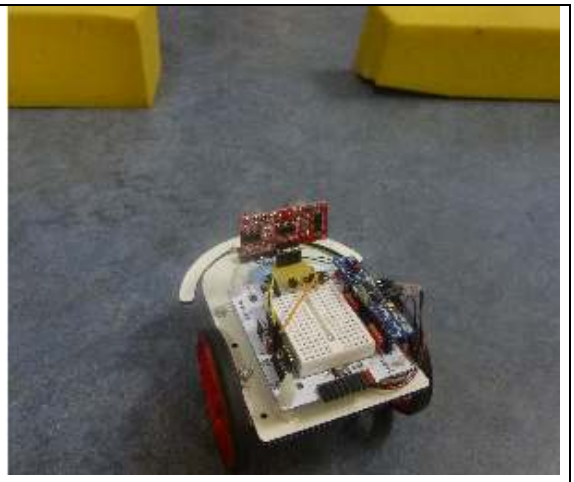
Projet Robot

Objectif
Le robot a 2 roues et peut facilement tourner sur lui-même. Il a un capteur de distance fixe et on lui demande de passer par une ouverture un peu plus large que lui.
Le robot tourne sur lui-même pour mesurer la distance à l'obstacle et avance un peu dans la direction calculée.
Une variante déplaçant le capteur avec un servo permettrait une avance continue avec correction de la trajectoire.



1 - Etapes du projet

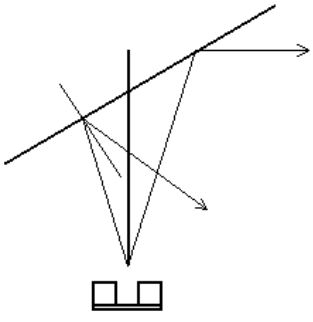
Le projet a été choisi comme exemple final dans le coursera EPFL "Microcontrôleurs".
Une première étape a été programmée par Raphael Voegeli, 15 ans, en utilisant un capteur à ultrasons SR04. Le programme n'étant pas fiable, il a été décidé de comparer avec un PSD, et de commencer par analyser ces deux capteurs.

2 - Capteurs à ultrasons SR04

Les capteurs de distance à ultrasons sont décrits en détail dans le document www.didel.com/xbot/DistSonar.pdf .
Les modèles SR04 et SR05 sont très faciles à obtenir. Une impulsion sur l'entréée TRIG déclenche l'envoi d'un train d'impulsions à 40 kHz. La sortie ECHO est activée pendant la durée de l'aller-retour du son.

L'indication de distance est rarement fiable. Le faisceau sonore a 45 degrés d'ouverture et l'objet le plus proche, même petit, renvoie un écho, sauf si c'est un miroir.

Si le retour du son est trop faible, un time-out doit bloquer l'attente. Avec le SR-05, la durée est limitée à 20 ms (3m) et la sortie OUT est sensée s'activer. Avec le SR-04, il y a un time-out de 200 ms, donc aucune mesure possible pendant tout ce temps. Ce modèle n'est donc pas conseillé.



On peut naturellement arrêter l'attente à 40 ms par exemples, mais le capteur ne redémarrera pas avant d'avoir fini son cycle.

2 - Lecture du capteur

Arduino propose la fonction `pulsin()` pour mesurer la durée de l'echo. Son inconvénient est d'être bloquante, en plus de ne pas être acceptée par un compilateur C qui n'a pas une librairie adaptée.
`pulsin` a 2 ou 3 paramètres: `pulseIn (pin, level)` mesure la durée de la prochaine impulsion positive (level=HIGH) ou négative (level=LOW). Avec `pulseIn(pin, level, timeout)`, le paramètre supplémentaire définit un temps d'attente maximum, qui ne permet pas de lire plus rapidement..

