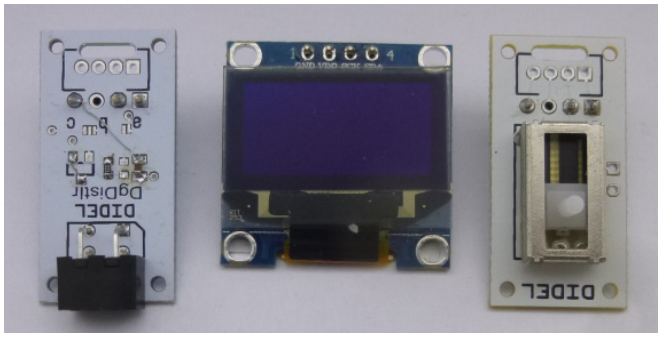
CH-1092 Belmont, Switzerland
Tel +41 21 728-6156, Fax 728-6157
Email info@didel.com, www.didel.com

www.didel.com/OledDistIr.pdf

Oled et capteur de distance IR
Tutorial associé au kit "OledCapteurs"



Introduction

Apprenons à programmer un capteur de distance utilisant un réflecteur infrarouge, et profitons de l'affichage Oled pour visualiser ce que le capteur voit. Ce document procède par étape, il est facile à suivre si vous avez apprécié le MOOC EPFL et les exercices avec le LearnCbot. Si vos connaissance se bornent au "digitalWrite", cette instruction géniale qui fait penser au vélo d'enfant avec 2 roues d'équilibrage, vous aurez parfois de la peine, comme pour le vélo!

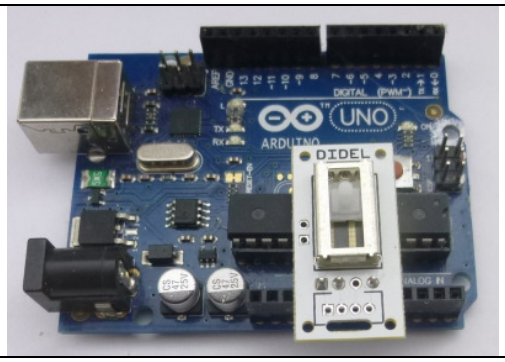
Lire un potentiomètre

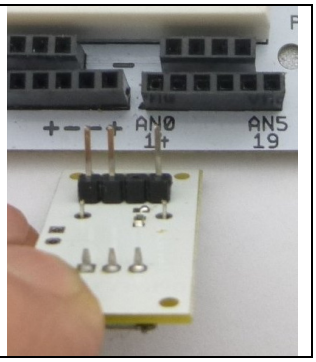
Chargeons le programme le plus simple, `Pot.ino` que vous trouvez comme tous les autres programmes de cette notice sous www.didel.com/OledDistIr.zip

Arduino connait la fonction `analogRead(A0)` qui mesure le rapport entre la tension sur la pin 14/A0 et la tension d'alimentation. La valeur lue vaut entre 0 et 1023 (12 bits).
Le librairie pour imprimer est chargée par défaut, mais il faut spécifier le "bit rate" dans le set-up.
Toutes les 500 ms on lit et on affiche.

```
//Pot.ino| Le programme le plus simple
void setup() {
  Serial.begin(9600);
}
int val;
void loop()  {
  val = analogRead (A0);
  Serial.println (val) ;
delay (500);
}
```

Le potentiomètre du kit facilite la cablage. Les extrémités du pot sont branchées au 0 et 5V (avec une résistance de protection) et le point milieu est envoyé sur la pin 14/A0. Vous pouvez câbler sur un breadboard. C'est plus simple et fiable avec notre pot, comme sur la photo.

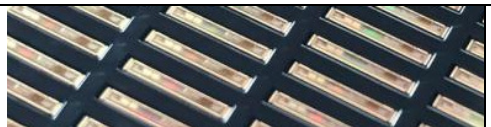




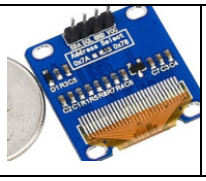
A partir de là, on pourrait jouer avec le terminal, mettre des commentaires, chercher la doc pour faire apparaître un graphique. On peut regarder les valeurs que donnent les capteurs qui ont une sortie analogique, il y en a beaucoup. Si la valeur ne change pas trop vite, le terminal est utilisable, mais quand votre application deviendra autonome, plouf, plus de terminal !

