L'univers Arduino

Partie II : Réalisation d'un prototype à base d'Arduino

Par f-leb

Date de publication : 25 septembre 2013

Dernière mise à jour : 25 septembre 2013

TOUT PUBLIC

Véritable mini-ordinateur au succès planétaire, traitant les données provenant de composants et capteurs divers (capteur de température, luminosité, mouvement ou boutons-poussoirs, etc.) et communiquant des ordres pour allumer des lampes ou actionner des moteurs électriques, la carte électronique Arduino permet de créer et prototyper de véritables objets numériques interagissant avec le milieu extérieur.

L'environnement de programmation qui l'accompagne propose un IDE et un langage basé sur les langages C / C++.

La communauté libre du monde Arduino contribue largement à diffuser les ressources permettant la création d'objets numériques à moindre coût et accessibles à toutes personnes motivées ayant même des connaissances modestes dans les domaines de l'informatique et l'électronique.

I - À qui s'adresse ce tutoriel ?

Il s'adresse aux débutants en programmation des microcontrôleurs voulant rejoindre les adeptes du mouvement *Do It Yourself* dans le domaine du *Physical Computing* (1) . Évidemment, en grand écumeur des forums Developpez.net, la programmation en général ne vous effraie pas. Vous aimeriez juste profiter de votre métier, votre passion pour la programmation, pour interagir avec le monde physique, piloter des robots, automatiser la montée/descente de vos volets déroulants en fonction de l'ensoleillement, construire votre propre station météo et déclencher des alarmes selon des seuils de température, etc.

Seulement vos connaissances en physique/électricité/électronique remontent à vos années lycée, une éternité… Quant à souder vous-même des composants électroniques, n'en parlons pas… Évidemment vous serez obligé de vous y mettre si votre projet doit prendre de l'ampleur, mais pas tout de suite.

Vous avez déjà entendu parler de la plateforme *Arduino*, peut-être même acheté votre *Arduino Uno* avec quelques composants de base (plaque de câblage rapide avec quelques fils, LED, résistances…) et programmé votre premier « Hello World » (programme qui consiste à faire clignoter la LED intégrée à la carte via la broche n°13). Vous avez peut-être même déjà commencé à combler vos lacunes en **électronique de base**, suivi quelques tutoriels pour allumer une, deux puis trois LED voire plus. Classique et indispensable pour débuter…

Ce tutoriel reprend un peu tout cela notamment dans la **première partie**, mais il propose également d'aller un peu plus loin avec l'utilisation de capteurs évolués, la découverte des cartes d'interface (ou *shields*) et de bibliothèques tierces. Le programme est vaste, mais les ressources proposées vous permettront de vous lancer dans des projets plus ambitieux en vous tenant le plus possible à l'écart du fer à souder.

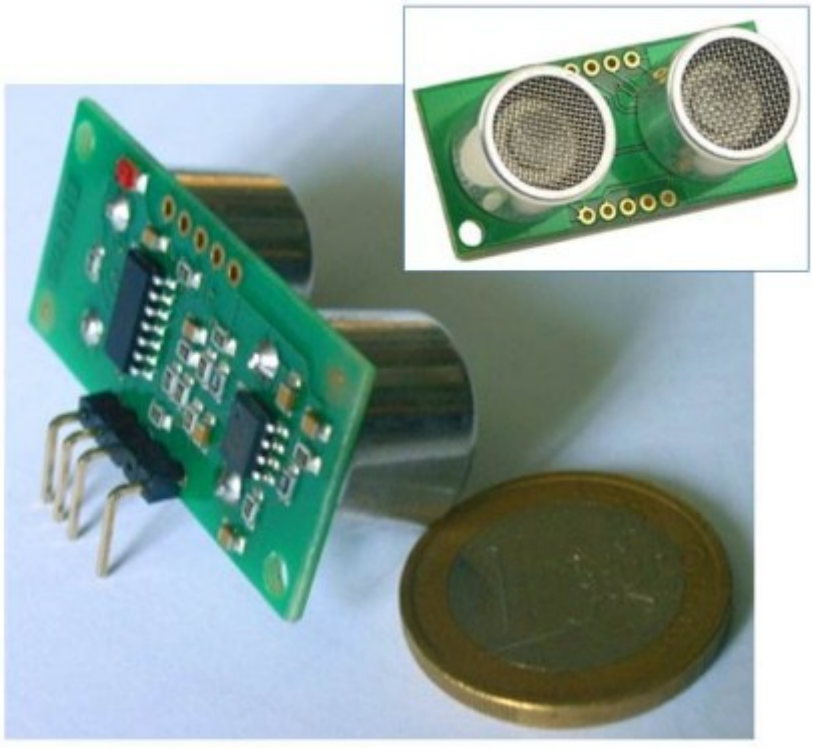
L'objectif reste le même que celui de la communauté *Arduino*, découvrir et s'amuser…

II - Objectifs et matériels utilisés

Pour **notre futur robot roulant**, on souhaite détecter la proximité d'un obstacle devant celui-ci afin d'agir en conséquence (marche arrière, procédure de contournement d'obstacle, etc.).

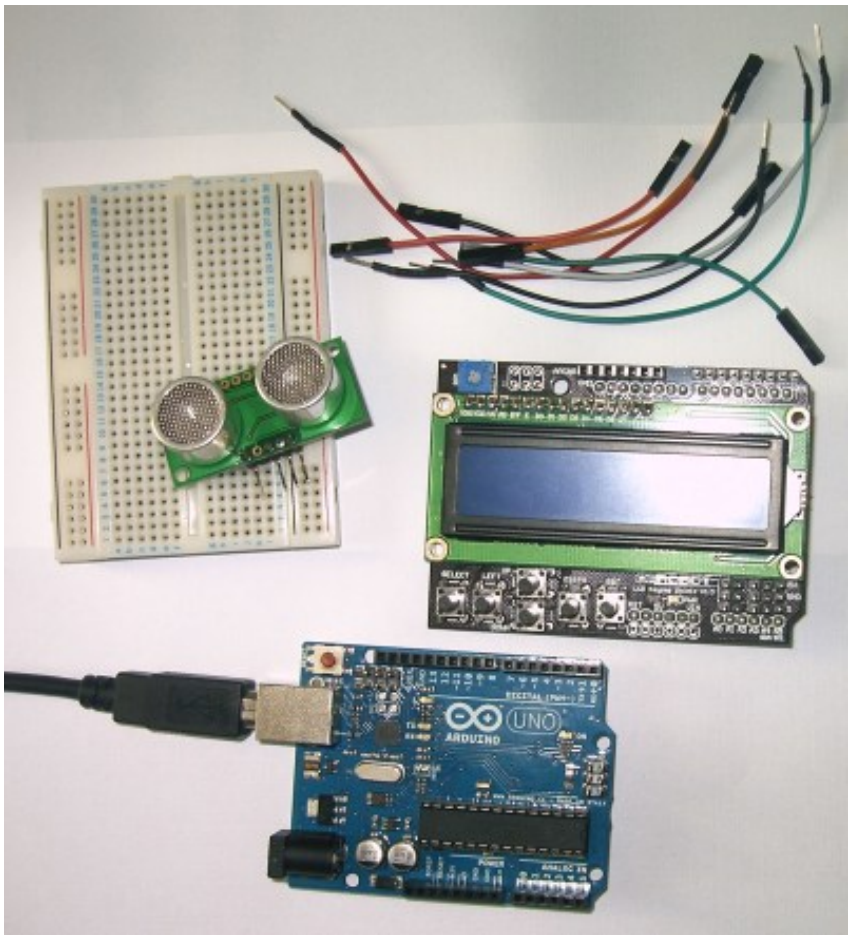
Afin de mesurer la distance d'un obstacle, mon choix s'est porté vers le capteur à ultrasons de la société **Robot Electronics**, le **SRF-05**. Ce capteur (une vingtaine d'euros) est d'usage courant pour la petite robotique et s'interface facilement avec la carte Arduino.

(1) En gros et au sens large, une discipline qui marie l'informatique, la physique et l'électronique, s'intéressant aux systèmes physiques interactifs qui utilisent des logiciels et des matériels s'interfaçant avec des capteurs, des actionneurs électriques, etc.



Afin de faire quelques essais du dispositif « sur le terrain » et avoir un retour des mesures, notre première maquette à base d'Arduino sera équipée du **LCD Keypad Shield**. Ce **shield**, avec son écran LCD rétroéclairé bleu et blanc (deux lignes de 16 caractères) et ses cinq boutons-poussoirs vous fera débourser une quinzaine d'euros supplémentaires. Vous pouvez bien entendu choisir un *shield* analogue d'un autre fabricant. J'ai choisi celui-là parce que c'est celui dont je dispose actuellement, mais pas d'inquiétude, celui-là ou un autre, vous n'aurez pas ou peu d'adaptations à faire.

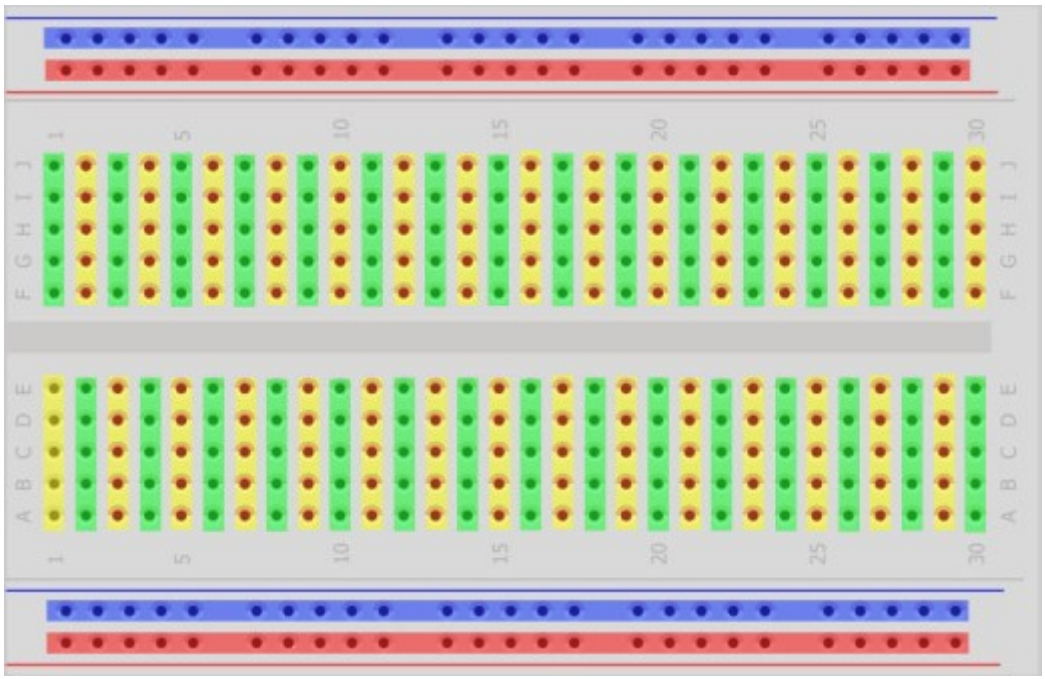
Pour compléter le dispositif, on investira dans une **plaque de câblage rapide** et quelques *jumpers* (mâle/mâle et femelle/femelle) :



III - La plaque de câblage rapide

Une **plaque de câblage rapide** (*breadboard* en anglais) permet de prototyper un montage électronique en connectant ses composants en un circuit démontable et sans soudure. La plaque est constituée de séries de trous où peuvent être enfichés fils de câblage et pattes de composants électroniques divers.

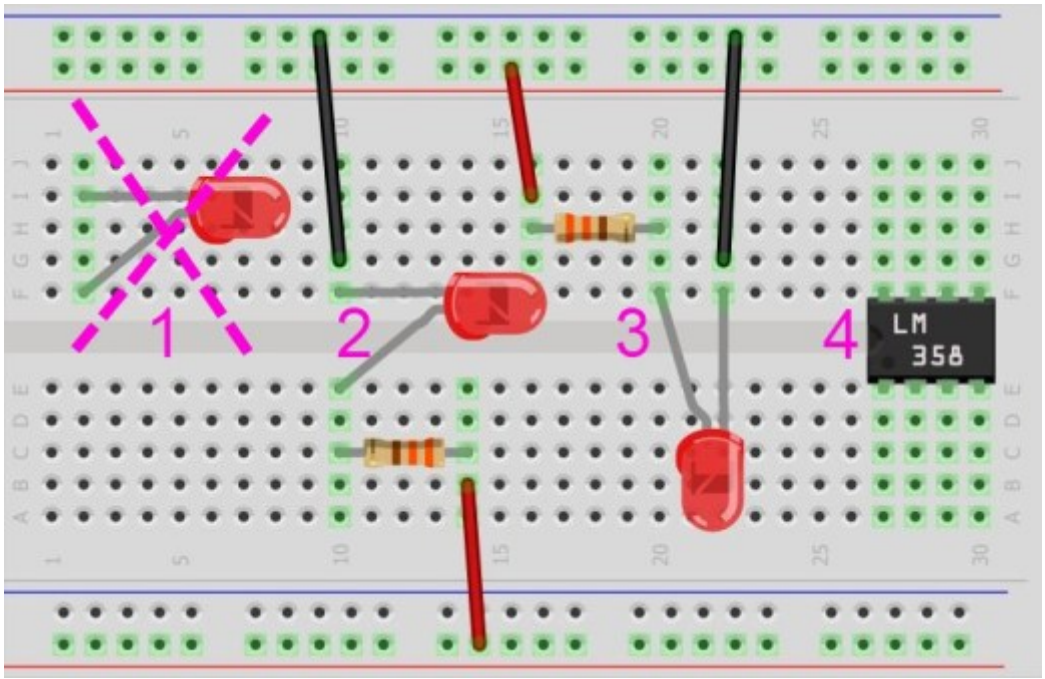
Sous la plaque, un réseau de bandes métalliques conductrices connecte ces trous selon un schéma préétabli :



Les bandes horizontales en haut et en bas sont prévues pour amener l'alimentation et la masse dans tout le montage. Les composants câblés au centre de la plaque peuvent être ainsi reliés plus facilement au 5 V (ou toute autre tension prévue) et au 0 V. Certaines plaques ont des bandes rouges et bleues pour repérer les trous à relier à l'alimentation (rouges) et ceux à relier à la masse (bleues).