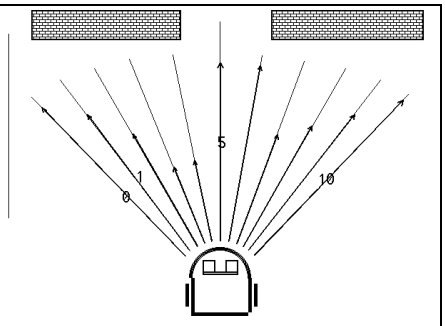
La solution C proposée ci-dessous à droite est toujours bloquante, mais elle est prête pour une variante gérée par interruption. La mesure toutes les 100 us donne une valeur 8 bits, l'unité valant ~16mm. Un obstacle à 1m donne une valeur de ~60; Plus de précision est illusoire. La distance est saturée à 200, environ 3 mètres.

```
//TestASonar  Arduino
#define Trig  15
#define Echo 14

int  tPropa ;   // temps de propag ultrason

void setup ()
{
   pinMode(Trig, OUTPUT);
   pinMode(Pulse, INPUT);
   Serial.begin(9600);
}

void loop ()
{
  digitalWrite(Trig,HIGH);
  delayMicroseconds(20); // min 10 us
  digitalWrite(Trig,LOW);
  tPropa = pulseIn(Echo,HIGH);
  Serial.println(tPropa,DEC);
  delay (1000) ;  // une lecture /s
}
//Taille mémoire 3314
//Taille sans Serial 1518
//Taille sans pulseIn 1180
```

```
//TestCSonar  C - IDE Arduino
#define bTrig  1  // PORTC
#define bEcho  0  // PORTC
#define TrigOn    bitSet (PORTC,bTrig)
#define TrigOff   bitClear (PORTC,bTrig)
#define EchoOn    PINC & 1<<bEcho

byte tPropa ;   // temps de propagation ultrason <250

void setup ()
{
   DDRC |= 1<< bTrig ;
   DDRC &= !(1<< bEcho) ;
   Serial.begin(9600);
}

void loop ()
{
  TrigOn;
  delayMicroseconds(100); // min 10 us
  TrigOff;
  tPropa=0;
  while (!EchoOn) {} // on attend le départ
  while (EchoOn) {  // on compte
    delayMicroseconds(100);
    tPropa++;
    if (tPropa>200) tPropa=200;  //saturé
  }
  Serial.println(tPropa,DEC);
  delay (1000) ;  // une lecture /s
}
//Taille mémoire 2668  //Taille sans Serial 740
```

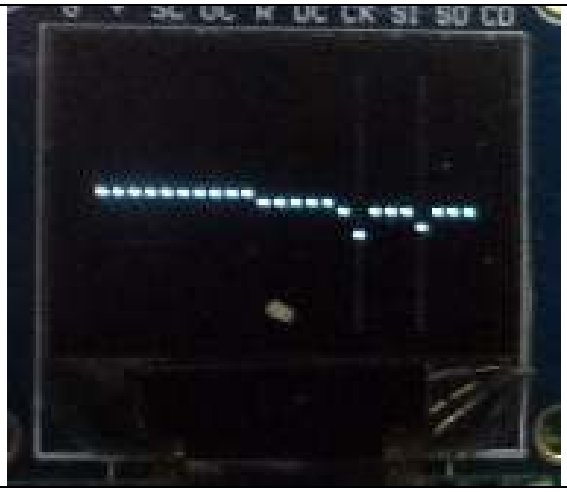
3 - Tester l'environnement

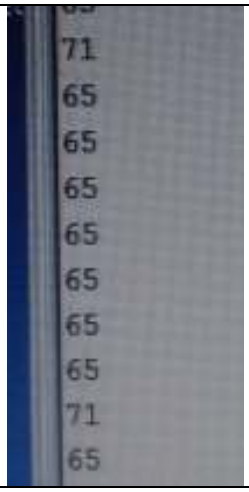
Il faut passer du temps pour bien comprendre les mesures fournies par le capteur, selon l'environnement, Le faisceau sonore a un angle de 30 degrés environ. Un objet lisse (irrégularités inférieures à ~1cm) est un miroir. La distance mesurée peut brusquement changer selon l'angle de l'objet.
Le robot doit passer entre 2 murs. Une bonne démarche est de noter les distances obtenues selon l'angle depuis une grille de positions. Nous allons nous limiter à 24 mesures faites pendant que le robot balaye l'espace. La programmation de ce déplacement est vue plus loin.