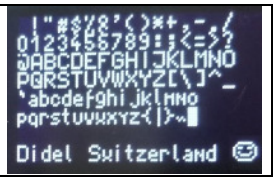
L'afficheur OLED

La technologie est admirable.
Courant de 1mA, tension de 2.5 à 5V. Mais il faut 100ms pour écrire tout l'écran.



La puce fait 1,1x6 mm et a 110 pads



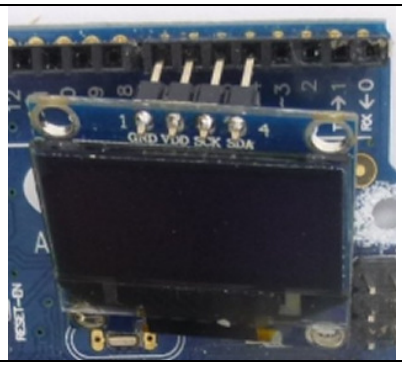


La librairie n'est pas plus compliquée que celle du terminal, et on a maintenant deux options. Faire comme tout le monde avec un breadboard et la librairie I2C de Adafruit, ou continuer à lire ce texte.
Premier avantage de notre approche, le Oled se branche sur 4 pins, du port D par exemple.
Et l'alimentation direz-vous? Que faut-il pour alimenter un Oled? Très peu! Dans ce cas, une pin programmée à 0, c'est un assez bon Gnd, et une pin programmés à 1, c'est un assez bon 5V, vous savez allumer des LEDs qui demandent 20mA.
Mais I2C demande 2 pins spéciales? Oui si vous voulez utiliser les registres spécialisés de l'AVR328, via la librairie Wire ou une autre plus efficace. I2C est facile à programmer en "bitbang" et de plus le Oled ne demande que des ordres I2C très simples.
On va charger deux librairies, une librairie I2Cbb.h et une librairies OledPix.h.
Mais c'est du C simple avec quelques contraintes temps réel. Tous les détails sous https://github.com/nicoud/Oled.
Ces librairies ne s'installent pas. Ce sont des définitions et fonctions regroupées dans des fichiers "inclus" qui en fait font partie du programme. Arduino facilite leur édition. On verra que cela permet une programmation très modulaire qui facilite la mise au point.

Câblage de l'OLED et définitions

Si le SSD1306 est câblé sur les pins 4,5,6,7, il faut dire que
pin7 = Gnd, pin6=+5V, pin5=SCL, pin4=SDA
out out in in
En Arduino:

```
pinMode (7,1); digitalWrite (7,0);
pinMode (6,1); digitalWrite (6,1);
pinMode (5,0);
pinMode (4,0);
```



Les fonctions Arduino sont trop lentes pour les transferts. On utilisera donc un autre ensemble de définitions, en agissant directement sur le port avec des opérations logiques. Le fichier OledI2Cbb.h contient ces définitions et les fonctions de transfert I2C qui seront appelées par les fonctions OledPix. La librairie plus complète I2Cbb.h est documentée sous www.didel.com/I2Cbb.pdf

Le #define

Vous n'êtes peut-être pas à l'aise avec le #define. Arduino a décidé de l'ignorer, estimant que vous êtes trop bête pour utiliser à bon escient ! A la base, c'est un simple dictionnaire:
#define - un mot - une phrase – un retour de ligne: le mot est remplacé par la phrase
`#define PinDistIr 14` Le compilateur voit 14 quant vous avez écrit `PinDistIr`
Arduino vous dit de déclarer `int PinDistIr = 14;` ce qui gaspille 2 positions mémoire

```
#define White() digitalWrite (pinRed,1); digitalWrite (pinGreen,1); digitalWrite (pinBlue,1)
```

C'est un peu comme une fonction. `White();` est remplacé par les 3 instructions.
Dans le programme `White()` est une instruction que l'on doit terminer par un ; . Il ne faut pas mettre ce ; à la fin du `#define`. Pas si difficile à comprendre!

