyenne ne peut pas se calculer sur des données précédentes. Le setup remplit la table avec une première mesure du capteur. Si on initialise la table avec des zéros, ce que le compilateur fait par défaut, il faut attendre 4 ou 8 mesures avant d'avoir rejoint les bonnes valeurs

4.1 Algorithme

L'algorithme travaille avec un tableau `taGlis[]` et une variable `total` 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. Le pointeur `ptTaglis` est un pointeur circulaire. Il pointe la plus ancienne mesure, que l'on soustrait du total, et pointe la nouvelle, que l'on ajoute au total et met dans la table. Ensuite, il se déplace et le total est divisé.

```
byte mesure;  // valeur fournie par le capteur
#define LonGlis 8
byte taGlis[LonGlis];  // 0.5% de précision avec une table 8 bits
byte ptTaGlis; int total;
void SetupGlis (byte dd) {
  for (byte i=0; i<LonGlis; i++)  {
    taGlis[i]= dd;
    total += dd;
  }
}
byte Glis (byte dd) {  // rend la moyenne (8 bits) après nouvelle donnée dd
  total -= taGlis[ptTaGlis];  // on soustrait la plus ancienne valeur
  total += dd;                // on ajoute la nouvelle
  taGlis[ptTaGlis] = dd;      // table à jour
  if (++ptTaGlis==LonGlis) {ptTaGlis=0;}  // incrémentation circulaire
  return (toTable/LonGlis);
}
```

Ce groupe d'instruction ne dépend pas de la nature de la mesure. On en fera un fichier inséré MoyGlis.h (section 4.3).

4.2 Test avec le terminal série

Ce programme de test aide à comprendre l'algorithme en affichant sur le terminal le contenu de la table. Le terminal série n'est pas pratiques pour entrer des valeurs. On se contente d'entrer les chiffres 0 à 9, code 48=0x30 à 57=0x39. Le résultat est la moyenne des codes ASCII introduits.

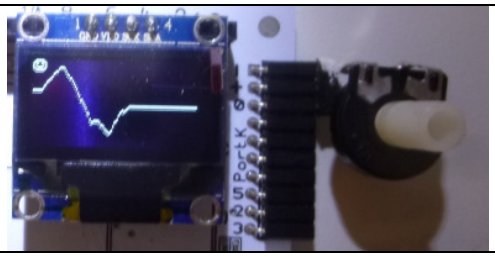
```
//TestMoyGlis.ino avec terminal série
//Taper un chiffre de 0 - 7 dans la ligne du haut suivi d'un Enter
#define LonGlis 8   // debut du fichier MoyGlis.h
byte taGlis[LonGlis];
byte ptTaGlis; int toTable;
void SetupGlis () {
  for (byte i=0; i<LonGlis; i++)  { taGlis[i]=0; }
  toTable=0;
}
byte Glis (byte dd) {
  toTable -= taGlis[ptTaGlis];
  toTable += dd;
  taGlis[ptTaGlis] = dd;
  if (++ptTaGlis==LonGlis) {ptTaGlis=0;}
  return (toTable/LonGlis);
}
void setup(void) {
  Serial.begin(9600);
  SetupGlis();
}
// Le terminal rend des car Ascii.
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);
  }
}
```

4.3 Fichier MoyGlis.h

Le fichier MoyGlis.h à insérer est utilisable pour des variables 8 et 16 bits:
Glis8(val8) et Glis16(val16) rendent la moyenne.
Le nombre de mesures est déclaré au début de MoyGlis.h

4.4 Exemple avec un potentiomètre

Glis8 et Glis16 sont testés avec un pot sur A0, avec affichage sur Oled (voir www.didel.com/Oled.pdf). En agissant sur le délai, on voit bien la moyenne rattraper la valeur lue. Le programme, comme les autres programmes de cette documentation, peut être chargé depuis www.didel.com/Moyennes.zip.



```
//TestOledPotGlis  170509 Lit un pot et affiche la moyenne
//\image:OledPot.bmp
typedef uint8_t byte;
#include "MoyGlis.h"
void setup(){
  SetupOledPix();
  SetupMoyGlis16();
  // pas besoin de setup pour le pot
}
int moy, pot; // variables globales
void loop(){
  pot = analogRead(A0);
  moy = Glis16 (pot);

  delay(100);
}
```

