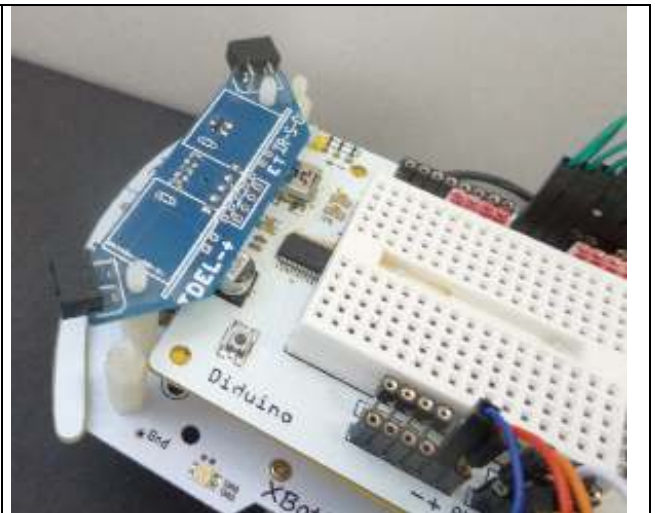


Capteur de distance infrarouge xDist2IR

Les capteurs par réflexion infrarouge ont comme avantage d'être petits, bon marché et faciles à mettre en œuvre. Mais ils sont sensibles à la lumière ambiante (surtout les spots) et sont difficiles à calibrer. Ils ne conviennent aussi que pour des courtes distances, qui dépendent de la taille du capteur, de son optique, de la puissance émise, de filtres éventuels. Des explications générales sont données sous www.didel.com/doc/sens/Doclr.pdf

Un circuit avec 3 capteurs est documenté sous www.didel.com/kits/Dist3lr.pdf avec une librairie en MpLab C documentée sous www.z-control.ch.

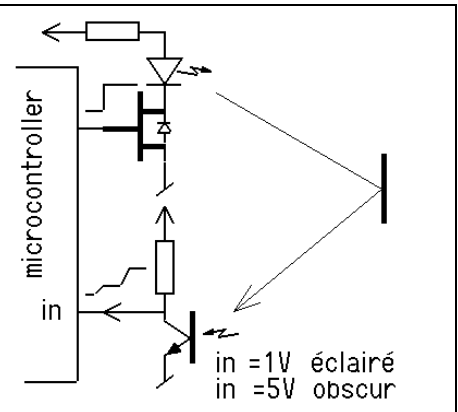
Le capteur Dist2R a été développé pour le XBotMicro, pour mesurer les distances à droite et à gauche. Il est facilement utilisable sur une autre plateforme de robot en tirant 4 ou 5 fils pour remplacer le connecteur en dessous du circuit. L'alimentation 3 à 5V 70 mA quand les diodes IR sont éclairées. La commande se fait par une sortie digitale pour commander l'éclairage IR (optionnel), et 2 entrées digitales pour lire la distance avec une procédure simple. Le module permet d'ajouter un capteur ultrason de type SR05 ou SR10.



Principe

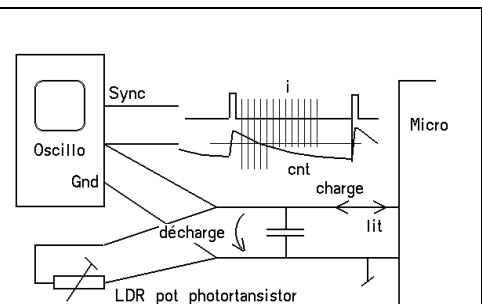
Le principe d'un capteur par réflexion est d'éclairer l'obstacle avec une LED infrarouge, et de mesurer la lumière réfléchie avec une photodiode ou un phototransistor. L'objet éclairé retransmet une énergie inversement proportionnelle au carré de la distance.

Le schéma de câblage d'un détecteur d'obstacle est évident. Une résistance fixe le courant dans la LED qui éclaire l'obstacle. La résistance du photo-transistor est mesurée soit avec un diviseur de tension, soit par mesure du temps de charge ou décharge d'un condensateur, ce qui couvre une gamme de distance plus grande.