Librairie OledPix

Ce qui nous intéresse, c'est d'afficher des nombres comme avec le terminal, et on va surtout apprécier la facilité pour faire des oscillogrammes. Avec le programme terminal, on dit print(xx); et xx est analysé pour savoir ce qu'il faut afficher. C'est beaucoup d'instructions! Avec un Oled ou autre affichage graphique, on peut afficher une lettre (`Car(code);`), un texte (`Text(nom)`), un nombre dans différentes bases. Comme les caractères sont petit sur le SSD1306, une option "big" est nécessaire.
On a peu de place sur l'écran, et on veut donner la mémoire en priorité à l'application, on a donc limité les fonctions, mais ces fonctions sont simples, et c'est un joli exercice d'ajouter celles qui manquent.

Résumons ce que vous allez utiliser, la doc complète se trouve sous www.didel.com/Oled1306.pdf.

Pour les caractères, l'écran a 8 lignes. Les majuscules ont 6 pixels de large, les minuscules et chiffres 5 pixels. L'écran graphique de 128x64 pixels a son origine en haut à gauche.
Les fonctions à disposition sont

LiCol(li,co);	. Place le pointeur ligne **li** (0 à 7) et en colonne **co** (0 à 127, mais il faut la place pour finir)
Bin8(v8);	. Affiche la variable 8 bits (byte, char, int8_t, uint8_t) en binaire
Hex8(v8);	. Affiche la variable 8 bits (byte, char, int8_t, uint8_t) en binaire
Hex16(v8);	. Affiche la variable 16 bits (int, int16_t uint6_t) en binaire
Dec8(v8);	. Affiche la variable 8 bits (byte, char, int8_t, uint8_t) en binaire
Dec9999(v16);	. Affiche la variable 16 bits (int, int16_t uint6_t) en décimal, limite 0x270F=9999 note: Dec16 aurait utilisé 5 chiffres, et c'est rare que l'on doive afficher des nombres plus grands que 9999 en interagissant avec des capteurs.

En double taille (utilise la ligne en-dessus, ne pas utiliser la ligne 0), on a:

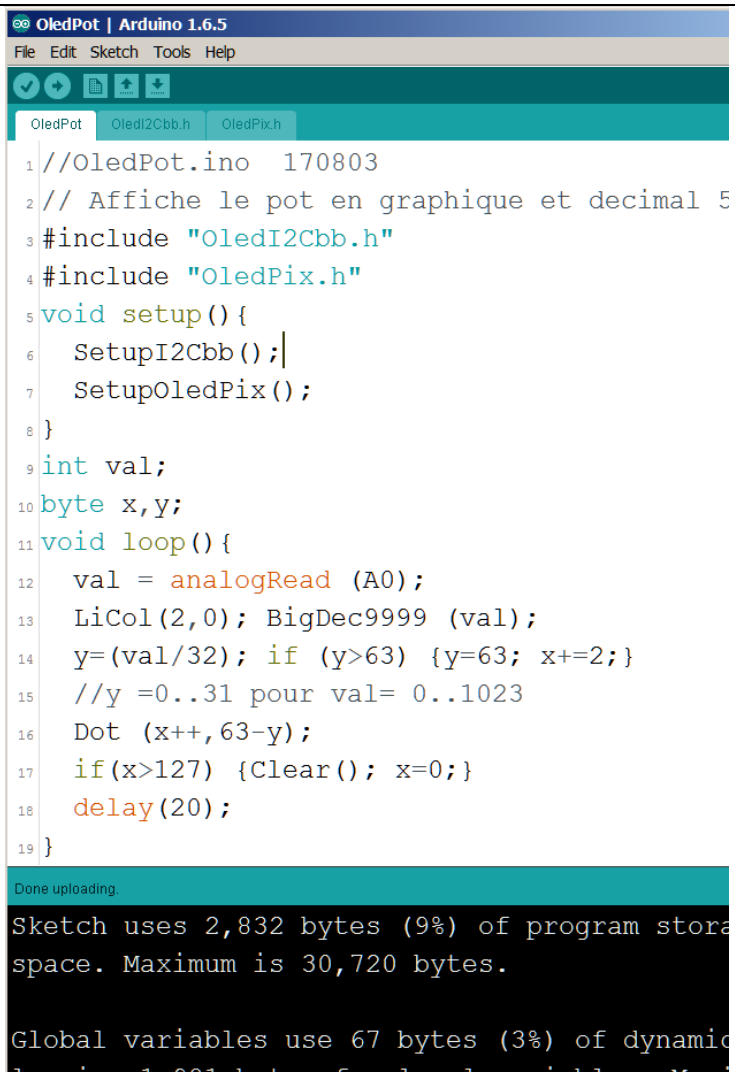
Big(); (ou BigCar) BigBin8(); BigHex8(); BigHex16(); BigDec8(); BigDec9999();