4 – Afficher les résultats

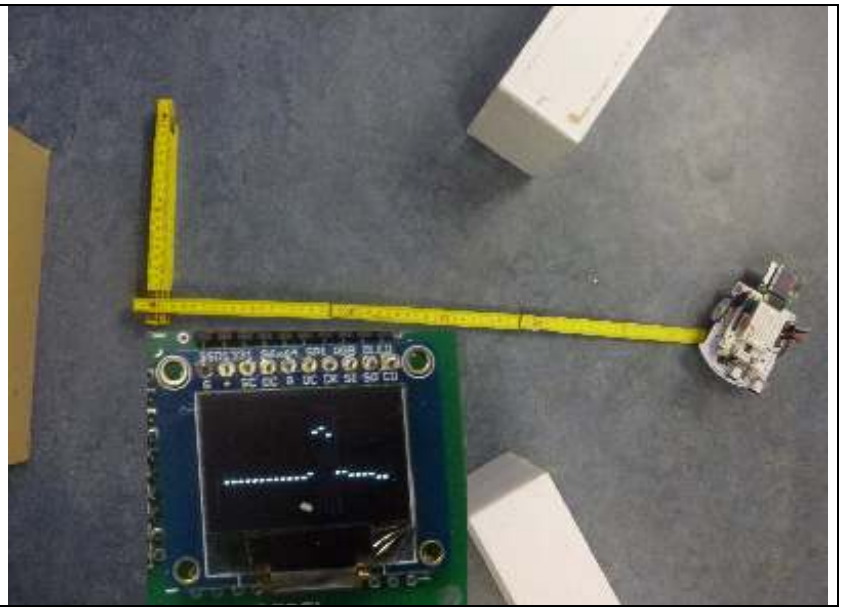
Le terminal permet d'afficher les résultats si le robot a assez de mobilité. Il existe des enrouleurs avec un fil très souple que nous avons utilisé.
Mieux que l'affichage sur le terminal, un afficheur graphique, oled ou autre.

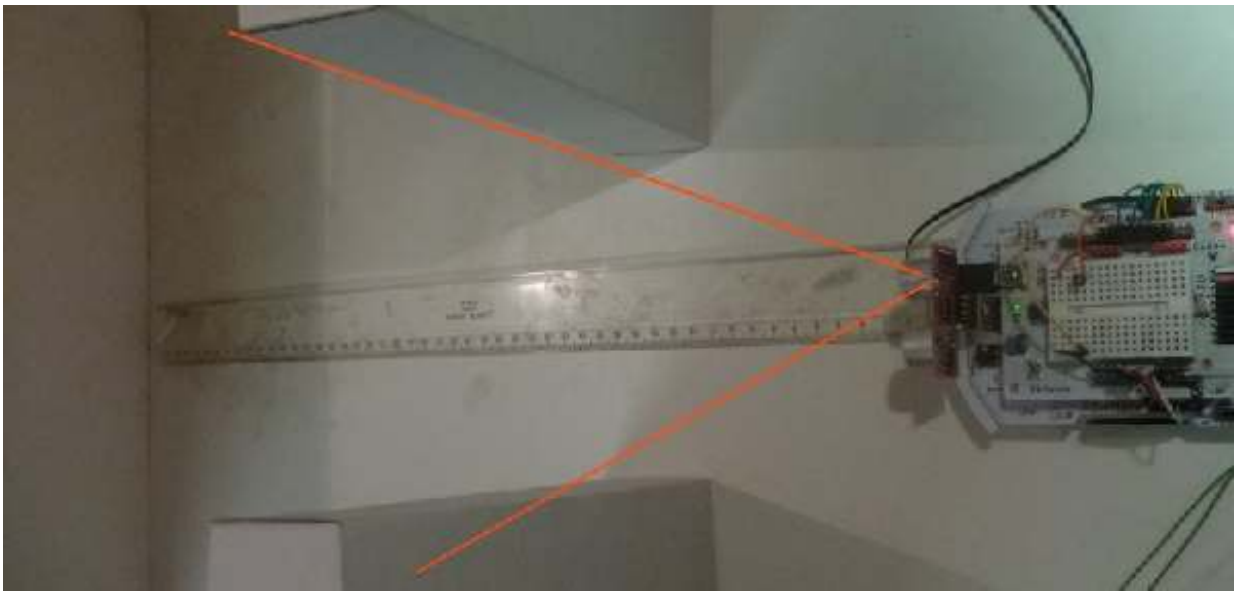




5 – Mesure dans l'environnement

Le robot est placé à 20cm de l'ouverture et tourne de 90 degrés en faisant ses 24 mesures. Le résultat semble utilisable, le capteur voit une augmentation de distance en face. Viser le centre, qui se trouve au pas 18 environ est facile.
Le problème est que cette impulsion est trop étroite. Il y a encore réflexion alors que le robot est bien orienté vers le centre. Reculez le robot à 30cm, il ne voit plus le trou!



