SOMMAIRE

1 introduction :

_ Présentation du concours

Le challenge inter IUT, réservé aux étudiants d'I.U.T. GEII, se déroulera à Vierzon du 06/06/03 au 09/06/03 et est organisé par Car Tch-Inno.

Ce concours se veut avant tout un challenge entre étudiants. Il se déroulera sous forme de passage simultané de deux robots qui doivent évoluer de façon totalement autonome, c'est à dire sans aucune aide extérieure.

Le design est libre : il donnera lieu à un prix du design, indépendamment de la course.

_ Présentation du règlement (CF Programme Car Tech-Inno)

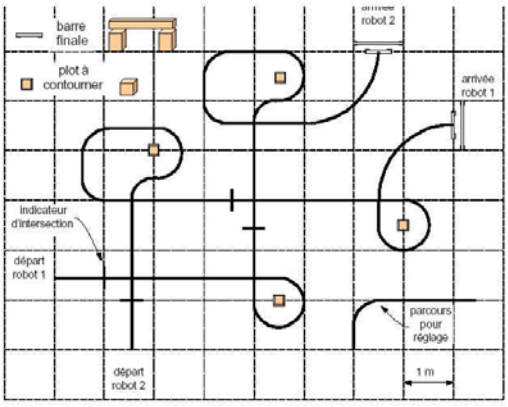
Le robot sera construit à partir d'un kit imposé par le comité d'organisation, comprenant le châssis, les moteurs, les roues et la batterie.

Le robot doit être capable de suivre une piste le plus rapidement possible. En cas de risque de collision entre deux robots, la règle est la priorité à droite.

A la fin de la piste, il faut faire tomber une première barre et laisser en place une seconde barre, distante de 20cm de la première.

Le but est de finir le premier, si un des robots ne respecte pas ces règles il recevra des points de pénalités traduit par des temps supplémentaires.

Exemple de parcours :



_ Résultat envisagé

Notre but est de participer au concours avec un robot le plus opérationnel possible. N'ayant aucune expérience dans ce domaine nous espérons tout de même ne pas finir dernier !!!

2 Cahier des charges :

Le cahier des charges est à séparer en deux parties. Premièrement la «partie imposée » par le règlement du concours, ce qui permet de créer une compétition équitable, sur des bases identiques, basée seulement sur la programmation et l'électronique. Ensuite la «partie libre » dans laquelle l'imagination de chacun entre en jeu.

_ La partie imposée par le règlement

Le robot est constitué d'une «base roulante universelle » de 240mm*240mm*70mm.

Équipé de deux moteurs à courant continu de fabrication spéciale, dont les caractéristiques sont proches du moteur G42x25. Tension nominale 12 Volts pouvant être alimentés en 15V (tension maximale), le courant en charge nominale est de 1.45 A avec un Imax égale à 11 A. Sa vitesse nominale est de 2640 Tr/mm à 12 Volts et 3300 Tr/mm à 15 Volts.

« moteur.tif »
Fabriquant dunkermotoren
Distributeur MDP

La partie puissance doit être alimentée par une batterie au plomb de 12 Volts d'une capacité 1,2 A.H.

Constructeur YUASA
Référence constructeur NP 1,2-12
Référence Farnell 147 472

_ La partie libre

Le cahier des charges qui nous a été présenté en début d'année était le règlement du concours.