Mesure par décharge d'un condensateur

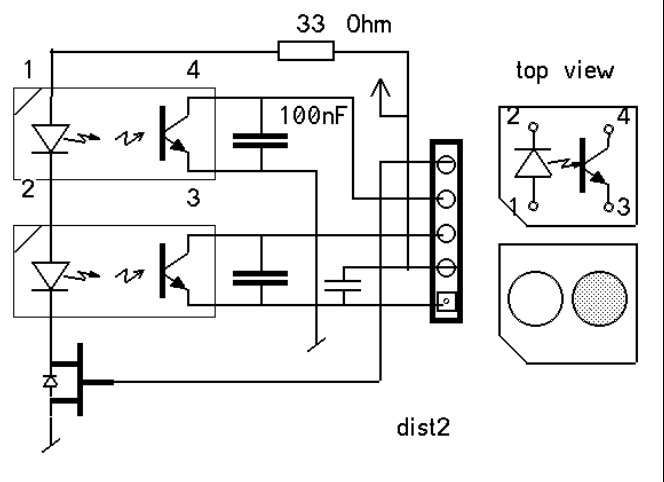
Une solution élégante pour mesurer une résistance variable est de charger un condensateur en parallèle avec la résistance et de mesurer le temps de décharge. Le microcontrôleur charge le condensateur en quelques microsecondes. On commute ensuite le port en entrée, et on mesure le temps de décharge, d'autant plus rapide que la résistance est faible, donc qu'il y a plus de lumière dans notre cas. 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 pas, et un compteur limite la mesure à par exemple une valeur de 100. Si on compte toutes les 100microsecondes, la durée de la mesure est de 10 ms.



Pour diminuer l'effet de la lumière ambiante, on peut faire une première mesure sans éclairage IR, puis une 2^e mesure avec éclairage, La différence corrige un peu, mais ce qui est important, c'est de vérifier que la lumière ambiante donne une valeur très différente de la valeur éclairée par IR. Si non, la capteur est mal dirigé ou doit être protégé par des caches bien placés.

Schéma

Les deux diode IR sont en série, la chute de tension par diode est 1.3V environ. Un transistor commande le courant, qui doit être important. Avec la résistance de 33 Ohm précablée, le courant est de ~70 mA. Le schéma donne la câblage sur les connecteur. On retrouve ces signaux sur le connecteur arrière du XBotMicro. Le câblage court usuel utilise les pins Arduino 14/A0,15/A1,16/A4 utilisées en mode digital.



Un câblage différent oblige à changer les 3 numéros de pins au début du fichier de définition.

Définitions

En plus des définitions des pins et de leurs actions, on définit pour les signaux IR Gauche et Droite, deux directions appelées DirCha (en sortie pour charger les capacités) et DirMes (entrées pour mesurer tester le passage à l'état 0) que l'on devra utiliser pour charger/décharger les condensateurs. Il faut dans le set-up, initialiser les deux bits Dist du PORTC; elles gardent leur valeur quand on passe en entrée.

La documentation du XBot encourage une programmation compatible avec un C portable sur d'autres microcontrôleurs. L'utilisation des notations Arduino est équivalente. Avec un fichier de définition complet, la suite des programmes est compatible.

A noter que dans une macros, la dernière instruction n'a pas de ; final.

<pre>// XBotDistIrDef.h #include <Arduino.h> #define bDistG 0 //PORTC Arduino pin 14/A0 #define bDistD 1 //PORTC pin15 #define bLedIr 2 //PORTC pin16 #define LedIrOn bitSet (PORTC,bLedIr) #define LedIrOff bitClear (PORTC,bLedIr) #define CapaCha PORTC = 1<<bDistG 1<< bDistD #define DirMes DDRC &=~(1<<bDistG 1<< bDistD) #define CapaGHigh PINC & 1<<bDistG #define CapaDHigh PINC & 1<<bDistD void SetupDistIr () { DirLedIr; DirMes; PORTC = 1<<bDistG 1<< bDistD }</pre>	<pre>// XBotDistIrDef.h #define DistG 14 #define bDistD 15 #define bLedIr 16 #define LedIrOn digitalWrite(LedIr,1) #define LedIrOff digitalWrite(LedIr,0) #define DirCha pinMode(DistG,1); pinMode(DistG,1); #define DirMes pinMode(DistG,0); pinMode(DistG,0); #define CapaGHigh digitalRead (DistG) #define CapaDHigh digitalRead (DistD) void SetupDistIr () { pinMode (LedIr,1); DirMes; digitalWrite(DistG,1); digitalWrite(DistG,1); }</pre>
--	---