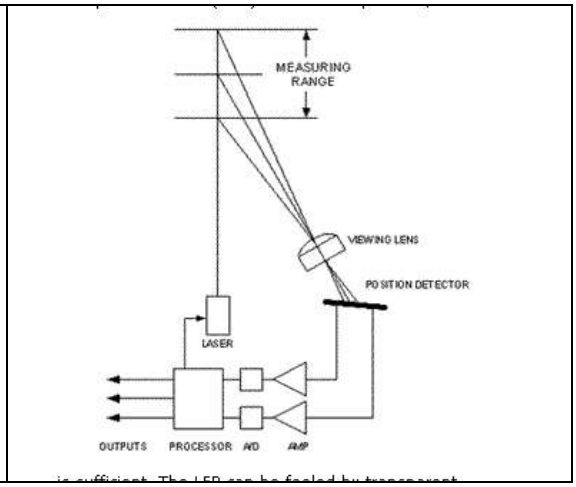


Des caches pour diriger le flux sonore ne sont pas évident. L'intérieur doit être lisse, et l'extérieur ne doit pas retransmettre les vibrations.
En conclusion, le capteur à ultrason est très bien pour mesurer une surface large, mais il a une très mauvaise résolution; il ne ne voit pas le trous et prend les bosses pour des grandes surfaces. Pour ce projet, il doit être abandonné.

6 - Capteur PSD

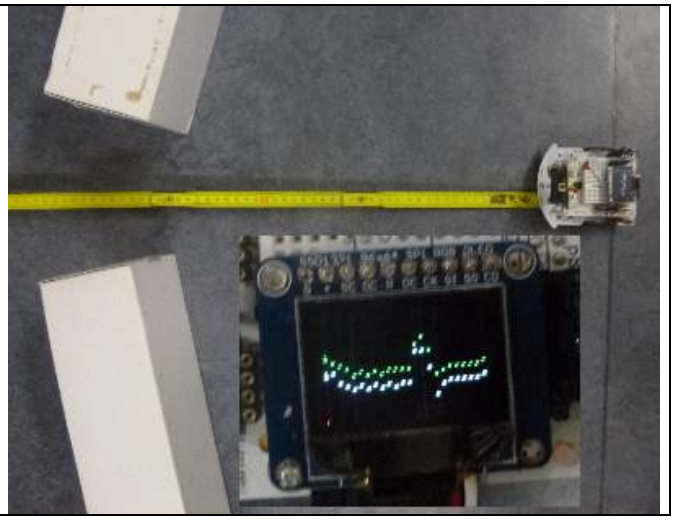
Sharp est le fournisseur principal de ces capteurs, qui ont bien baissé de prix.
Le principe est simple, mais nécessite une optique et un assemblage précis.

Un faisceau infrarouge étroit est émis. Il est réfléchi dans toutes les directions. Des rayons viennent sur la 2e lentille et touchent un capteur PSD (Position Sensitive Device) à un emplacement différent, générant une tension différente. Plus de détails sur internet et www.didel.com/xbot/DistPSD.pdf



Le modèle Sharp GP2Y0A21 est facile à mettre en œuvre, puisqu'il a une sortie analogique. La tension de sortie est de 2V à 20cm et elle diminue à 0.8V à 1m. Avec une alimentation de 3V, la tension est env 20% plus faible.
La courbe n'est pas linéaire, et la mesure de distance supérieures à 50cm est peu précise. En dessous de 12-15cm, la valeur diminue et on ne peut pas savoir si c'est 10cm ou 50cm!
Le capteur plus gros a une meilleur optique, et des résultats plus précis.
Il n'y a pas d'autre capteur de distance de prix raisonnable que nous pouvons envisager.

Le test à 40 cm montre que ce capteur a un angle d'ouverture de ~5 degrés. Sur l'écran, la courbe verte est une moyenne glissante sur les 4 dernières valeurs, décalée verticalement de 8 points.