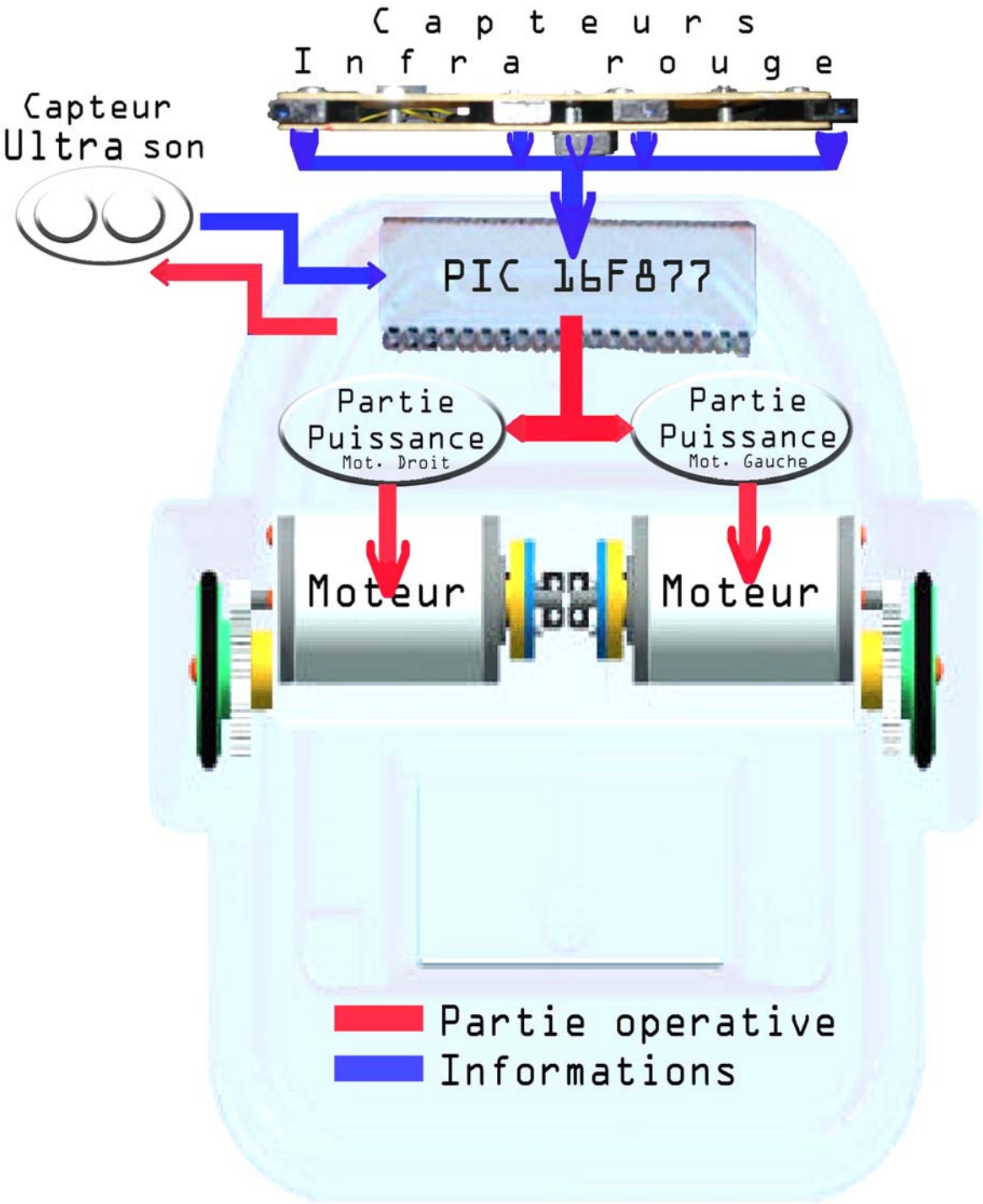
C'est à dire qu'il fallait tout d'abord suivre une ligne blanche avec la plus grande vitesse possible. Nous avons opté pour un PIC 16F877 pour commander le robot, et trois capteurs (phototransistor) pour détecter la ligne, et pour la partie puissance des commandes motrices d'un pont en H : L6203 (full bridge driver for motor control).

Ces composants ayant été testés et validés par un étudiant allemand en stage dans notre IUT durant l'année 2001/2002.

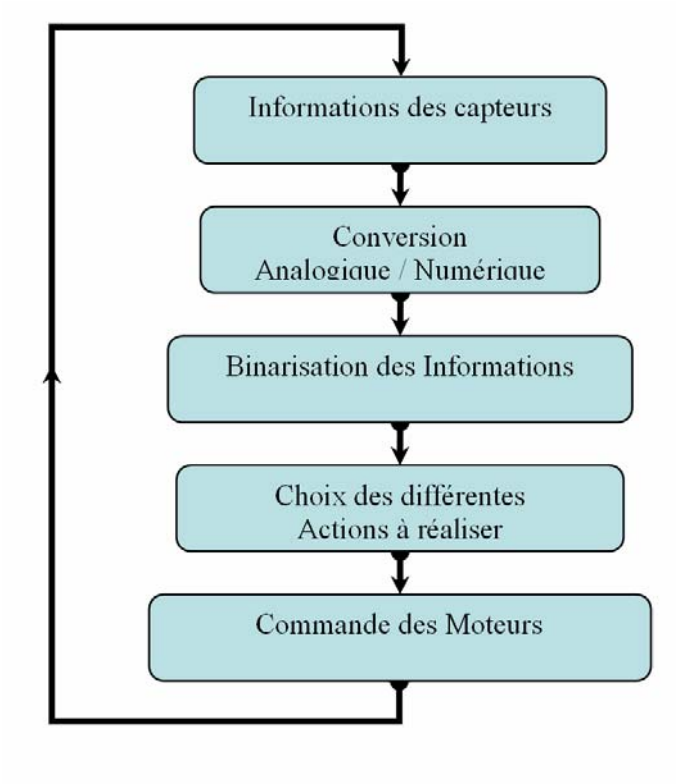
Par la suite nous avons fait des modifications et des ajouts, tel qu'un télémètre à ultra son, pour détecter la priorité à droite, un «pare choc » pour détecter la fin du parcours et deux capteurs supplémentaires pour les raccourcis et pour les croisements. Nous envisageâmes ensuite d'installer une web-cam à la place des capteurs infrarouges afin d'améliorer la rapidité d'action du robot par anticipation.