Les bandes verticales au centre de la plaque sont plus courtes avec un intervalle au milieu. Cela permet à un composant de circuits intégrés de chevaucher l'intervalle et d'avoir chacune de ses pattes, situées de part et d'autre, connectées à un jeu de trous différent.

Par exemple, avec les quatre montages suivants :



Le montage 1 n'est pas correct, la cathode (-) et l'anode (+) de la LED sont reliées entre elles.

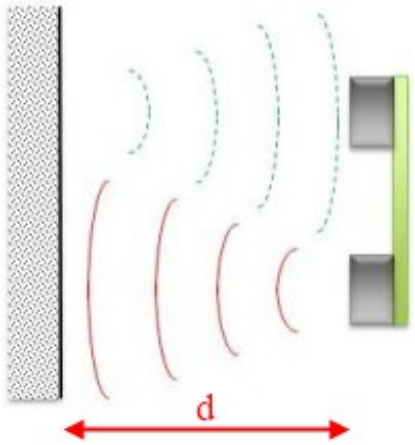
Les montages 2 et 3 sont corrects. Dans les deux montages, une LED et une résistance sont montées en série entre une ligne de masse et une ligne d'alimentation (la patte la plus courte de la LED étant la cathode à relier à la masse).

Le montage 4 est aussi correct. Chacune des huit pattes du LM358 est connectée à un jeu de trous différent.

IV - Le capteur de distance à ultrasons SRF-05

IV-A - Présentation du capteur

Ce genre de capteur est muni d'une paire d' « yeux », un émetteur qui émet un son à une fréquence de 40 kHz (et donc dans le domaine des ultrasons) et un récepteur qui collecte le son répercuté par l'obstacle.



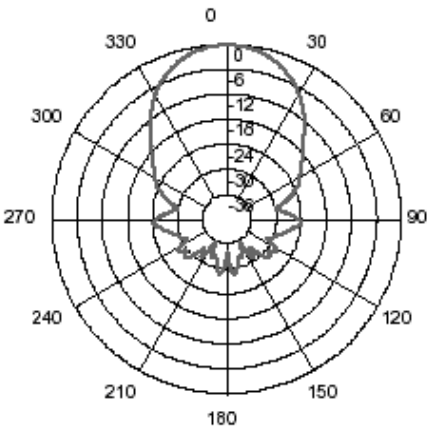
La distance à l'obstacle peut alors être calculée par le temps mis par le son pour faire l'aller-retour, la vitesse du son étant à peu près stable.

$$d = \frac{vitesse\ du\ son \times t}{2}$$

car pendant l'intervalle de temps t, le son parcourt deux fois la distance d.
vitesse du son = 340 m/s dans l'air.

Vous trouverez les caractéristiques du module et son fonctionnement détaillé dans les **spécifications techniques**.

Specifications	
Frequency	40kHz
Max Range	4 meters
Min Range	3 centimeters
Input Trigger	10uSec minimum, TTL level pulse
Echo Pulse	Positive TTL level signal, proportional to range



En deux mots, la portée de ce module ultrason est de 3 cm à 4 m. La détection de l'obstacle se produit normalement dans un cône de 30°.

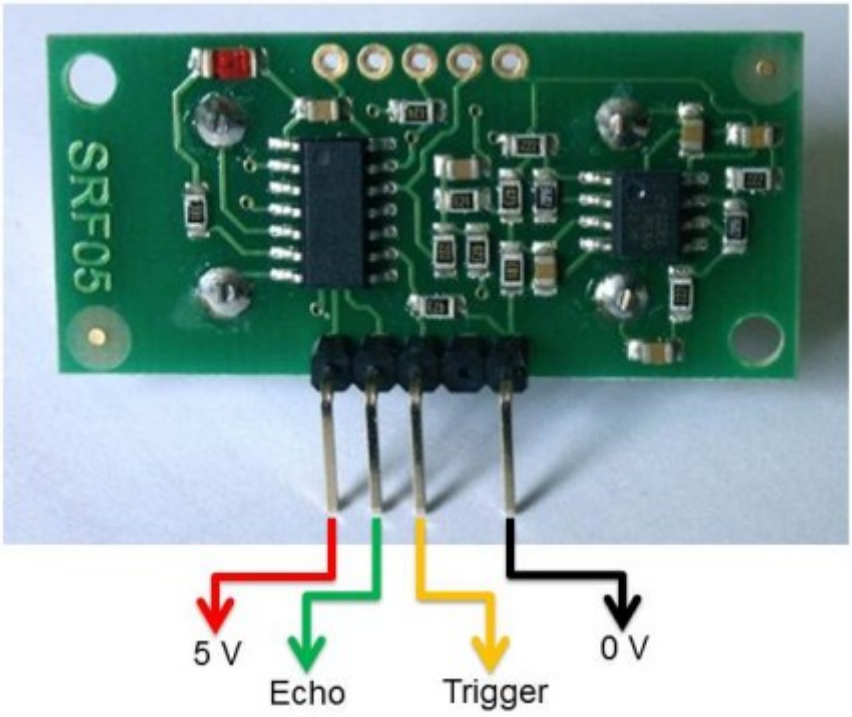
Le module ne peut faire des mesures que toutes les 50 ms (millisecondes), c'est-à-dire 20 fois par seconde.

IV-B - Connectique et mode de fonctionnement

Dans cet article, nous utiliserons exclusivement le mode à *Trigger* et *Echo* séparés.

Voici donc les broches de connexion utilisées dans ce mode :

- alimentation 5 volts ;
- broche de communication *Echo* (broche configurée en sortie) ;
- broche de communication *Trigger* (broche configurée en entrée) ;
- masse 0 volt.



IV-C - Principe de fonctionnement

Je reprends ici les explications de la documentation.

Pour utiliser le SRF-05, il faut lui envoyer une demande de mesure, que l'on appelle *Trigger*. C'est une impulsion de 5 volts dont la durée est de 10 µs (microsecondes) minimum. Par précaution, il est conseillé d'envoyer un signal *Trigger* dont la durée est entre 12 et 15 µs.

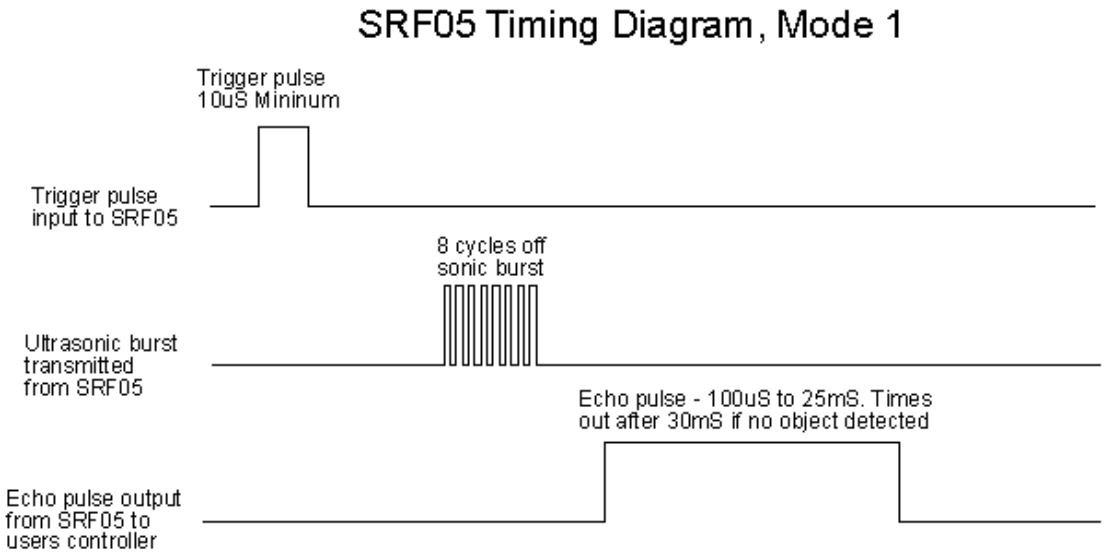
Dès qu'il reçoit ce signal, le SRF-05 envoie un train d'ultrasons de 40 kHz (huit signaux de courtes périodes). Dès l'envoi du dernier ultrason, le SRF-05 envoie un signal sur la broche *Echo* qu'il maintient à 5 volts jusqu'à ce qu'il reçoive un retour des ultrasons. À ce moment-là, il redescend le signal *Echo* à la masse (0 volt).

Nous avons donc un signal Echo dont la durée est proportionnelle à la distance entre le SRF-05 et l'objet.

Le signal *Echo* redescend à la masse au bout de 30 ms si aucun obstacle n'est détecté (*time-out*).

Afin d'éviter des résultats faussés par des mesures lancées quasi simultanément, il est recommandé d'attendre 50 ms entre deux demandes de mesure, ce qui permet de déclencher jusqu'à 20 mesures par seconde d'utilisation au maximum.

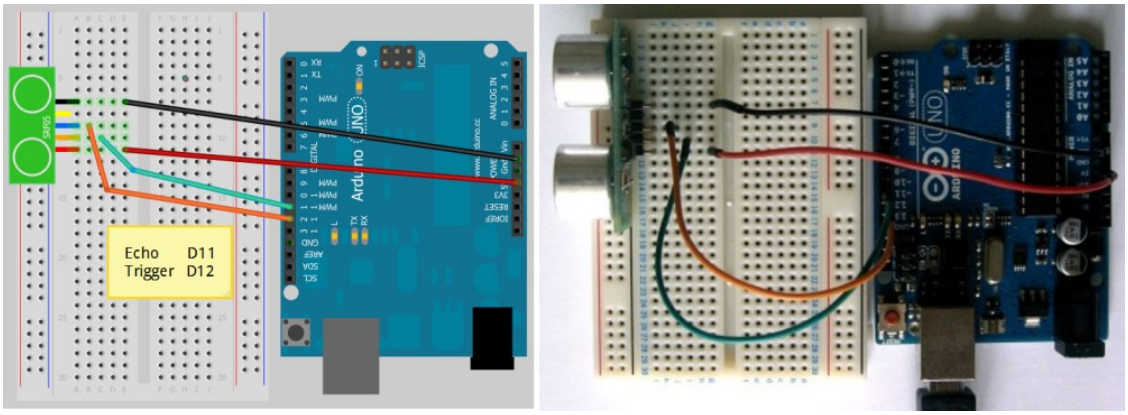
Le principe du fonctionnement est résumé sur le chronogramme constructeur ci-dessous :



IV-D - Montage du capteur

Dans la suite de ce tutoriel, les broches *Echo* et *Trigger* seront connectées respectivement aux broches numériques D11 et D12 de la carte Arduino.

Voici un schéma et une photo de cette première maquette :



IV-E - La gestion du capteur par programmation

Une fois assimilé le principe de fonctionnement du capteur, il ne nous reste plus qu'à gérer la programmation des broches *Echo* et *Trigger* reliées à deux broches numériques de la carte Arduino.