Logiciel

Pour la mesure, on précharge pendant 100 us, puis commute en entrée, et on lit la valeur qui décroît exponentiellement. Tant que le seuil est supérieur à l'état 1 (environ 1.9V à 5V, 0.9V à 3V) on compte. Il faut définir 3 compteurs et toutes les 100 microsecondes passer par les instructions

```
byte cnt=0, cntG=0, cntD=0 ;
for (i=0; i<MaxVal; i++) {
  if (CapaGHigh) cntG++ ;
  if (CapaDHigh) cntD++ ;
}
```

La distance max est de 100, ce qui fait une durée d'acquisition de 10ms et une distance de 60mm. Durant ce temps, le robot va se déplacer de 1 cm s'il est rapide (1m/s). C'est acceptable. Ce programme est bloquant, pendant 10ms seules les interruptions (donc le PWM) sont exécutées.

On peut augmenter le comptage max à 200, mais cela rajout 8mm à la distance pour un temps de mesure double.

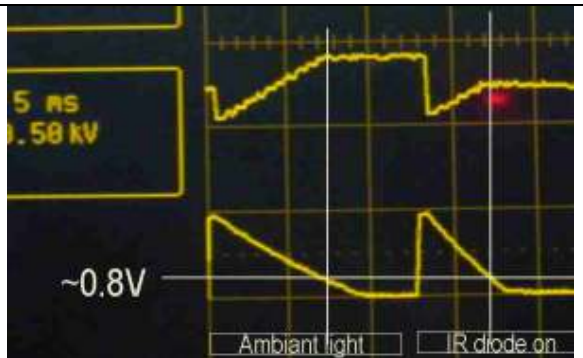
Programme de test

Le programme mesure en continu et affiche les valeurs sur le terminal. Mais on ne va pas afficher toutes le 10m sur le terminal. On compte 100 cycles de 100 us avant d'afficher.

Au lieu du `#include`, on peut naturellement insérer à la place les instructions du fichier de définition. Les programmes se trouvent sous www.didel.com/xbot/xDist2lr.zip

<pre>//TestDist1.ino #include "XBotDistIrDef.h" void setup() { SetupDistIr (); Serial.begin(9600); } // variables globales byte mesureG, mesureD; #define MaxVal 100 // max 250 void GetDist () { volatile byte cntG=0, cntD=0 ; DirCha ; // precharge delayMicroseconds (100) ; LedIrOn ; DirMes ; cntG = 0 ; cntD = 0 ; for (byte i=0; i<MaxVal; i++) { delayMicroseconds (100) ; if (CapaGHigh) cntG++ ; if (CapaDHigh) cntD++ ; } LedIrOff ; mesureG = cntG ; mesureD = cntD ; }</pre>	<pre>volatile byte cntAffi; void loop() { GetDist () ; // tous les 100x 10ms, on affiche cntAffi ++ ; if (cntAffi > 100) { // on affiche cntAffi = 0 ; Serial.print("Distance "); Serial.print(mesureG); Serial.print(" "); Serial.println(mesureD); } }</pre> <p>Test distance max mesurées (valeur 100)</p> <p>papier blanc 70 mm papier noir 72 mm boîte plastique jaune 48mm</p> <p>La distance minimale est de ~10mm → valeur 2</p>
--	---

Dans ce diagramme, le programme a été complété par le transfert de la valeur 8 bits `cntG` sur un port ayant un convertisseur digital-analogique (Microdual DA8). La sortie du convertisseur montre les escaliers de comptage, qui cessent quand la tension du condensateur passe en dessous de 0.8V. Cela montre aussi qu'il faut utiliser la décharge du condensateur, et pas la charge.