4 – Déplacer le robot

L'idée est d'avoir le robot qui balaye son espace devant, détermine la bonne direction et avance un peu ou beaucoup avant de se repositionner.

Définissons 3 mouvements: un pas à gauche, un pas à droite, 1 pas en avant répété pour parcourir une certaine distance.
Pour les pas à gauche et à droite on pulse les moteurs pour une durée qui correspond à l'angle voulu. Il faut une attente de même durée pour que le moteur s'arrête, autrement il risque de prendre de la vitesse dans des pas successifs.

```
 // AvTourne Arduino
#define RecG 4
#define AvG 5
#define AvD 6
#define RecD 7

int dStep =10;  // 10ms durée pas
void StepD () {  // pas à droite
  digitalWrite(RecG,HIGH);
  digitalWrite(AvD,HIGH);
  digitalWrite(RecD,LOW);
  digitalWrite(AvG,LOW);
  delay (dStep) ;
  digitalWrite(RecG,LOW);
  digitalWrite(AvD,LOW);
  digitalWrite(RecD,LOW);
  digitalWrite(AvG,LOW);
}
```

```
// AvTourne  C
#define TourneD PORTD |= 0x50
#define TourneG PORTD |= 0xA0
#define Avance  PORTD |= 0x60
#define Stop    PORTD &= 0x0F
// ou doit faire un Stop avant un nouveau pas
int dStep =10;  // 10ms durée pas
void StepD () {  // pas à droite
  TourneD; delay (dStep) ;
  Stop;
}
void StepG () {  // pas à droite
  TourneG; delay (dStep) ;
  Stop;
}
void StepAv (int aa) { // aa=1 vit max
  Avance;  delay (dStep) ;
  Stop; delay (aa);
}
```

5 - Table des distances

Pour scanner, on va faire 12 pas à gauche pour se positionner, puis 24 pas à droite pour balayer

```
// On balaye en mesurant la distance
byte dist;
```

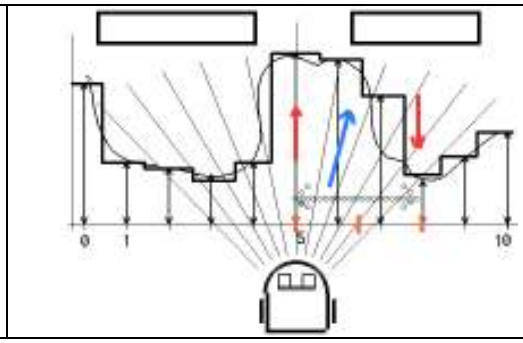
et mesurer la distance. On affiche les distances sur le terminal pour avoir une bonne idée des mesures.
Les mesures sont affichées en temps réel, et stockées dans une table.
Il ne faut pas oublier que l'on fait 24 pas, mais il y a 25 mesures. Dans la boucle for, on mesure après avoir fait un pas. Il faut commencer par ajouter la première mesure.

```
byte tableDist [11] ;
  //dans loop
for (int i=0; i<6; i++) {
  StepG(); delay (5);
}
for (int i=0; i<10; i++)  {
  StepD();
  dist=GetDist();
  tableDist [i] = dist ;
//   Serial.println(dist,DEC);
}
  tableDist [10] = dist ;

for (int i=0; i<6; i++) {
  StepG(); delay (5);
}
```

6 - Représentation

L'objectif est de passer entre les briques avec un premier algorithme qui suppose que le robot est positionné à 50cm -1m devant l'ouverture, à peu près en face. Une fois que cela marchera, on ajustera les paramètres pour être plus tolérant. Ces premiers tests ont été fait avec le capteur ultrasons, qui a le problème de la réflexion des sons latéralement.

Le but est de déterminer où est le centre de l'ouverture, et de diriger le robot dans cette direction.
Partant de la gauche, on détermine une variation positive grande, on mémorise le pas. Idem pour une variation négative importante, en ignorant une éventuelle première diminution de la distance. La moyenne donne un nombre de pas à faire depuis le centre, dans la bonne direction.



La vérification de la validité de cet algorithme et la recherche des bons paramètre se fait sans déplacement, en variant les positions et les angles et en affichant lers valeurs intéressantes sur le terminal.