Pour les textes, il faut définir une chaîne, et l'appeler:
byte hello[] = {"Hello"}; Licol(li,co); Text(hello);

C'est facile et amusant de programmer des lutins. Vous pouvez essayer `Sprite(smile);`

Affichons la valeur du potentiomètre

Chargeons le programme OledPot.ino
On voit les deux #include et les Setup correspondants.
Par rapport au programme Pot.ino, on voit les variables x,y pour le graphique.
Pour afficher la valeur lue, il faut préciser la position dans l'écran. Il y a 8 lignes pour les caractères, début selon 128 colonnes.
On doit afficher une variable 12 bits. On pourrait l'afficher en 8 bits en perdant un peu de précision `Dec (val/4);` ou `Hex(val/4)` ou `Bin(val/4).`
Essayez, `BigDec()etc.` aussi.
Notre astuce est la fonction `Dec9999()` qui limite l'affichage à une valeur qui n'utilise que 4 chiffres. C'est rare et franchement inutile d'avoir des nombres plus grands ou plus précis en robotique amateur.
Pour afficher un point (`Dot(x,y);`), l'origine est en haut. La valeur à afficher, de 0 à 1023 est comprimée en 0-31 pour afficher la mesure en haut.
Toutes les 20ms, on incrémente x. Si x est en fin d'écran, on efface et recommence.
Petite astuce, si la valeur x dépasse 63, on sature, mais en plus on affiche ces valeurs saturées en pointillé.

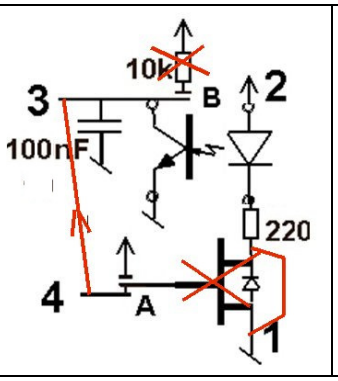


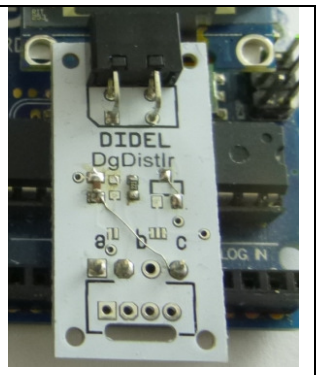
La taille du programme n'est pas négligeable (les caractères Big sont compliqués à calculer). Le même programme avec la librairie Adafruit doit faire 12k, plus 1.5k de variables (et 3 pages de listage).
On voit que l'on a un outil utile et facile pour afficher beaucoup de choses qui varient rapidement sur l'écran. Ecrire un Dot(x,y) prend 1.2 ms. Hex(v8) prend 2ms et naturellement BigDec9999(v16); est le plus long avec 3.4 ms. Avec la librairie Adafruit standard, chaque mise à jour de l'écran prend 100 ms.

Capteur de distance IR

Les capteurs en réflexion sont bien connus, mais ils sont rarement bien utilisés pour mesurer une distance. Même pour suivre une piste, les exemples donnés résultent d'un tâtonnement avec réglage d'un potentiomètre. Les schémas que vous trouvez utilisent un potentiomètre en diviseur de tension avec le phototransistor. C'est facile, on lit le signal analogique, on règle le potentiomètre et on a des bonnes valeurs autour de la valeur 512 donnée par la fonction analogRead(). Mais si la lumière reçue varie d'un facteur 10, on se trouve dans des valeurs proches de zéro ou du max et la précision est catastrophique. C'est bien expliqué sous www.didel.com/doc/sens/DocIr.pdf