Le jeu d'instructions de haut niveau proposé par les concepteurs du langage nous facilite la vie. La gestion s'opère en trois temps :

- on configure deux broches de l'Arduino avec l'instruction **pinMode** :

```
pinMode(ECHO_PIN, INPUT);     // la broche ECHO de la carte est une entrée numérique
pinMode(TRIGGER_PIN, OUTPUT); // la broche TRIGGER de la carte est une sortie numérique
```

- pour déclencher la mesure, on envoie une impulsion sur la broche *Trigger* pendant une douzaine de microsecondes grâce à l'instruction **digitalWrite** :

```
digitalWrite(TRIGGER_PIN, HIGH);    // signal TRIGGER à l'état HAUT (5V)
delayMicroseconds(12);              // durée de l'état haut = 12 microsecondes
digitalWrite(TRIGGER_PIN, LOW);     // signal TRIGGER à l'état BAS (0V)
```

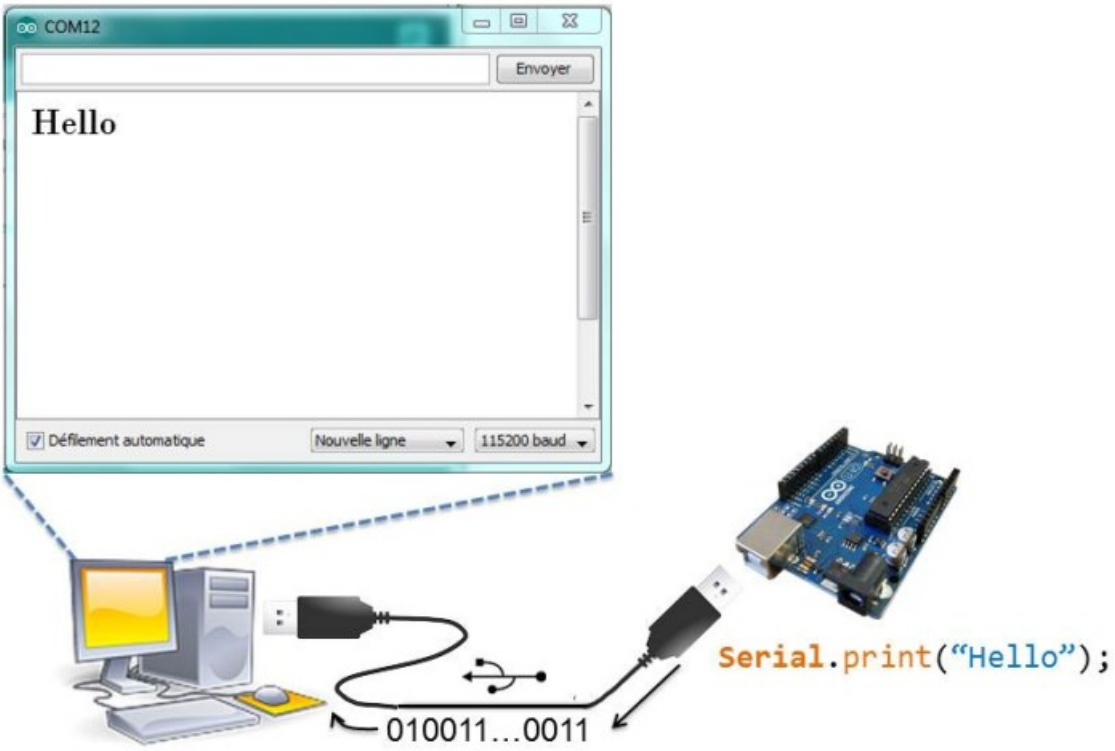
- on « attend » le retour du signal *Echo* à l'état BAS et on récupère la durée pendant laquelle le signal *Echo* était maintenu à l'état HAUT. Tout cela est rendu très facile avec l'instruction **pulseIn**.

```
echoPulseTime = pulseIn(ECHO_PIN, HIGH);
  // attend le retour du signal ECHO au niveau BAS et retourne la durée du signal
```

Le programme complet avec l'affichage de la valeur brute de la durée du signal *Echo* dans le moniteur Série donnerait ceci :

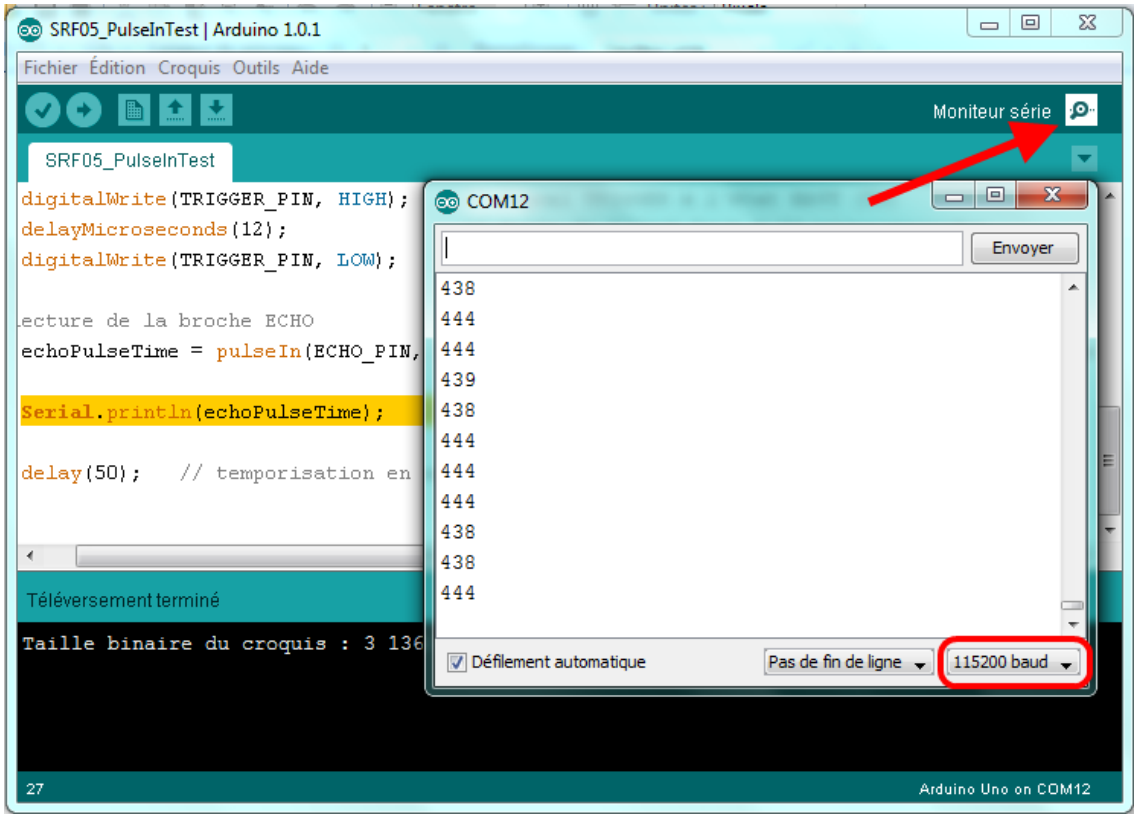
```
/*
  1er programme SRF-05
*/

#define ECHO_PIN      11  // broche ECHO du SRF05 sur D11
#define TRIGGER_PIN   12  // broche TRIGGER du SRF05 sur D12
unsigned long echoPulseTime = 0;   // durée du signal ECHO en microsecondes

void setup() {
// configuration des E/S numériques de la carte Arduino
  pinMode(ECHO_PIN, INPUT);     // la broche ECHO de la carte est une entrée numérique
  pinMode(TRIGGER_PIN, OUTPUT); // la broche TRIGGER de la carte est une sortie numérique

// initialisation du port série à 115200 bauds
  Serial.begin(115200);
}

void loop() {
// déclenchement de la mesure
    digitalWrite(TRIGGER_PIN, HIGH);    // signal TRIGGER à l'état HAUT (5V)
    delayMicroseconds(12);              // durée de l'état haut = 12 microsecondes
    digitalWrite(TRIGGER_PIN, LOW);     // signal TRIGGER à l'état BAS (0V)

// lecture de la broche ECHO
    echoPulseTime = pulseIn(ECHO_PIN, HIGH);
  // attend le retour du signal ECHO au niveau BAS et retourne la durée du signal

    Serial.println(echoPulseTime);   // affichage du résultat brut dans le moniteur Série

    delay(50);   // temporisation en ms avant de recommencer la prochaine mesure
}
```

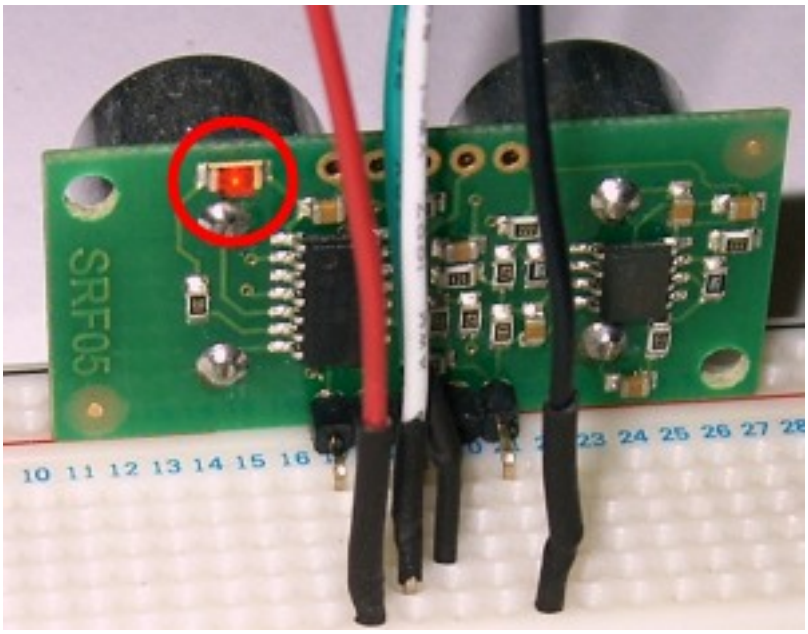
N'oubliez pas que le programme est « flashé » dans la mémoire du microcontrôleur. Grâce à l'objet **Serial**, vous établissez une connexion série entre la carte Arduino et l'ordinateur via le câble USB pendant le déroulement du programme. Les méthodes **Serial.print** et **Serial.println** permettent de communiquer les mesures à l'ordinateur et les afficher dans le moniteur Série (ne pas oublier de régler la vitesse de transmission conformément à la déclaration Serial.begin(115200)).



Communication Série : Arduino vers PC

Voilà une méthode très pratique en phase de test et qui permet de rendre compte très facilement du fonctionnement du capteur (en plus de la petite LED rouge sur la façade arrière du capteur qui doit clignoter à la fréquence des déclenchements).





IV-F - Vers un fonctionnement multitâche - La bibliothèque NewPing

Ce premier programme, fonctionnel au premier abord, présenterait toutefois un inconvénient majeur si vous aviez à gérer un fonctionnement multitâche tel celui d'un robot dont on doit à la fois gérer le déplacement tout en détectant les obstacles alentour.

Le fonctionnement séquentiel du programme avec les instructions pulseIn et delay mettent en effet inutilement le microcontrôleur « en attente » rendant toute détection du monde extérieur inopérante pendant plusieurs fractions de seconde à chaque passage dans la boucle loop.

Par chance, tous les microcontrôleurs actuels, et celui d'Arduino ne fait pas exception, sont équipés de moyens matériel et logiciel pour gérer les **interruptions**. On rappelle qu'une interruption est un événement pouvant se produire n'importe quand pendant le déroulement d'un programme. Lors d'une interruption (déclenchée périodiquement grâce à un *Timer* ou sur un événement extérieur comme un front montant ou descendant d'un signal numérique capturé sur une broche, voir la fonction **attachInterrupt()**), le programme est interrompu et un sous-programme de gestion de l'interruption est appelé. Une fois le sous-programme terminé, le programme principal reprend son cours normalement.

Le jeu d'instructions par défaut proposé par Arduino pour la gestion des interruptions n'est pas très riche et n'autorise que les interruptions résultant d'un changement d'état (front montant ou descendant) d'une entrée numérique externe (et pas sans inconvénient en plus).