7 - Calculer la direction à prendre

La solution utilisée dans la vidéo Coursera était basée sur l'algorithme suivant. Ile est apparu en remesurant le capteur ultrason qu'il ne voyait pas bien le trou et que les trajectoires prises étaient aléatiores, donnant parfois l'impression que cela marchait.

On parcourt la table en cherchant la première grande différence positive, puis négative. On mémorise ces deux nombres. Les constantes et variables concernées sont

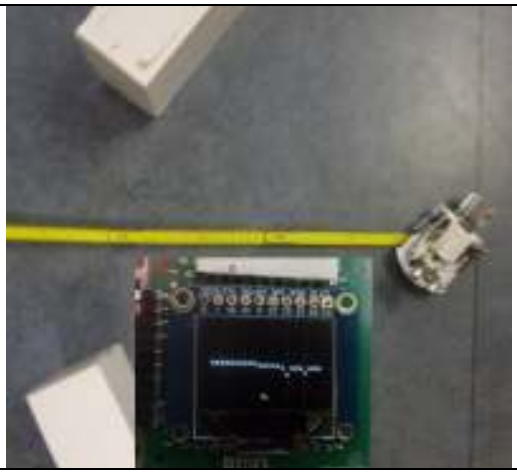
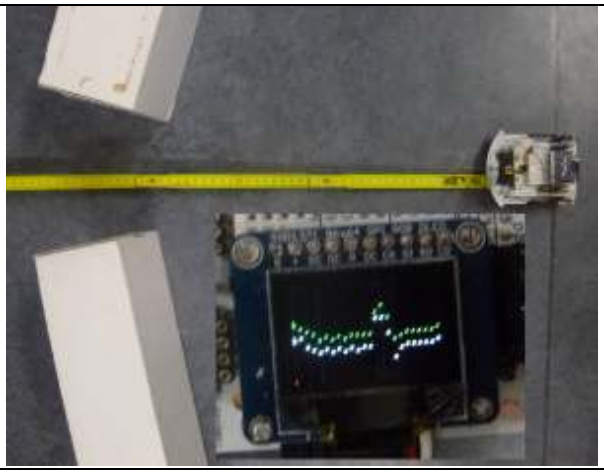
```
#define NbPas = 11   // cas de la figure
#define DiffBord 6  // bruit < 4
byte bordG,bordD;
```

```
 byte TaDist [(NbPas*2)+1] ;
. . . . .
// on attend une bosse
enum {Gau,Cen,Droi} next ;
next = Gau ;
// dans loop()
  for (int i=0; i< NbPas; i++) {
    switch (next)
    case Gau:
       if ((Tadist[i+1]-Tadist[i]) > DiffBord) {
  bordG = i; next = Cen; break;
  }
    case (Cen:
       if ((Tadist[i]-Tadist[i+1]) > DiffBord) {
  bordD = i; next = Gau ; break;
  }
}
```

8 – On recommence

Comme c'est le plus souvent le cas dans les projets nouveaux, il faut recommencer à zéro plutôt que bricoler des améliorations à n'en plus finir.
Le problème était une confiance initiale dans un capteur ultrason, qui ont une bonne réputation pour une utilisation "normale" ou la directivité ne joue pas de rôle. Aucun site ne document la taille d'un objet isolé qui crée suffisamment de réflexion, ni la dimension du trou dont les bords ne créent pas de réflexion.

Une caractérisation scientifique n'étant pas dans nos objectifs, et la visualisation des résultats sur le terminal étant peu efficace, nous avons investi dans l'affichage de l'information sur un Oled de 64x96 pixels. C'est le robot qui est programmé pour balayer sans avancer, et l'observation du signal reçu permet une comparaison facile des capteurs à disposition.