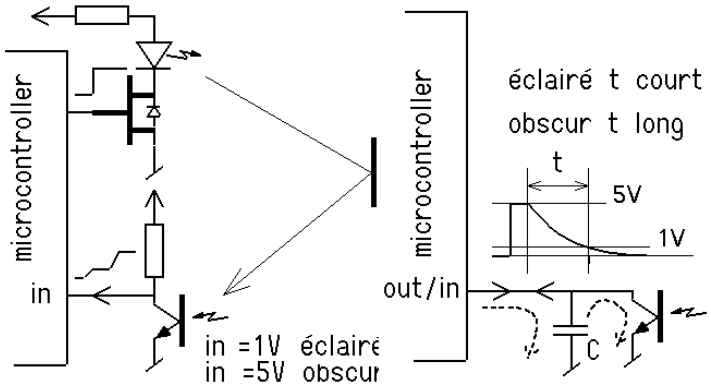
Les capteurs IR comportent une diode IR et un phototransistor sensible à la même longueur d'onde. Ceci permet d'être un peu moins dépendant de la lumière ambiante.
Le module DgDistIr a été modifié pour être installé aussi facilement que le potentiomètre.
Le module original permet de pulser la diode IR, ce qui économise de la puissance et permet une mesure sans éclairage, donc offre la possibilité de soustraire la lumière ambiante.





Mesure par décharge d'un condensateur

La solution élégante pour mesurer une résistance variable est de charger un condensateur en parallèle avec la résistance et de mesurer le temps de décharge. La pin du microcontrôleur est d'abord programmée en sortie avec l'état 1. En quelques microsecondes le condensateur est chargé. On commute ensuite la pin en entrée, et on mesure le temps de décharge, c'est à dire le temps pendant lequel le signal est à 1. Vers 1.5V, le signal passe à zéro.



La valeur du condensateur est telle que le processeur compte avec suffisamment de précision pendant un temps assez court pour que le robot ne se déplace pas trop entre deux mesures. Dans l'obscurité, le condensateur se décharge très lentement, on va donc limiter la valeur à 256, ce qui est bien assez précis avec une bonne linéarité. Si on compte toutes les 100microsecondes, la durée de la mesure est de 25 ms. Augmenter le condensateur augmente la distance, mais la lumière parasite gêne. L'effet de la période d'échantillonnage et de la taille du compteur est facile à vérifier.

Définitions et fonction bloquante

On doit donc commander un pin tantôt en sortie, avec l'état 1 (mode "charge"), et tantôt en entrée, en évitant absolument d'avoir une résistance pull-up ou pull-down qui perturbe la décharge (mode "décharge" ou "mesure").

```
Déclarations en C portable
#define bDist  0 //PORTC bit 0  Arduino pin 14/A0
#define Charge DDRC !=1<<bDist; PORTC |= 1<<bDist
#define Mesure DDRC &=~(1<<bDist; PORTC &=~(1<<bDist)
#define CapaHigh PINC & 1<<bDist

void SetupDistIr () {
   DirLedIr;
   DirMes;
}
```

```
Déclarations Arduino
#define Dist  14  //Arduino pin 14/A0
#define Charge pinMode(Dist,1); pinMode(Dist,1)
#define Mesure pinMode(Dist,0); pinMode(Dist,0)
#define CapaGHigh digitalRead (Dist)

void SetupDistIr () {
   pinMode (LedIr,1);
   DirMes;
}
```

Pour la mesure, on charge pendant 100 us, puis on commute en entrée, et on lit la valeur qui décroit exponentiellement. Tant que le seuil est supérieur à l'état 1 (environ 1.2Và 5V, 0.9V à 3V) on compte dans une boucle qui limite la durée.

```
byte GetDist () {
  byte cnt=0;
```

```
        while (++cnt<255 && CapaHigh) {
           delayMicroseconds(200);
         }  return(cnt);
      }
```

Ce programme est bloquant, pendant 255x200 µs seules les interruptions sont exécutées. Elles vont perturber le comptage. On verra plus loin comment faire les mesures par interruption.

Programme de test

Le programme mesure en continu et affiche les valeurs sur le terminal.