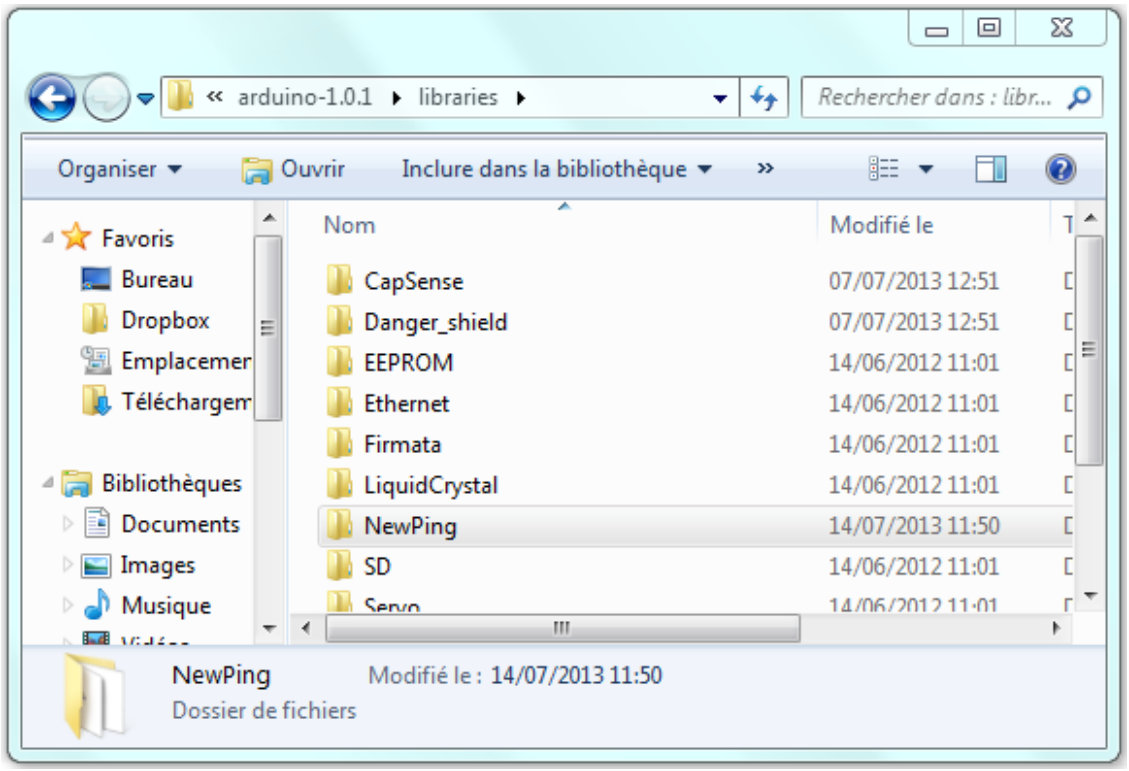
Fort heureusement, le jeu d'instructions standards peut être enrichi grâce aux nombreuses bibliothèques disponibles sur le Net comme celle qui permet de gérer des interruptions périodiques avec le **Timer2** du microcontrôleur AVR Atmel de l'Arduino.

Afin de nous faciliter les développements et rendre la gestion des interruptions la plus transparente possible, je vous propose d'utiliser la bibliothèque **NewPing** que nous allons décrire ci-après.

IV-F-1 - Installation de la bibliothèque

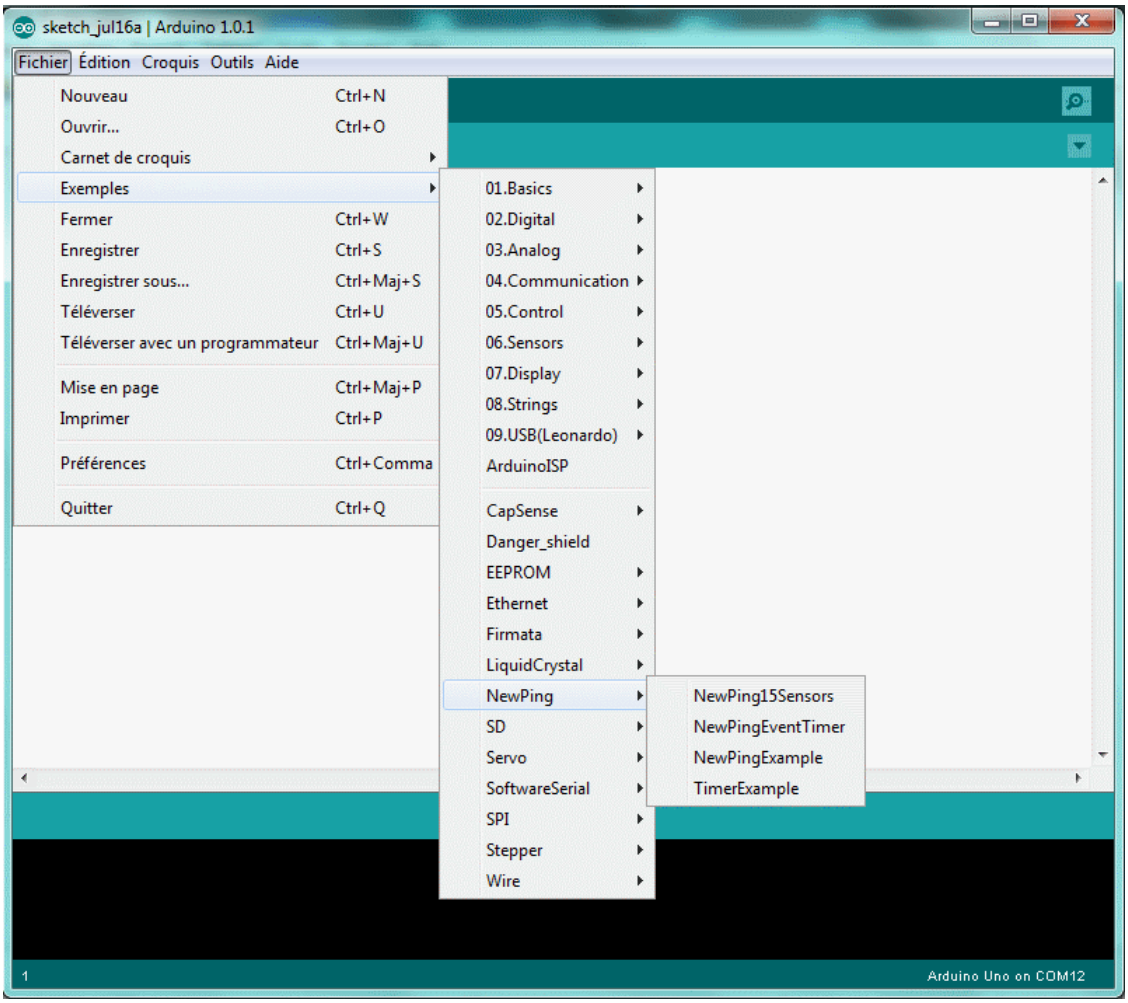
Vous pouvez télécharger cette bibliothèque en allant faire un tour dans l'**aire de jeu Arduino**.

L'installation d'une bibliothèque tierce et compatible avec l'environnement Arduino est très simple. Une fois l'archive décompressée, vous la déplacez dans le sous-répertoire **libraries** du dossier d'installation :



Sous Linux, il faut aller voir du côté du répertoire **/usr/share/arduino/libraries**.

C'est tout ! Lorsque vous démarrez l'IDE Arduino, des exemples d'utilisation sont proposés dans le menu **Fichier>Exemples>Newping** :



Ces exemples mettent en œuvre la bibliothèque **NewPing** en incluant la directive :

```
#include <NewPing.h>
```

IV-F-2 - Gestion des interruptions

J'ai repris l'exemple **NewPingEventTimer**, ce qui donne le code suivant :

NewPingEventTimer

```
/* ---------------------------------------------------------------------------
 d'après l'exemple NewPing/NewPingEventTimer
 ---------------------------------------------------------------------------*/
#include <NewPing.h>

#define ECHO_PIN     11  // broche Echo sur D11
#define TRIGGER_PIN  12  // broche Trigger sur D12

#define MAX_DISTANCE 50 // distance maximale (en centimètres) que nous voulons "pinger". La portée maximale du SR

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // instanciation d'un nouvel objet NewPing.

unsigned int
 pingSpeed = 50; // période d'envoi du "ping". Une période de 50 millisecondes correspond à 20 "pings" par second
unsigned long pingTimer;     // timer, pour le prochain "ping".

void setup() {
  Serial.begin(115200); // moniteur Série à 115200 bauds pour afficher les résultats.
```

NewPingEventTimer

```
 pingTimer = millis(); // contient le temps écoulé en millisecondes depuis le début de l'exécution du programme.
}

void loop() {
  if (millis() >= pingTimer) {   // si temps écoulé depuis dernier "ping", on fait à nouveau un "ping".
    pingTimer += pingSpeed;      // temps pour déclenchement du prochain "ping"

 sonar.ping_timer(echoCheck); // envoie un "ping". Interruption toutes les 24 microsecondes (interruption du Time
  }
  //
  // autres tâches ici, fonctionnement en multitâche
  //
}

void echoCheck() { // sous-programme d'interruption
  if (sonar.check_timer()) { // "ping" complété dans les limites de distance souhaitées ?
    // ajouter votre code d'interruption ici
    Serial.print("Ping: "); // affichage dans le moniteur Série

 Serial.print(sonar.convert_cm(sonar.ping_result)); // "ping" complété, convertit les microsecondes de la durée d
    Serial.println("cm");
  }
}
```

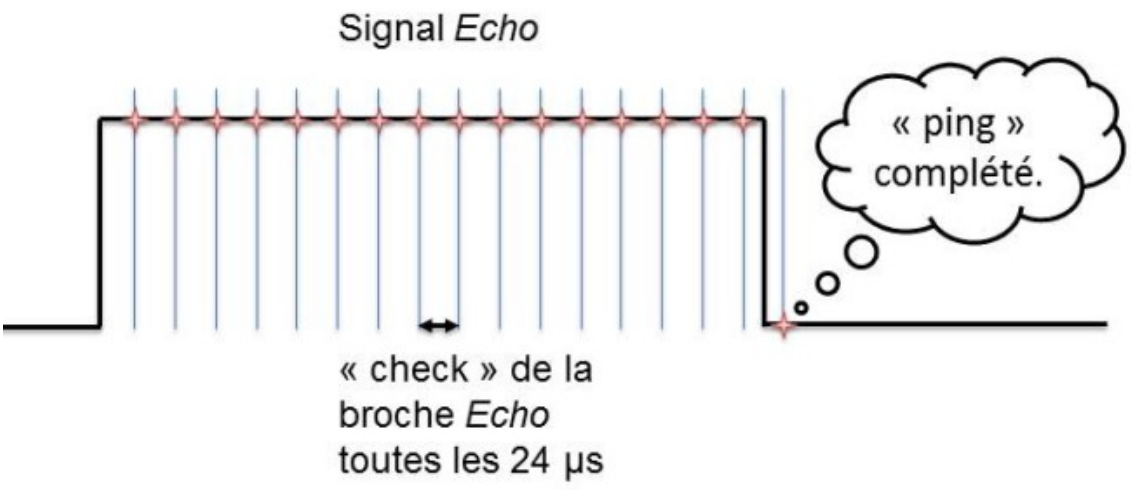
Vous pouvez télécharger directement ce programme dans la carte et le tester. Pensez à connecter les broches *Echo* et *Trigger* du capteur respectivement aux broches numériques 11 et 12 de la carte *Arduino*. Dans le moniteur Série, réglez la vitesse de transmission à 115 200 bauds pour voir défiler les mesures du capteur.

L'objet NewPing est instancié à la ligne :

```
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // instanciation d'un nouvel objet NewPing.
```

Un « ping » est déclenché toutes les 50 ms, mais cette fois sans faire de pause dans le programme avec un delay(), grâce à la fonction **millis()**.

On ordonne alors une interruption toutes les 24 µs(*) pour « checker » l'état de la broche *Echo* du capteur et incrémenter un compteur. Lorsqu'un retour à l'état LOW est détecté, le « ping » est complété et on peut convertir la durée du signal en distance.



(*) On peut faire remarquer par un petit calcul que si on éloigne ou rapproche un obstacle de 4 mm, la durée du signal *Echo* varie justement d'environ 24 µs. La précision de mesure est

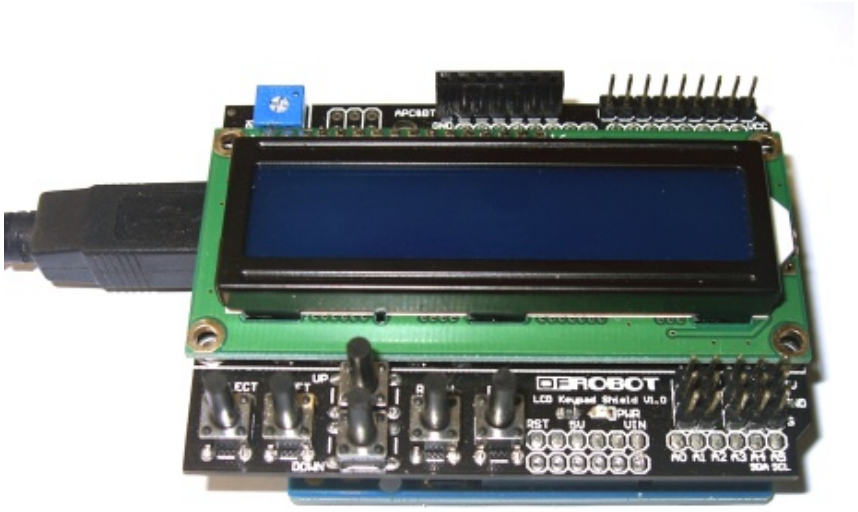
donc de 4 mm par défaut, mais celle-ci peut être modifiée en changeant la ligne du fichier d'entête **libraries/NewPing/NewPing.h** :

```
#define ECHO_TIMER_FREQ 24
```

Il n'y a plus d'attente ou de délai dans la boucle principale et la scrutation périodique de l'état du capteur en toute transparence par interruption autorise un fonctionnement multitâche.

Vous devriez d'ailleurs constater un affichage plus rapide et plus fluide des résultats dans le moniteur Série par rapport au premier programme.