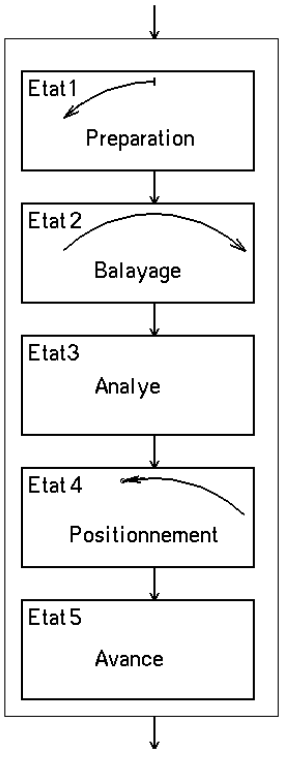
	
Sonar à 40cm – ne voit pas le trou	PSD à 40cm

9. Evolution du projet

Le projet a eu 3 itérations. Une première en croyant que le projet était facile et qui semblait marcher dans certains cas avec le capteur ultrasons. Une seconde avec un capteur PSD et un algorithme qui tenait compte de la variation importante au bord du trou.
La 3e version est expliquée ici; le programme a été ré-écrit avec des procédures pour chaque fonction. Identifier les bonnes procédures et toujours difficiles dans les premières versions des programmes, car la structure générale optimale n'est pas encore connue.
Notons encore que les interruptions ne sont pas utilisées, les tâches à effectuer étant séquentielle. Un gestion des moustache par interruption serait intéressant dans une suite de projet ou le robot chercherait les ouvertures autour de la scène.

10 Décomposition

Le programme démarre avec le robot correctement positionné dans une scène idéale. Dans l'état1 le robot se prépare à scanner. L'état2 fait l'acquisition des distances. Une moyenne glissante des 4 dernières valeurs est effectuée. Le problème d'une moyenne glissante est comment l'initialiser. Notre choix a été de commencer cette moyenne dans l'état1.
Le balayage est suivi de l'analyse de la table des distances, qui permet dans l'état4 d'orienter le robot vers le centre de l'ouverture. Il n'y a plus qu'à le faire avancer et recommencer jusqu'à une condition de fin qui n'est pas évidente.
Les angles sont numérotés de 0 à 24, 12 au centre.
Les définitions et fonctions sont regroupées dans 5 fichiers:
PassePorte.ino Le fichier principal
DefXbot.h Les définitions pour les moteurs et moustaches
Steps.h Les fonctions pour déplacer le robot
Analyse.h Les fonctions pour traiter l'information des capteurs
Pour l'aide à la mise au point, et un aspect spectaculaire du comportement avec l'affichage graphique, deux fichiers sont ajoutés
Oled.h Allège le programme principal de quelques initialisations
Debug.h Les fonctions d'affichage sur terminal et sur Oled.



11 Steps.h

Les pas à droite et à gauche s'appuient sur les définitions du matériel dans DefXbot.h
Avance Recule TourneD TourneG Stop actions continues sur les 4 pins moteurs.
ObsG ObsD valeur booléenes des contacts moustaches, à 1 si un obstacle est touché.

Les pas en rotation sont inspirés du PFM pour être reproductibles. L'angle est fixé par quelques impulsions de 2 ms, séparées par 4ms pour que le moteur s'arrête. L'angle est proportionnel au nombre d'impulsions. Avec une impulsion longue, le moteur accélère et est plus sensible aux frottements.
Par contre, pour faire avancer le robot, une action continue est acceptable (avec l'inconvénient de démarrage-arrêt brusques, que l'on pourrait facilement corriger par quelques impulsions PFM au démarrage.

```
#define Ds 2
#define Ss 4
#define Ns 3   //Nombre de microstep pour un step
void StepD () {
  for (int j=0; j<Ns ; j++) {
    TourneD;
    delay (Ds);
    Stop;
    delay (Ss);
  }
  delay (50);
}
```

StepG similaire

```
void MoveAv (byte nn) {
  for (int i=0; i<nn;i++) {
    Avance;   delay (Ds);
  }
  Stop; delay (Ss);
}
```

Les Etats 1,2,4 sont exécutés dans des fonctions faciles à comprendre

Dans l'état1, on effectue 12 pas vers la gauche avec mesure de la distance et de la moyenne. La distance analogique de 10 bits est divisée par 16 pour avoir des valeurs inférieures à 64, compatibles avec l'affichage, et bien assez précises.

```
void IniScan () {  // va de 12 à 0 Etat1
  // Prépare la moyenne glissante
  for (byte i=0; i<12; i++) {
    StepG ();
    dist = analogRead (A0)/16;
    disGlis = DoMoyGlis (dist) ;
  }
}
```