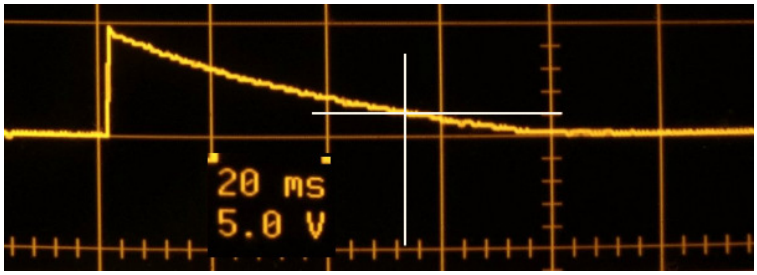
```
//DistIR0.ino  2794b 211v
// Connecté sur pins 14
#define Charge() pinMode (14,OUTPUT); \
                 digitalWrite (14,1)
#define Mesure() pinMode (14,INPUT); \
                 digitalWrite (14,0)
#define CapaHigh  digitalRead (14)

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

byte dist;
byte GetDist() {
   byte cnt=0;
   Charge(); // precharge
   delayMicroseconds (100) ;
   Mesure();
   while (++cnt<255 && CapaHigh) {
     delayMicroseconds(200);
   }
   return (cnt);
}
```

```
void loop() {
  dist = GetDist();
    Serial.print("Distance ");
    Serial.println(dist);
  delay(100);
}
```

Note: Le \ permet de continuer un #define sur une nouvelle ligne



Le programme DistIr1.ino est "identique". Les définitions sont en C et non pas en Arduino. La vitesse de comptage est un peu plus élevée.

Ce test de la fonction distance fait, il faut l'encapsuler dans un fichier inclus DistIr.h

```
//DistIR2.ino 2270b 211v
//Connecté sur pins 14 PORTC bit 0
#include "DistIr.h"
void setup() {
  SetupDistIr();
  Serial.begin(9600);
}
byte dist;
void loop() {
  dist = GetDist();
   Serial.print("Distance ");
   Serial.println(dist);
   delay(100);
}
```

La fonction GetDist dure 25ms max.
L'affichage (~15car) dure 15ms
Le délai supplémentaire fixe la vitesse de déroulement sur l'écran

```
//DistIr.h
#include <Arduino.h>
// Connecté sur pins 14 PORTC bit 0
#define bDist  0 //PORTC  pin14
#define Charge() DDRC |= 1<<bDist; \
                 PORTC|= 1<<bDist
#define Mesure() DDRC  &= ~(1<<bDist); \
                 PORTC &= ~(1<<bDist)
#define CapaHigh  PINC & (1<<bDist)
void SetupDistIr() {
  Mesure();
}
byte GetDist() {
   byte cnt=0;
   Charge();
   delayMicroseconds (100) ;
   Mesure();
   while (++cnt<255 && CapaHigh) {
     delayMicroseconds(200);
   }
}
```

Résumons: La vitesse de décharge dépend de la capacité et du capteur. La durée max mesurable dépend de la lumière ambiante. Un néon la fait varier à 50 Hz.
La période d'échantillonnage agit sur les valeurs mesurées – expérimentez !

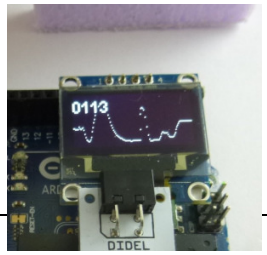
Afficher sur le Oled

C'est trivial, maintenant que la fonction `GetDist()` est testée et encapsulée, d'afficher sur le Oled plutôt que sur le terminal. On a fait l'exercice pour le potentiomètre.

```
//DistIrOled.ino 2794b 64v
//IR connecté sur pin 14 PORTC bit 0
// Oled sur Arduino pins 4 5 6 7
```

```
byte dist;
byte x,y;
void loop() {
```

```
#include "DistIr.h"
#include "OledI2Cbb.h"
#include "OledPix.h"
void setup() {
  SetupDistIr();
  SetupI2Cbb();
  SetupOledPix();
}
```




```
  dist = GetDist();
  LiCol(2,0); BigDec9999 (dist);
  y=(dist/4); if (y>63) {y=63; x+=2;}
  //y 0..63, pour dist 0..255
  Dot (x++,63-y);
  if(x>127) {Clear(); x=0;}
  delay(20);
}
```