V - Le « LCD Keypad Shield »



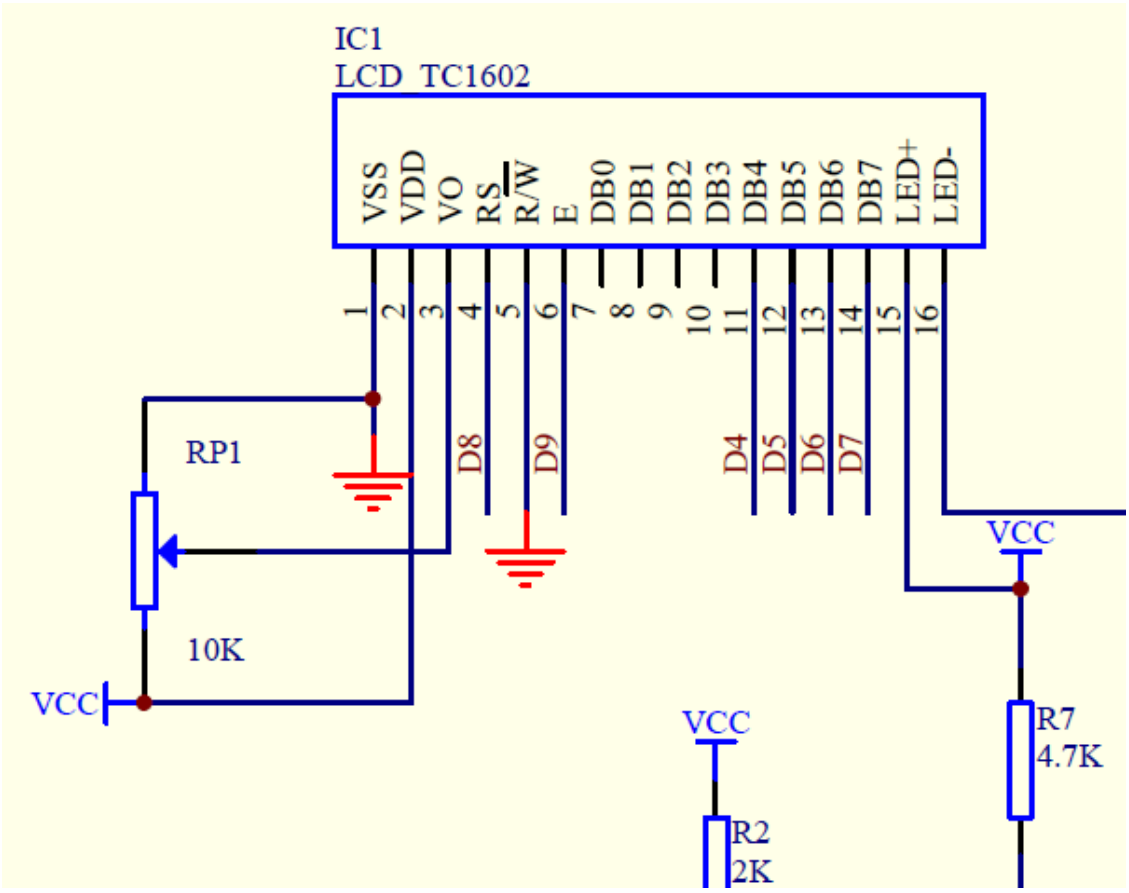
Pour piloter l'affichage sur l'écran LCD du *shield*, dont le driver de l'afficheur est compatible avec le standard Hitachi, vous avez besoin de la bibliothèque **LiquidCrystal** proposée par défaut et dont les exemples d'utilisation sont disponibles dans l'IDE en suivant le menu **Fichier>Exemples>LiquidCrystal**.

V-A - Mode de fonctionnement de l'afficheur

La syntaxe d'instanciation d'un objet LiquidCrystal diffère selon le mode de fonctionnement de l'afficheur :

```
LiquidCrystal lcd(rs, enable, d4, d5, d6, d7) // mode 4 bits - RW non connectée (le plus simple!)
LiquidCrystal lcd(rs, rw, enable, d4, d5, d6, d7) // mode 4 bits - RW utilisée
LiquidCrystal lcd(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7) // mode 8 bits - RW non connectée
LiquidCrystal lcd(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7) //mode 8 bits - RW utilisée
```

Dans la documentation du *shield*, vous devriez retrouver le schéma électrique :



Dans notre cas, l'afficheur fonctionne avec quatre bits de données (D4, D5, D6 et D7). Les broches de commandes RS et E seront connectées respectivement aux connecteurs D8 et D9 de l'Arduino. C'est le mode de fonctionnement le plus simple et, surtout, celui qui utilise le moins grand nombre de connecteurs sur la carte Arduino.

Ainsi, le code d'instanciation, à adapter si vous prenez un *shield* au brochage différent, ressemblera à :

```
#include <LiquidCrystal.h>

#define RS   8
#define E    9
#define DB4  4
#define DB5  5
#define DB6  6
#define DB7  7

LiquidCrystal lcd(RS, E, DB4, DB5, DB6, DB7);
```

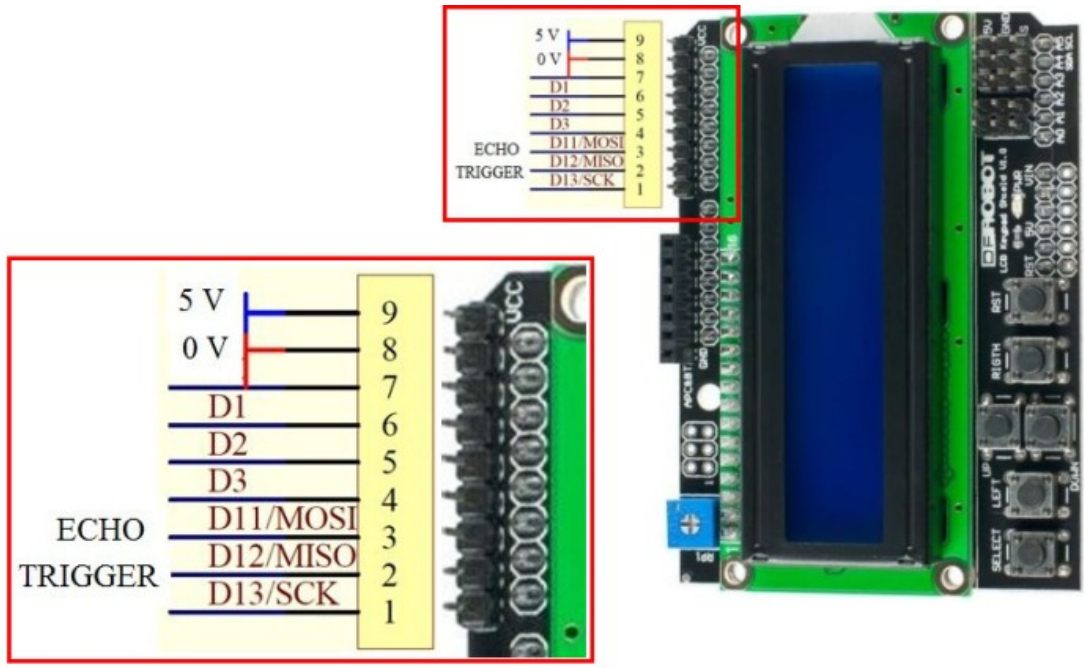
Vous pouvez maintenant enficher le *shield* sur la carte Arduino (débranchez les fils du capteur à ultrasons reliés à la carte pour le moment) afin de tester et découvrir cette bibliothèque en parcourant les codes des nombreux exemples fournis, dont le fameux « HelloWorld ».

Vous trouverez la référence de la bibliothèque sur le **site officiel** et en français sur le site **www.mon-club-elec.fr**.

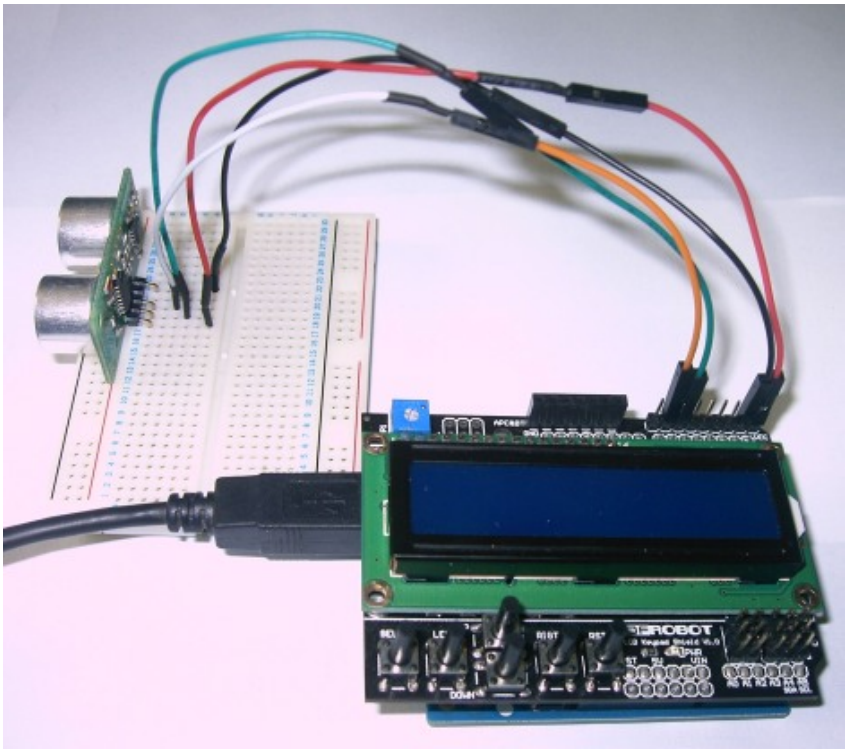
V-B - Intégration du capteur à ultrasons SRF-05

V-B-1 - Montage

D'après la documentation du *shield*, les connecteurs mâles repérés sur la photo ci-dessous reprennent les broches D11 et D12 de la carte Arduino ainsi que des broches 0 et 5 volts pour alimenter le capteur.



Nous pouvons maintenant câbler le capteur ultrason sur le *shield* lui-même enfiché sur la carte Arduino (« le fil rouge sur le bouton rouge, le fil vert sur le bouton vert… »).



V-B-2 - Le code d'affichage de la distance de l'obstacle sur l'afficheur LCD

Pour afficher les informations du capteur sur l'afficheur LCD, il faut reprendre le code du paragraphe **IV.F.2** en évoquant cette fois les commandes **setCursor()** et **print()** :

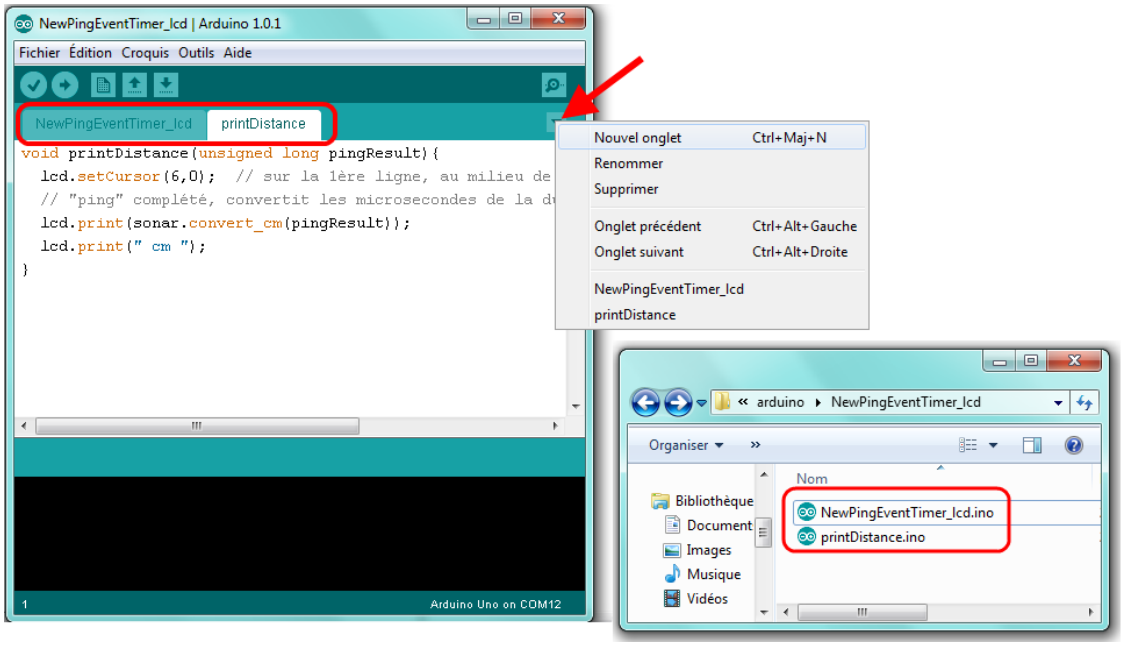
```
/* ------------------------------------------------------------------------
 Affichage de la distance en centimètres
 ------------------------------------------------------------------------*/
```