Avec un délai de 0.1s et une profondeur de 8, on voit que les variations brusque du potentiomètre sont prises en compte avec un retard de 0.5s

5 Interruption synchrone

Le capteur (ici un signal analogique sur A0) est lu régulièrement par interruption. La moyenne glissante est calculée chaque fois et la variable globale MoyA0 est mise à jour. Le programme principal utilise cette variable, sans savoir comment elle a été fabriquée.
Le principe de la programmation synchrone "LibX" (www.didel.com/LibX.pdf) est facile à comprendre: Travailler avec des interruptions est comme avoir un processeur supplémentaire avec son setup et son loop qui décide des tâches à exécuter à chaque interruption. Le fichier inclus `"Inter.h"` contient la fonction `SetupInter();` à mettre sans le setup et la fonction `ISR (TIMER2_OVF_vect) {}` qui liste les tâches à exécuter. Ces tâches doivent toutes sêxécuter en moins de 10microsecondes, mais sont appelées toutes les 60 microsecondes pour avoir une bonne réponse "temps réel".
l'application.
Notre exemple lit un potentiomètre. Arduino l'installe par défaut, il n'y a pas de setup explicite. Ce potentiomètre est lu par interruption, et moyenné en parallèle. Le fichier InterGetA0.h est donc le suivant:

```
//InterGetA0.h
void  SetupInter(){ // initialiser le timer
  TCCR2A = 0; //default
  TCCR2B = 0b00000010;  // clk/8
  TIMSK2 = 0b00000001;  // TOIE2
  sei();
}
// déclarer les variables locales "volatile"
ISR (TIMER2_OVF_vect) {
   TCNT2 = 141;  //  58 us (256-141=58x2)
   GetMoyA0();
}
```

Le fichier inséré qui offre les deux fonctions moyenne glissante est

```
// MoyGlis.h
#define LonGlis 8
uint8_t taGlis[LonGlis];
int taGlis16[LonGlis];
uint8_t ptTaGlis; uint16_t total; uint32_t toTa32;
void SetupMoyGlis () {
  for (uint8_t i=0; i<LonGlis; i++)  { taGlis[i]=0; }
  total=0;
}
byte Glis8 (byte dd) {
  total -= taGlis[ptTaGlis];  // on soustait la plus ancienne valeur
  total += dd;          //total à jour
  taGlis[ptTaGlis] = dd;  // table à jour
  if (++ptTaGlis==LonGlis) {ptTaGlis=0;}
```

```
  return (total /LonGlis);
}

void SetupGlis16 () {
  for (uint8_t i=0; i<LonGlis; i++)  { taGlis16[i]=0; }
  toTa32=0;
}
int Glis16 (int dd) {
  toTa32 -= taGlis16[ptTaGlis];  // on soustait la plus ancienne valeur
  toTa32 += dd;          //total à jour
  taGlis16[ptTaGlis] = dd;  // table à jour
  if (++ptTaGlis==LonGlis) {ptTaGlis=0;}
  return (toTa32/LonGlis);
}
```

En ayant bien préparé et compris ces librairies, le programme est très simple. Le problème que l'on a parfois en appelant des fichier inclus est de respecter l'ordre voulu par le compilateur. Une fonction ou variable doit être définie avant d'être utilisée. Les variables globales sont donc déclarées en premier.

//TestMoyenneGlissanteInter.ino

```
uint_16 MoyA0;  //Variables globales
#include "Inter.h
#include "MoyGlis.h"
#include "OledPix.h"

void setup() {
  SetupMoyGlis();
  SetupInter();
    // A0 ne doit pas être initialisé
}
uint8_t x,y1,y2; // on veut le graphique de la valeur et sa moyenne
void loop() {
  uint16_t val=GetMoy16(analogRead(A0));  <----- ko
  delay(100);
}
```


A noter que s'il faut faire la moyenne glissante d'un capteur ultrason, on ne peut pas utiliser la fonction Arduino Pulsein() qui est bloquante. La fonction GetUson() de la librairie X doit être utilisée, et est facilement complétée par une moyenne glissante qui filtre les anomalies de lecture (voir www.didel.com/OledUson.pdf).

jdn 170527