Interruptions

Déclencher régulièrement une interruption pour faire une mesure nécessite un timer. Ce timer peut gérer toutes nos tâches dans un robot, c'est le concept LibX, qui s'oppose à la solution traditionelle multi-interruptions avec ses problèmes de priorités, et de temps de réponse.
Le timer2, bien expliqué sous www.didel.com/coursera/LC5.pdf crée une interruption toutes les 200 microsecondes. C'est ce qui permettra de compter la durée de la décharge. Mais il y a des opérations à faire avant et après. Une machine d'état s'impose. Dans un premier état, on charge le condensateur, dans le 2e on compte et teste la fin, dans le 3e on met à jour la variable globale distance.
Détaillons les instructions de cette "state machine"; savoir programmer en C, c'est pour beaucoup savoir écrire une machine d'état.

```
enum {Cha,Mes,Sav } etatIr = Cha;
```
`enum {x,y,.. }` donne à x,y, .. les valeurs 0,1,..
`etatIr = Cha;` La variable du switch case et sa première valeur. Elle n'a pas de type.

```
volatile byte cnt;
```
"volatile" doit être spécifié pour les variables utilisées dans une routine d'interruption

```
void GetIrDist () {
```
Cette fonction doit être appelée toutes les 60-100 µs, par une boucle du programme principal ou par une interruption

```
  switch (etatIr) {
    case Cha:
```
Si `etatIr` vaut `Cha` (=0) les instructions qui suivent sont exécutées

```
      Charge(); etatIr = Mes;
      break;
```
`break;` fait aller à `//end switch.` `etatIr` a été modifié, dans 100 us on exécutera les instructions de `case Mes:` etc..

```
    case Mes:
      Mesure();
      if (i++>255) {etatIr = Sav; break;}  // on quitte
      if (CapaHigh) { cnt++; }  // on compte
      break;
    case Sav:
      distIr = cnt; // mise à jour de la variable globale
      etatIr = Cha;
      break;
  } // end switch
}  // end function
```

L'interruption du timer2 doit être configurée selon la doc AVR328

```
    void SetupInter() {  // initialisation
       cli();
      TCCR2A = 0; //default
      TCCR2B = 0b00000011;  // clk/8 2us
      TIMSK2 = 0b00000001;  // TOIE2
      sei();
    }
```

La routine d'interruption réinitialise le timer et appelle notre fonction qui met à jour `distIr`.

```
     // Inter.h tous les inter
    ISR (TIMER2_OVF_vect) {
       TCNT2 = 156;  // 256-(200/2)
       GetMesIr();
    }
```

Evidemment, un fichier inclus va contenir ce qui concerne cette interruption.
Avec l'interruption toutes les 200 us, les transferts vers l'Oled seront "saucissonnés" de quelques microsecondes toutes les 200 microsecondes. Est-ce acceptable? Les spécifications du SSD1306 ne donnent aucune durée max pour les impulsions Sck et Sda.
Testons. Il y a 2 éléments nouveaux. L'interruption et la machine d'état. Mettons en route les interruptions, sans appeler `GetMesIr();` mais en ajoutant un `delayMicroseconds(30);` qui va bloquer le programme principal 50% du temps. On voit effectivement le pointeur graphique qui se déplace plus lentement (la valeur 0 est affichée puisque la distance n'est pas mesurée).
On peut alors ajouter la machine d'état. Si cela se bloque ou que la distance est incorrecte, on relit, on simplifie et reconstruit. Ce qui est important pour limiter le temps de mise au point c'est que l'erreur ne puisse provenir que d'un seul module bien localisé.
L'emplacement de la déclaration des variables est délicat. Les variables locales sont naturellement déclarées dans leur fichier lib.h. Les variables globales doivent être déclarées avant d'être utilisées. On peut hésiter à les mettre en en-tête de programme ou dans les modules lib.h. L'ordre dans lequel les #include sont déclarés doit parfois être revu.
A noter que le premier fichier inclus doit contenir la ligne #include <Arduino.h>. Il semble que cet include est ajouté par le setup (); Il déclare les noms des ports et registres timer de l'AVR328. Avec les fichiers inclus, le setup apparaît après.

```
//DistIrOled.ino 2918b 68v
// IR connecté sur pins 14 PORTC bit 0
// Oled sur Arduino pins 4 5 6 7
volatile byte distIr;
#include "DistIrInter.h"
#include "OledI2Cbb.h"
#include "OledPix.h"
#include "Inter.h"
void setup() {
  SetupDistIr();
  SetupI2Cbb();
  SetupOledPix();
  SetupInter();
}
byte x,y;
void loop() {
  LiCol(2,0); BigDec9999 (distIr);
  y=(distIr/4); if (y>63) {y=63; x+=2;}
  //y 0..63, pour dist 0..255
  Dot (x++,63-y);
  if (x>127) {Clear(); x=0;}
  delay(20);
}
- - - - - - -
// Inter.h Timer2
ISR (TIMER2_OVF_vect) {
 TCNT2 = 141;  //  60 us
 GetIrDist();
}
void SetupInter() {  // initialisation
  TCCR2A = 0; //default
  TCCR2B = 0b00000010;  // clk/8
  TIMSK2 = 0b00000001;  // TOIE2
  sei();
}
```