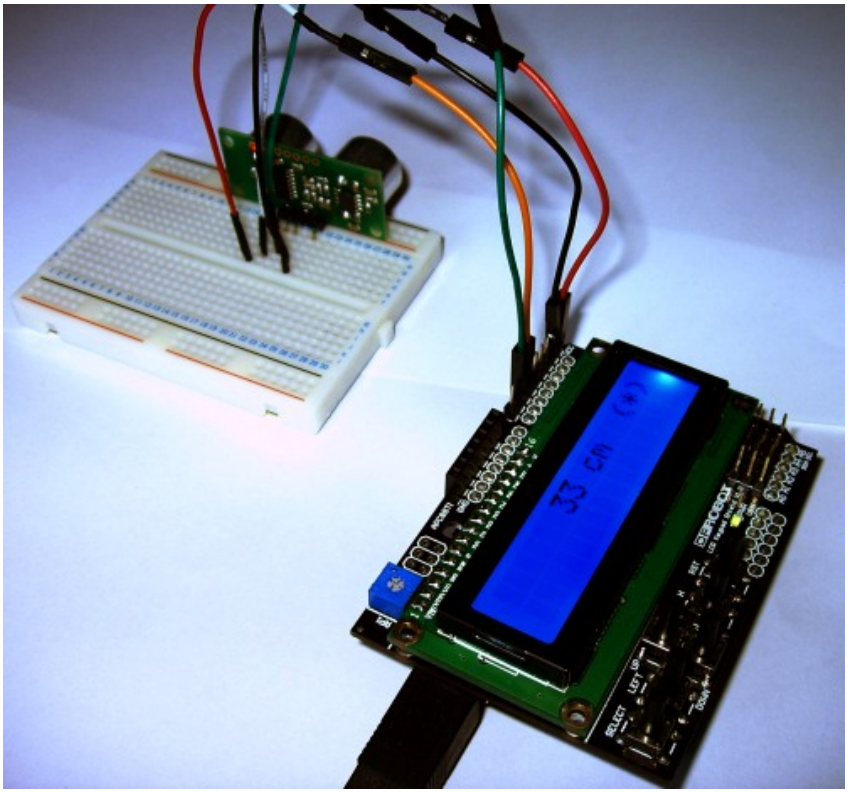
```
#include <NewPing.h>
#include <LiquidCrystal.h>

#define ECHO_PIN     11  // broche Echo sur D11
#define TRIGGER_PIN  12  // broche Trigger sur D12

#define MAX_DISTANCE 200 // distance maximale (en centimètres) que nous voulons "pinger". La portée maximale du Si

#define RS   8
#define E    9
#define DB4  4
#define DB5  5
#define DB6  6
#define DB7  7

//#define DEBUG
#undef DEBUG

LiquidCrystal lcd(RS, E, DB4, DB5, DB6, DB7);
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // instanciation d'un nouvel objet NewPing.

unsigned int
 pingSpeed = 50; // période d'envoi du "ping". Une période de 50 millisecondes correspond à 20 "pings" par seconde
unsigned long pingTimer;     // timer, pour le prochain "ping".

void setup() {
  Serial.begin(115200); // moniteur série à 115200 bauds pour afficher les résultats.
  lcd.begin(16,2);  // intialisation afficheur 16 x 2 caractères

 pingTimer = millis(); // contient le temps écoulé en millisecondes depuis le début de l'exécution du programme.
}

void loop() {
  if (millis() >= pingTimer) {   // si temps écoulé depuis dernier "ping", on fait à nouveau un "ping".
    pingTimer += pingSpeed;      // temps pour déclenchement du prochain "ping"
    lcd.setCursor(13,0);
    lcd.print("(*)");     // pas de "ping" valide initialement

 sonar.ping_timer(echoCheck); // envoie un "ping". Interruption toutes les 24 microsecondes (interruption du Time
  }
  //
  // autres tâches ici, fonctionnement en multitâche
  //
}

void echoCheck() { // sous-programme d'interruption
  if (sonar.check_timer()) { // "ping" complété dans les limites de distance souhaitées ?
    // ajouter votre code d'interruption ici
    lcd.setCursor(13,0);
    lcd.print("   "); // efface l'étoile, le "ping" est complété
    printDistance(sonar.convert_cm(sonar.ping_result), " cm"); // affichage en centimètres sur LCD
  }
}

void printDistance(unsigned int distance, String unit ){
#ifdef DEBUG  // si mode DEBUG
  Serial.print("Ping: "); // affichage dans le moniteur série
  Serial.print(distance);
  Serial.println(unit);
#endif

  lcd.setCursor(5,0);  // sur la 1re ligne, au milieu de l'afficheur
  if (distance<10)
    lcd.print("  ");  // efface les centaines et dizaines
  else if (distance<100)
    lcd.print(" ");  // efface les centaines

  lcd.print(distance);
  lcd.print(unit);
}
```

Le mode DEBUG peut être réactivé à tout moment en phase de test avec affichage sur le moniteur Série.

De plus, si la mesure n'est pas complétée après un « ping » (parce que la distance est supérieure à MAX_DISTANCE), l'affichage est figé avec une étoile (*) dessinée sur l'écran LCD.

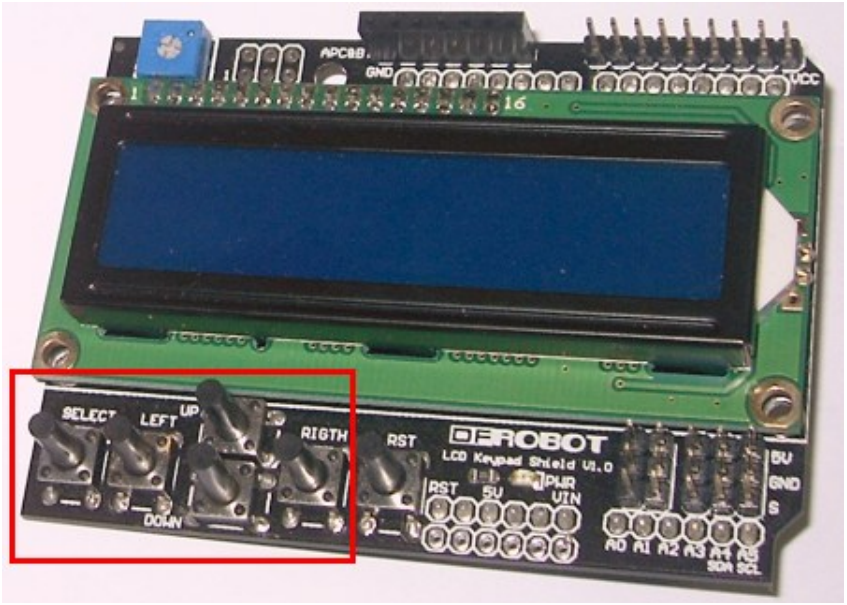
Petit raffinement de l'IDE Arduino, vous pouvez rédiger le code de la fonction printDistance() dans un onglet séparé, le premier onglet contenant le programme principal.





VI - Programmation d'un bouton

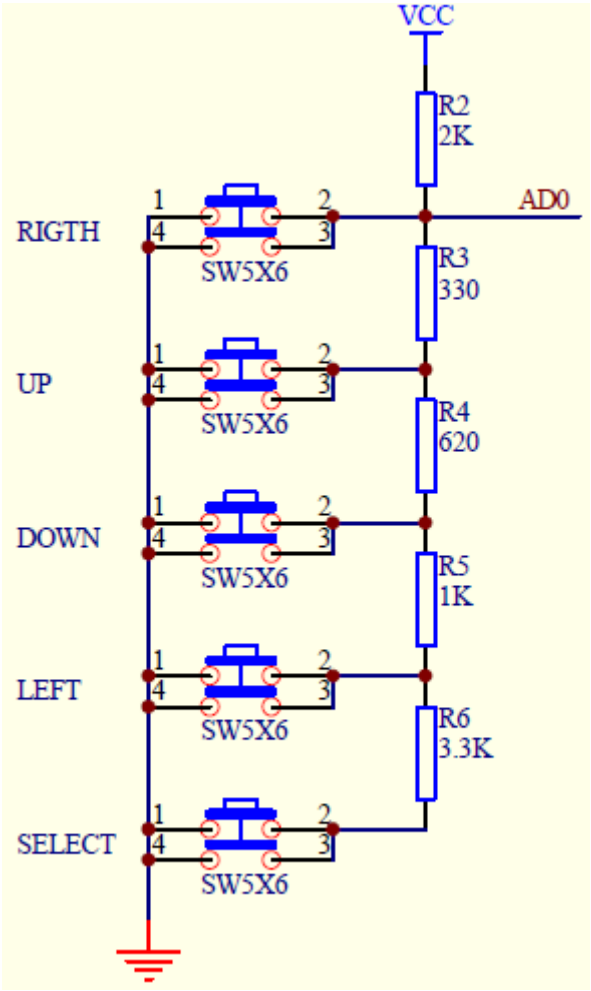
En plus de son afficheur LCD, le *shield* comporte sur sa façade cinq boutons programmables nommés SELECT, LEFT, RIGHT, UP, DOWN (ainsi qu'un bouton RST, pour faire un « Reset » de la carte, situé trop à proximité à mon goût).



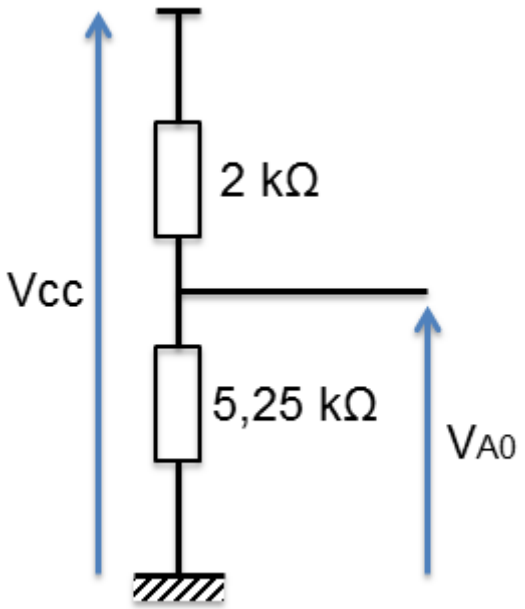
Par la suite nous aimerions profiter d'un de ces boutons pour ajouter une petite fonctionnalité : l'appui sur la touche SELECT permet de faire basculer l'affichage de la distance de l'obstacle en centimètres (cm) ou en pouces (in). Certes, la fonctionnalité est discutable, mais la programmation d'un bouton vous sera utile dans de nombreuses autres circonstances.

VI-A - Fonctionnement des boutons, un peu de physique/électricité

Voici l'extrait du schéma électrique fourni par le constructeur concernant les cinq boutons-poussoirs.



La tension appliquée à l'entrée analogique A0 de la carte dépend de la touche pressée. Par exemple si l'utilisateur presse le bouton SELECT (comportement d'un interrupteur fermant un circuit normalement ouvert), le schéma électrique équivalent est :



Un retour au principe du **diviseur de tension** permet de calculer la tension à l'entrée A0 :

$$V_{A0}=\frac{330+620+1000+3300}{330+620+1000+3300+2000}\times V_{cc}=\frac{5,25}{5,25+2}\times 5=3,62\,V$$

Soit la valeur théorique en sortie du **convertisseur Analogique-Numérique** (10 bits) :

$$N=3,62\times\frac{2^{10}}{5}=741$$

Vous pouvez faire de même pour les autres boutons.