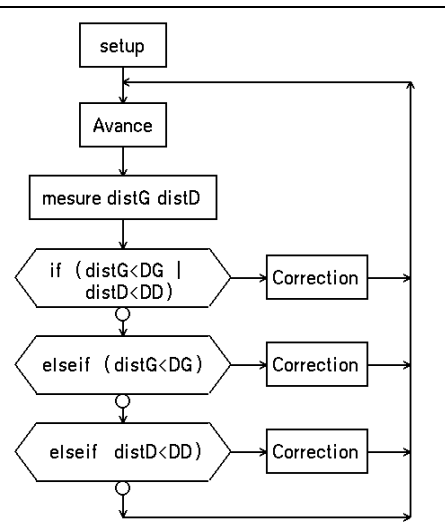


On peut mesurer la valeur diodes IR éteintes en supprimant l'instruction `LedIrOn`. En éclairant avec une lampe de poche, on peut faire descendre la valeur en dessous de 100. Le programme `TestDist1b.ino` affiche les valeurs éclairées et non éclairée.

Eviter les obstacles

Une solution simple inspirée de l'évitement d'obstacle avec les moustaches décide qu'il y a obstacle si la distance est inférieurs à une valeur prédéfinie. On peut commander kles moteurs en tout ou rien, mais une vitesse ralentie par PWM peut être préférable, pour mieux ajuster les paramètres. Dans le programme, il fait ajouter les définition du XBot et avoir compris comment faire reculer le robot avec le PWM d'Arduino sur 2 canaux:

www.didel.com/diduino/CommandeMoteurs.pdf



Interruptions

Pour être appelée toutes les 100 microsecondes par interruption, la fonction GetDist () doit être transformée en une machine d'état. On laisse tomber la mesure obscure.

<pre>volatile byte etat=0; volatile byte cnt, cntG=0, cntD=0 ; void GetDist () { switch (etat) { case 0: CapaCha ; DirCha ; // precharge cntG=0; cntD=0; cnt=0; etat=1; break; case 1: LedIrOn ; DirMes ; cnt++; if (CapaGHigh) cntG++ ; if (CapaDHigh) cntD++ ; if (cnt>100) { LedIrOff ; etat=2; } break; case 2: mesureG = cntG ; mesureD = cntD ; etat=0; break; } // end switch }</pre>	
---	--

Le test avec appel toutes les 100 microsecondes est donné en TestDist3.ino

Utilisons le Timer2 pour créer une interruption toutes les 100 microsecondes, comme expliqué sous

L'appel de GetDist() par interruption prend ~5us, donc ralentit le programme principal de 5% (mais pas la routine delay(), gérée par interruption). Dans cette interruption, on gère le PFM et d'autres tâches si nécessaire. Le programme de test affiche les valeurs lues sur le terminal, toutes les secondes en utilisant un delay(1000).

<p>La routine d'interruption peut maintenant appeler ce module testé sans déverminage supplémentaire.</p> <pre>ISR (TIMER2_OVF_vect) { TCNT2 = 65; // 100 us GetDist (); } void SetupTimer2() { cli(); TCCR2A = 0; //default TCCR2B = 0b00000010; TIMSK2 = 0b00000001; sei(); }</pre>	<p>Le programme est maintenant clair avec les fonctions bien séparées.</p> <pre>//TestDist4.ino Distance par interruption #include "XBotDef.h" #include "XBotDistIrDef.h" #include "XBotDistIr.h" #include "ISRTimerDistIr.h" void setup() { SetupXBot (); SetupDistIr (); SetupTimer2 (); Serial.begin(9600); } void loop() { delay (1000); // tous 1s, on affiche Serial.print("Eclaire "); Serial.print(mesureG); Serial.print(" "); Serial.println(mesureD); }</pre>
--	--

Suivi de mur

On a tous les outils pour programmer des déplacements du robot. Prenons l'exemple de suivre un mur. Il faut asservir le pfm sur la distance du mur droite. La valeur mesurée est entre 20 et 100 (10 à 70mm). Si elle est supérieurs à 50 (loin), il faut ralentir le moteur droite. inférieur à 40, il faut l'accélérer.

--	--