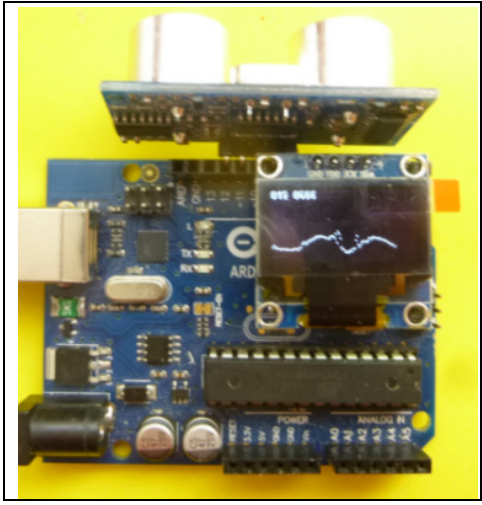
```
//DistIrInter.h
#include <Arduino.h>
// Connecté sur pins 14 PORTC bit 0
#define bDist 0 //PORTC  pin14
#define Charge() DDRC  |= 1<<bDist; PORTC
|= 1<<bDist
#define Mesure() DDRC  &= ~(1<<bDist);
PORTC &= ~(1<<bDist)
#define CapaHigh  PINC  & 1<<bDist
void SetupDistIr() {
  Mesure();
}
enum {Cha,Mes,Sav } etatIr = Cha;
volatile byte cnt, i;
void GetIrDist() {
  switch (etatIr) {
    case Cha:
      Charge(); cnt=0; i=0;
      etatIr = Mes;
      break;
    case Mes:
      Mesure();
      Mesure();
      if (CapaHigh) {
        cnt++;   // on compte
      } else {
        etatIr = Sav; // Low, on sauvera
      }
      if (cnt==255) {etatIr = Sav;}
      break;
    case Sav:
      distIr = cnt;
      etatIr = Cha; // on recommence
      break;
  } // end switch
}  // end GetIrDist
```

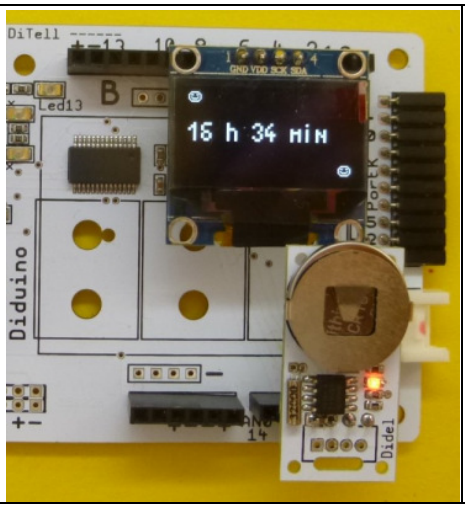
Autres capteurs

Les documents cités ci-dessous sont antérieurs à ce document et étudient d'autres capteurs en utilisant l'affichage de leurs paramètres sur un Oled. Il peut y avoir quelques différences de notations.

www.didel.com/Oled1306.pdf (SSD1306 avec ses choix de librairies)
www.didel.com/OledUson.pdf (capteur de distance à ultrasons)

www.didel.com/OledDS1307.pdf (circuit horloge avec batterie)
www.didel.com/OledDHT22.pdf (capteur de température et humidité)
www.didel.com/xbot/LibXDist2Ir.pdf





jdn 170805