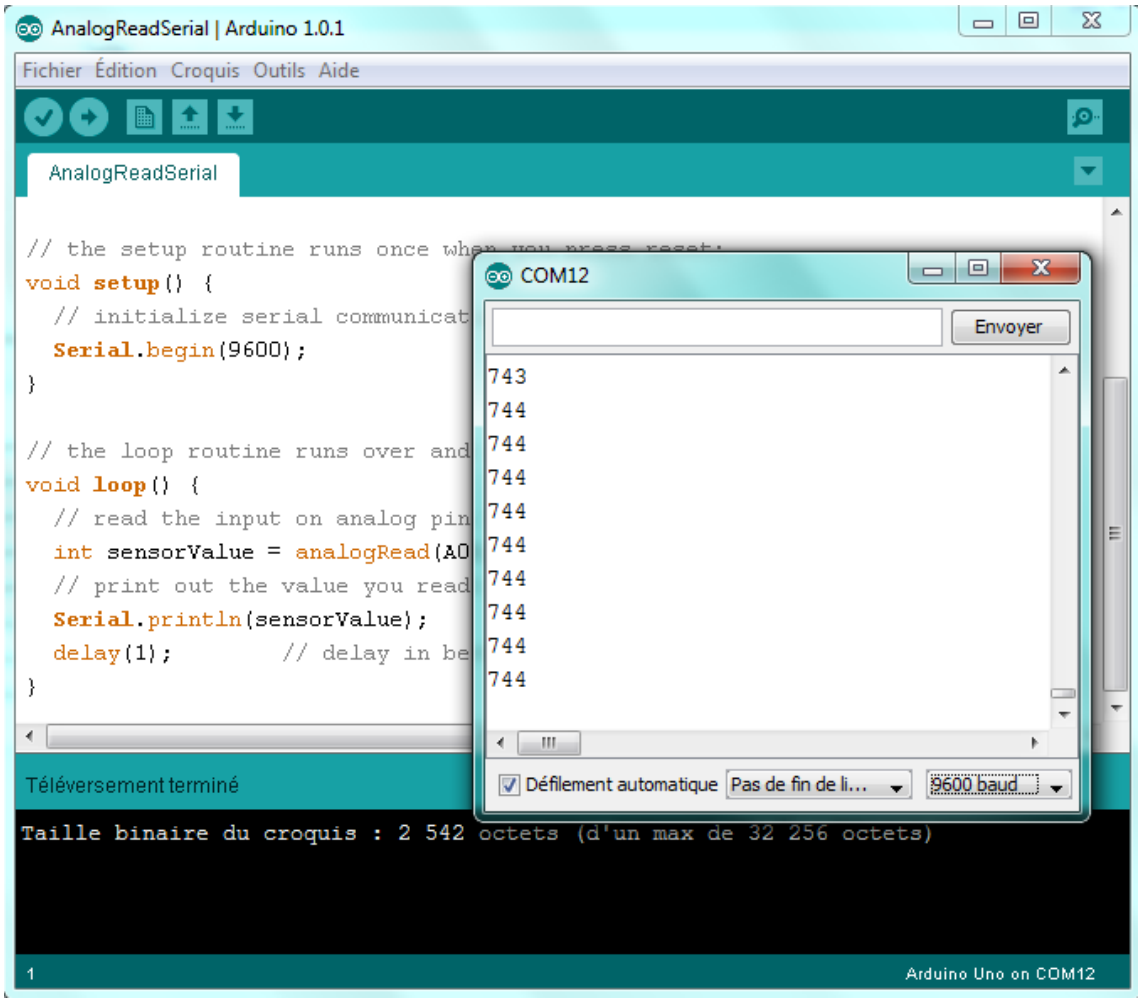
Ce principe de clavier analogique est fort simple, ne nécessite aucune bibliothèque extérieure et ne consomme qu'une entrée de la carte Arduino.

Pour vérifier que le principe fonctionne, il vous suffit de reprendre le programme exemple **AnalogReadSerial** (dans le menu **Fichier>Exemples>01.Basics**) qui se contente d'afficher en boucle la valeur lue sur l'entrée A0 dans le moniteur Série :

AnalogReadSerial

```
/*
  AnalogReadSerial
  Reads an analog input on pin 0, prints the result to the serial monitor.
  Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground.

 This example code is in the public domain.
 */

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```

Téléchargez ce programme et testez-le en pressant le bouton SELECT (on oublie et on met de côté le capteur ultrason pour ce test) :



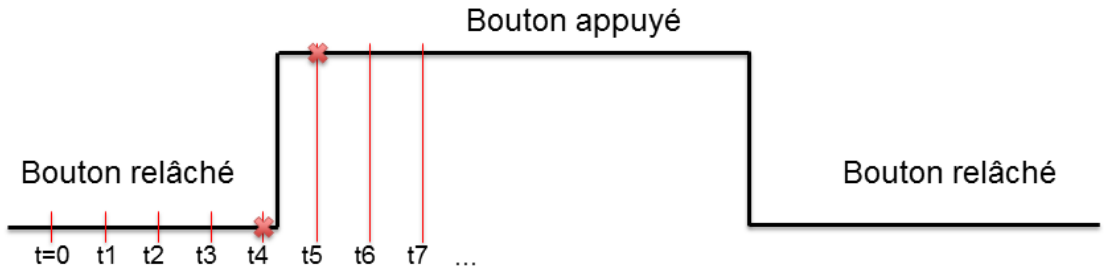
N'oubliez pas de régler la vitesse de transmission à 9 600 bauds dans le moniteur Série conformément à la déclaration du programme.

Lorsque la touche SELECT est maintenue pressée, la valeur lue tourne autour de 744 dans mon cas. L'écart par rapport à la valeur théorique (741), très faible, mais qui peut fluctuer d'un *LCD KeyPad Shield* à l'autre, peut s'expliquer par la précision sur les valeurs des résistances électriques.

VI-B - Gestion du bouton

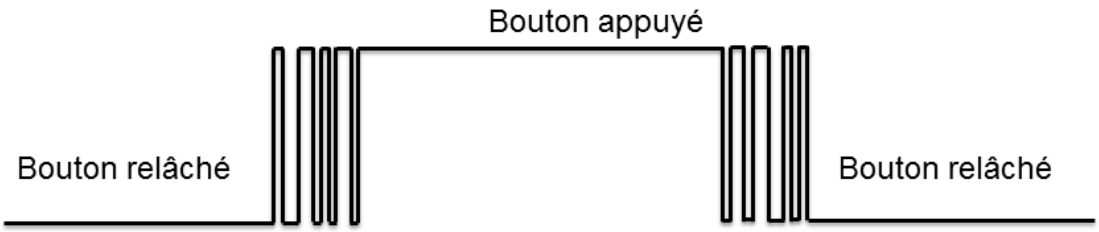
On revient à l'idée de basculer l'affichage de la distance en centimètres ou en pouces à chaque fois que l'utilisateur appuie sur le bouton SELECT.

Une solution simple consiste à relever périodiquement l'état du bouton (une fois à chaque passage dans la boucle loop() suffira dans notre cas), ici aux instants 0, t1, t2, t3…



Le changement d'unité doit intervenir seulement dans le cas où le bouton bascule de l'état « relâché » à l'état « appuyé » entre deux instants consécutifs, ici entre les instants t4 et t5.

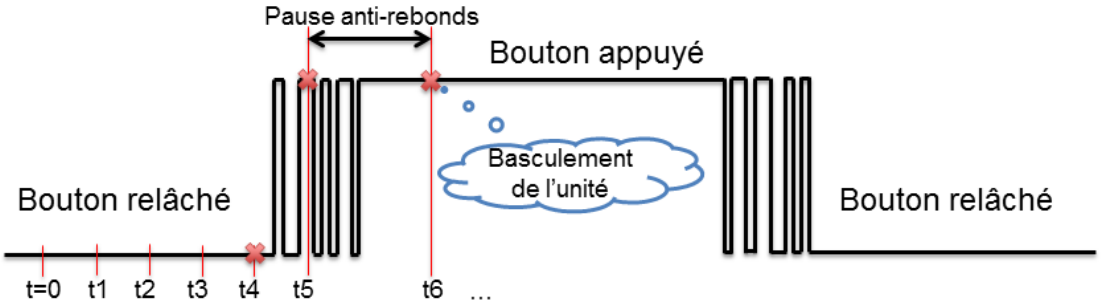
Seulement, la réalité du terrain complique les choses puisque la séquence d'appui du bouton en situation réelle nous donnera un chronogramme qui ressemble plutôt à la figure ci-dessous :



Les boutons-poussoirs possèdent en effet un inconvénient de taille, ils sont sujets aux **rebonds**, c'est-à-dire qu'ils ont la désagréable habitude de ne pas passer directement d'un état à l'autre. Pendant une durée de l'ordre de quelques millisecondes, l'état du bouton va osciller. Chaque rebond, même ceux qui se produisent pendant le relâchement du bouton, risque alors d'être interprété comme une demande de changement d'unité par l'utilisateur.

À défaut de solution matérielle immédiate sous la main pour filtrer ces rebonds (filtre analogique à base de circuit RC par exemple), on se tourne vers une solution logicielle.

Une solution simple à mettre en œuvre consiste à faire une **pause** (un delay() à rajouter) pour passer les rebonds, juste après la détection du premier changement d'état entre t4 et t5. Si, à l'instant t6, à la fin des rebonds, le bouton est toujours à l'état « appuyé » on peut opérer le changement d'unité pour la distance (centimètres - pouces) :



C'est ce principe que nous retiendrons par la suite, au risque de sacrifier un peu sur la fréquence de déclenchement des mesures de distance à cause de la pause anti-rebonds supplémentaire.

VI-B-1 - Une classe personnalisée Button

Pour la gestion des boutons, pourquoi ne pas écrire une classe personnalisée, au sens du langage C++, et réutilisable pour d'autres applications ?

Rappelez-vous que le langage Arduino dans l'IDE agit comme une enveloppe du langage C++ avec un compilateur AVR GCC sous-jacent.

Si vous n'êtes pas familier des classes C++, vous pouvez consulter la ressource : **How to write libraries for the Arduino ?** (ou plus généralement dans **les tutoriels C++ pour débuter** sur Developpez.net).

Le code de la classe Button proposé ci-dessous n'est qu'une ébauche qui ne demande qu'à être complétée.

Le fichier d'entête :

Button.h
```
#ifndef BUTTON_H
#define BUTTON_H
#include <Arduino.h> // indispensable ici, le type boolean n'est pas implanté
class Button{
private:
  int _adcValue;
  boolean _buttonPrevious;
  int _buttonHysteresis;
  int _bounceDelay;
public:
  Button(int,int);
  boolean pressed();
  ~Button();
};
#endif
```

Le code de la classe :

ButtonClass
```
/*********************************************************************
Classe Button
Cette classe, très incomplète, permet de détecter l'appui d'un bouton
d'un clavier analogique sur l'entrée A0.
http://www.developpez.net/forums/u283256/f-leb/   aout 2013
Ce code est dans le domaine public.
*********************************************************************/
#include "Button.h"

Button::Button(int adcValue, int bounceDelay){
  _adcValue=adcValue; // valeur issue de la conv. analogique-numérique
  _bounceDelay=bounceDelay; // pause anti-rebonds
  _buttonPrevious=false;
  _buttonHysteresis=10;  // fourchette de détection: adcValue + ou - hysteresis
}

boolean Button::pressed(){ // retourne true si basculement détecté
  int buttonValue = analogRead(A0);   // lecture de l'entrée analogique A0
  boolean buttonJustPressed =
    ( buttonValue >= _adcValue - _buttonHysteresis )
    && ( buttonValue <= _adcValue + _buttonHysteresis );
  boolean toggle =
 buttonJustPressed && !_buttonPrevious; // bascule si bouton pressé ET état précédent NON pressé
  _buttonPrevious = buttonJustPressed;  // mémorise l'état du bouton
  if (toggle){
   delay(_bounceDelay); // pause anti-rebonds
   buttonValue = analogRead(A0);   // nouvelle lecture de l'entrée analogique A0
   buttonJustPressed =
    ( buttonValue >= _adcValue - _buttonHysteresis )
    && ( buttonValue <= _adcValue + _buttonHysteresis );
```

ButtonClass
```
    if (buttonJustPressed){
      return (true);
    }
   }
   return (false);
}

Button::~Button(){
}
```

La méthode pressed() retournera true si le basculement de l'état « bouton non pressé » à « bouton pressé » est détecté compte tenu des rebonds qui seront ignorés grâce à la pause anti-rebonds.

L'état du bouton est mémorisé dans la donnée membre _buttonPrevious pour pouvoir être comparé à son état au prochain appel de la méthode.

La détection du changement d'état du bouton se fait au niveau de la ligne :