Dans l'état2, on lit et mémorise dans une table. Une première lecture est effectuée avant d'entre dans la boucle des 24 pas et mesures.
A chaque pas, un affichage est prévu pour les 3 paramètres qui peuvent nous intéresser: la position i, la distance et la moyenne glissante. Le distance ne sera pas utilisée, mais cela peut être utile de comparer.

```
void Scan ()  {  // de 0 à 24 Etat2
  dist = (analogRead (A0))/16;
  disGlis = DoMoyGlis (dist) ;
  taDist[0] = disGlis ;
  AfVal (0,dist,disGlis) ;
  for (int i= 1; i<25;i++) {
    StepD ();
    dist = (analogRead (A0))/16;
    disGlis = DoMoyGlis (dist) ;
    taDist[i] = disGlis ;
    AfVal (i,dist,disGlis) ;
  }
}
```

Dans l'état 4, on tourne du bon angle, et pour avancer un peu, pas besoin d'une fonction.

```
void VaEnPos (int pn)  {  // depuis 24
for (byte i=0; i < (24-pn); i++) {
    StepG ();
  }
```

12 – Analyse.h

Les fonctions pour rassembler l'information et l'analyser dépendent de l'algorithme pour déterminer la direction optimale à prendre. Après avoir observé les courbes obtenues à différentes distances, la solution utilisée est de calculer la moyenne de toutes les mesures, et de faire une moyenne des positions qui ont des valeurs en dessous de la moyenne. L'affichage dont on reparle plus loin est très utile pour vérifier cet algorithme.

Une première fonction permet la moyenne glissante sur 4 mesures consécutives.

```
byte table [4] ;
byte DoMoyGlis (int dd) {
// in nouvelle valeur  // out moyenne
avec 3 valeurs précédentes
  byte mg=0;
  table [3]= table [2] ;
  table [2]= table [1] ;
  table [1]= table [0] ;
  table [0]= dd ;
  for (int i= 0; i<4; i++) {
```

```
      mg += table [i] ;
   }
   return mg/4;
}
```

Calculer la moyenne des valeurs accumulées dans l'état2 est facile.

```
byte taDist [25];
byte DoMoyTable () {
  byte mo;
  int sum = 0;
  for (byte i=0; i<24 ;i++) {
     sum += taDist [i] ;
  }
  return mo = sum/24;
}
```

La fonction GetCap ne peut gérer que des cas simples, avec une variation de distance qui a une forme idéale. On voit que l'on parcourt la table. Si la valeur est inférieurs à la moyenne, on compte et on additionne la position. Par exemple, si les valeurs minimales sont en position 3, 7,8,9,12, `cnti` prend les valeurs 1,2,3,4,5 et `sumi` 3,10,18,27,38 La division 38/5 donne 7,6, donc 7. La valeur 3 parasite a éloigné du centre qui semble être en 9.

```
int GetCap (byte moy) {
  int cap;
  int sumi=0,cnti=0;
  for (byte i=0; i<24 ;i++) {
     if  (taDist [i] < moy ) {
        sumi += i; cnti++;
      }
   }
   cap = sumi/cnti ;
   if (cap<=0) {cap=12; }
   if (cap>24) {cap=20; }
   return cap;
}
```

13 Le programme

Le programme appelle ces fonctions.
Pour la mise au point, des délais ont été ajoutés pour mieux suivre la séquence des opérations.

```
IniScan ();
Scan ();
moyenne = DoMoyTable ();
newPos=GetCap(moyenne);
AfPos (newPos,moyenne);
VaEnPos (newPos);
MoveAv (200);
```

Le problème qui reste est de détecter que la porte est passée. La forme du signal change complètement. Il faut détecter ce changement et appliquer un autre algorithme, avec par exemple le robot qui fait un tour complet pour reconnaître les bords de la porte.
Une détection simplifiée affiche parfois un message et bloque le robot. Ce genre de projet est infini et il faut bien s'arrêter une fois.
Le programme est disponible sous www.didel.com/coursera/PassePorte.zip et une video (8,5 Mega) sous www.didel.com/coursera/PassePorte.MP4

14 L'affichage

L'affichage sur le terminal est utile dans les premières étapes, et en particulier pour mettre au point les routines d'affichage graphique, très démonstratives.
L'afficheur Oled utilisé est documenté avec les fonctions graphiques ajoutées sous www.didel.com/xbot/Oled.pdf