```
  boolean toggle =
 buttonJustPressed && !_buttonPrevious; // bascule si bouton pressé ET état précédent NON pressé
```

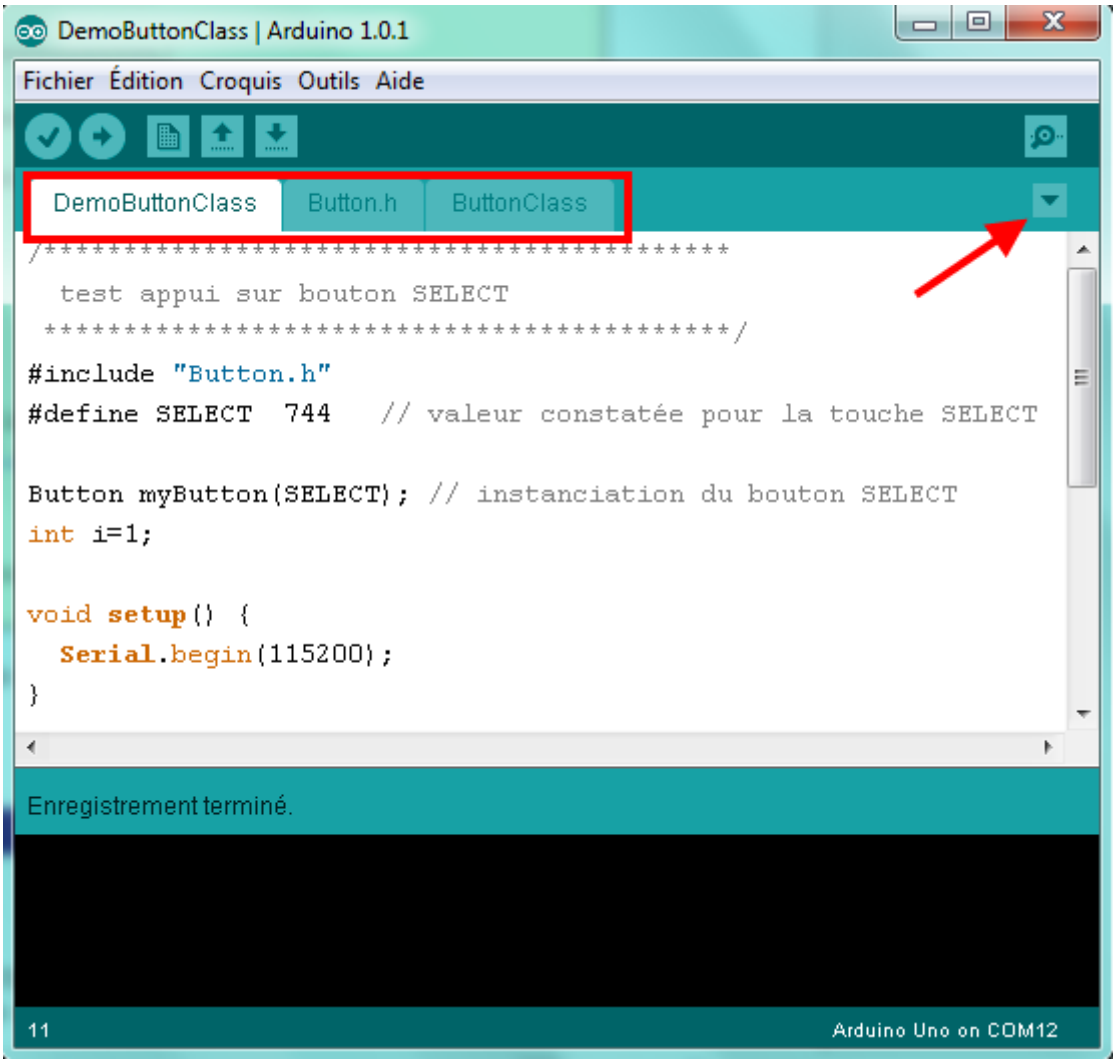
Remarquez ensuite comment, en cas de changement d'état, une nouvelle lecture du port analogique A0 est effectuée après la pause anti-rebonds :

```
  if (toggle){
   delay(_bounceDelay); // pause anti-rebonds
   buttonValue = analogRead(A0);   // nouvelle lecture de l'entrée analogique A0
   buttonJustPressed =
    ( buttonValue >= _adcValue - _buttonHysteresis )
    && ( buttonValue <= _adcValue + _buttonHysteresis );
   if (buttonJustPressed){
     return (true);
   }
  }
```

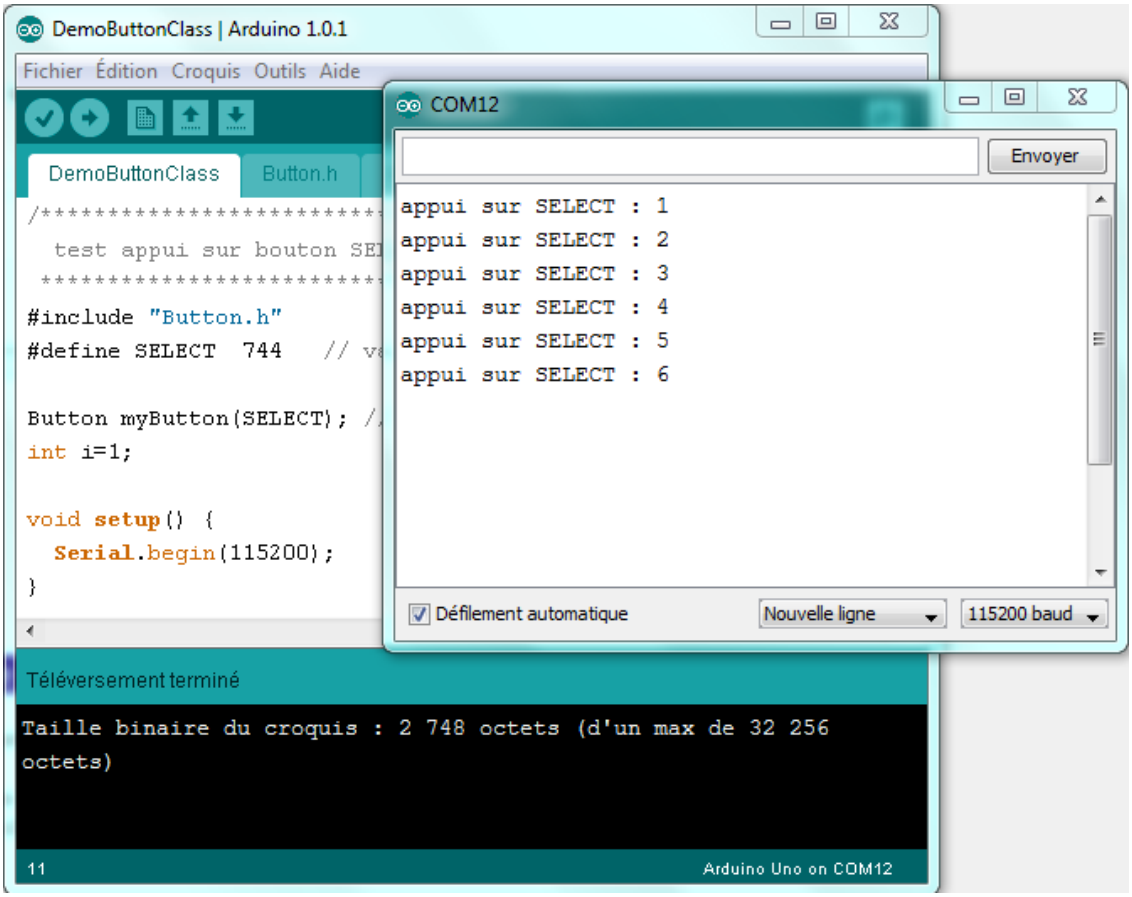
Voici le code d'un programme de démonstration mettant en œuvre la classe Button :

DemoButtonClass
```
/*****************************************
  test appui sur bouton SELECT
 *****************************************/
#include "Button.h"
#define SELECT  744     // valeur constatée pour la touche SELECT
#define BOUNCE_DELAY  5 // pause anti-rebonds = 5 ms

Button myButton(SELECT, BOUNCE_DELAY); // instanciation du bouton SELECT
int i=1;

void setup() {
  Serial.begin(115200);
}

void loop() {
  if (myButton.pressed()){  // si la touche SELECT est pressée
    Serial.print("appui sur SELECT : ");
    Serial.println(i);
    i++;
  }
}
```

Une fois de plus, le programme de démonstration, l'entête et la classe sont rédigés dans des onglets séparés de l'interface :



Vous pouvez télécharger le code dans la carte et tester son fonctionnement dans le moniteur Série (115 200 bauds) :

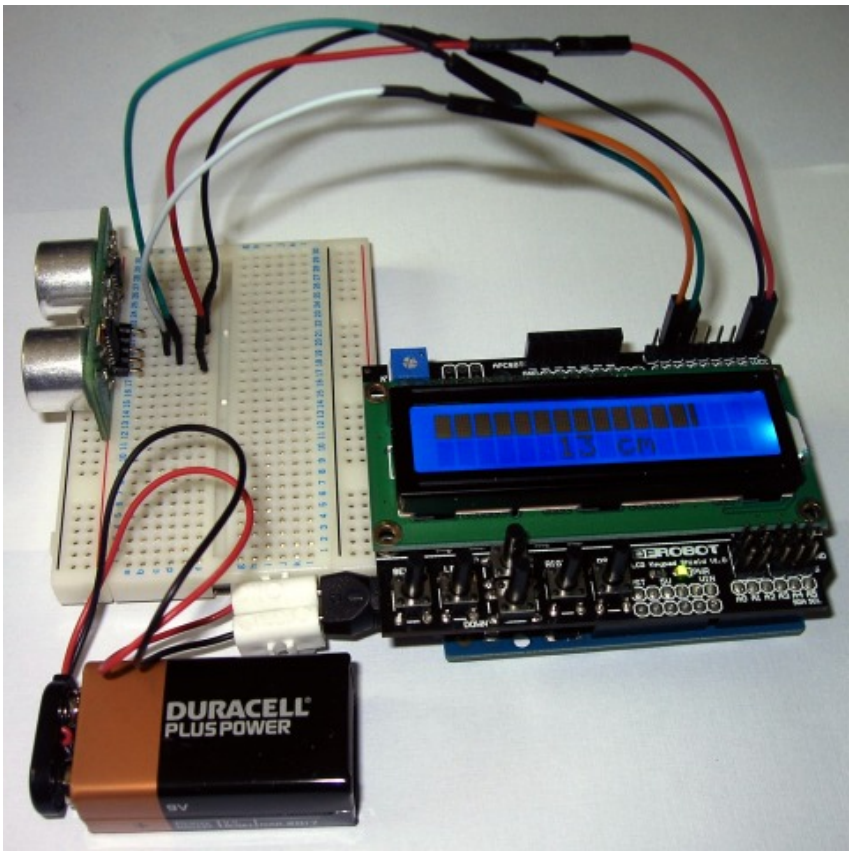


Si vous souhaitez rendre la classe réutilisable et la rendre disponible dans l'IDE, comme pour la bibliothèque NewPing, il vous suffit de suivre les étapes de l'article : **Writing a Library for Arduino**.

VII - La maquette finale

Nous rassemblons les morceaux :

- les distances mesurées entre 0 et MAX_DISTANCE sont retournées. Une petite étoile (*) s'affichera sur l'écran LCD si la mesure n'est pas complétée (distance > MAX_DISTANCE) ;
- la distance est affichée en centimètres par défaut, mais l'appui sur le bouton SELECT permet de basculer d'une unité à l'autre (centimètres ou pouces) ;
- afin de visualiser plus facilement la proximité de l'obstacle, une animation graphique de type bargraphe a été rajoutée.



Une fois le programme téléchargé, le dispositif alimenté par une pile carrée 9 V est autonome.

L'archive du programme complet est disponible ici : **SRF05_LCDKeypadShield.zip**

Le code a été rédigé avec la version 1.0.1 de l'IDE Arduino (Windows). N'oubliez pas d'installer la bibliothèque **NewPing**.

VIII - Conclusion

Voilà une première maquette réalisée à base d'Arduino. Le dispositif affiche la distance mesurée par le capteur sur un écran LCD (20 mesures par seconde) selon l'unité choisie (centimètres ou pouces) grâce à un bouton-poussoir. La maquette a été réalisée pas à pas en affichant les résultats bruts dans le moniteur Série dans un premier temps.

L'IDE offre une interface sobre (coloration syntaxique, onglets, mais pas d'aide syntaxique, d'autocomplétion, de pliage de portion de code) avec de nombreux exemples de programmes de démonstration pour les premiers tests. Le langage, à base C/C++, offre un jeu d'instructions de haut niveau pour accéder aux entrées/sorties de la carte. Dans ce tutoriel, nous avons d'ailleurs exploité les entrées/sorties D11 et D12 pour la gestion du capteur et une entrée analogique A0 pour le bouton-poussoir. Les autres entrées/sorties numériques de la carte que nous avons déclarées sont exploitées en toute transparence par le *LCD Keypad Shield* grâce à la bibliothèque LiquidCrystal.

Un afficheur LCD est un objet normalement complexe à exploiter qui nécessite un circuit électronique spécialisé. Mais sous forme de *shield* avec la bibliothèque logicielle adaptée, le débutant peut rapidement exploiter ses fonctionnalités pour arriver à ses fins. La gestion pointue de l'électronique pourra être étudiée plus tard. On ne doit cependant pas ignorer les aspects physiques, comprendre les caractéristiques électriques, le principe de fonctionnement de ce capteur à ultrasons ou les phénomènes de rebond dans les commutateurs est essentiel. Il en sera de même lorsque vous interfacerez des moteurs électriques, des lampes ou tout autre dispositif complexe.

Dans la foulée de cet article, vous pouvez faire évoluer le dispositif en incluant un signal sonore modulé en fonction de la distance de l'obstacle à la façon d'un radar de recul équipant les véhicules automobiles. Rien de bien compliqué avec un **buzzer** piézo-électrique et la bibliothèque **Tone** pour l'exploiter.

Bref, Arduino et son écosystème sont sans doute le moyen le plus rapide pour développer une application ou mettre au point une prémaquette de système embarqué sans être un spécialiste pointu de la programmation des microcontrôleurs ou de l'électronique. Vous êtes finalement prêt à vous lancer et rejoindre la communauté.

Do It Yourself…

IX - Remerciements

Je remercie **deletme**, **al1_24** et **Bktero** pour leur relecture attentive de cet article et leurs conseils.

Je remercie également l'ami **ClaudeLELOUP** pour sa relecture orthographique.