3 Description, utilisation et problème rencontré avec les divers éléments.

_ Schéma synoptique du robot



_ Algorithme de fonctionnement général



_ Avant le concours

Partie mécanique :

~ Base roulante :

Le châssis est en plastique thermoformé blanc.
- Dimensions (mm) : 240 x 240 x 70
- Masse à vide : 2 Kg (avec moteur et batterie)
- Charge utile : 2 Kg
- Résolution théorique avec quadrature : ±0.24mm

~ Carrosserie :

La carrosserie est en fibre de verre et polyester. Elle supporte le télémètre à ultra son, le pare chocs, le jack de démarrage et l'interrupteur d'arrêt d'urgence.
- Dimensions (mm) : 250 x 360 x 150
- Masse à vide : ~ 500 g

~ Moteur

Deux moteurs à courant continu alimentés en 12V.
-Inom= 1.45 A avec Imax 11 A
-2640 Tr/mm à 12 Volts et 3300 Tr/mm à 15 Volts
- Vitesse linéaire max. :1,3m/s
- Diamètre des roues : Ø50mm
- Voie : 230 mm
- Effort de poussée : 30 N (»2 Kg)

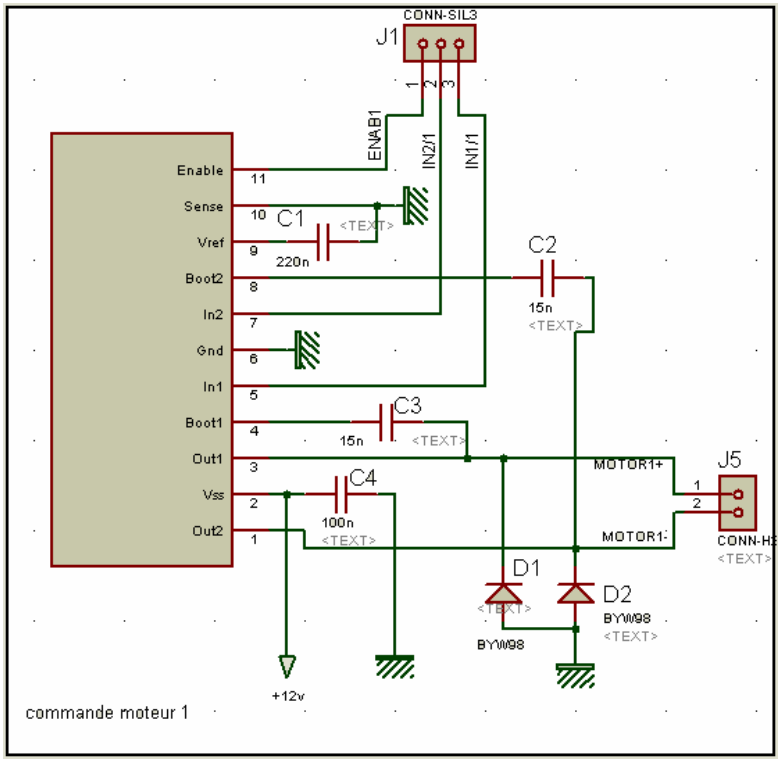
« moteur.tif »
Fabriquant dunkermotoren
Distributeur MDP

Partie électrique

~ Carte mère

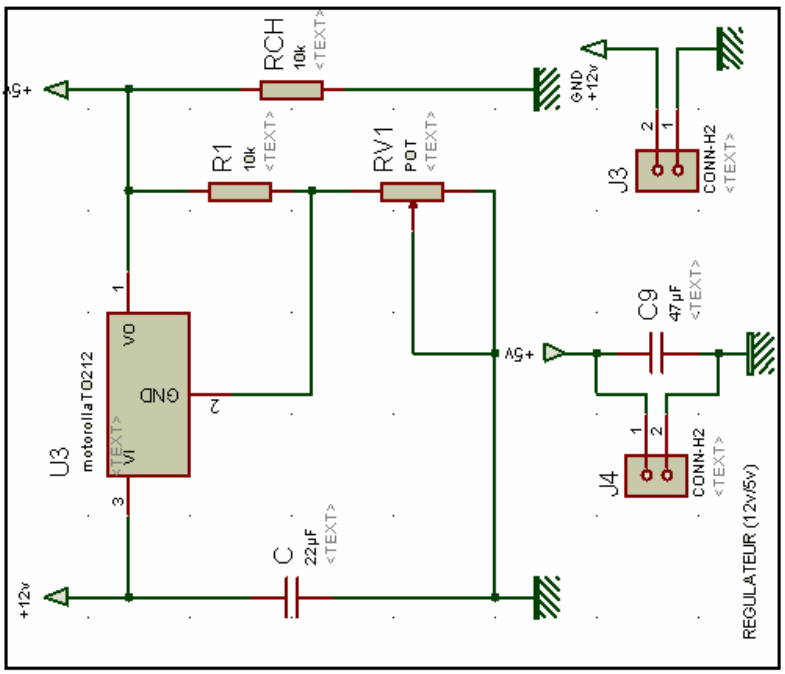
Cette carte contient la partie puissance. Elle permet d'alimenter le micro-contrôleur et les capteurs en 5V et gère l'alimentation des moteurs en 12V.

. Commande moteurs



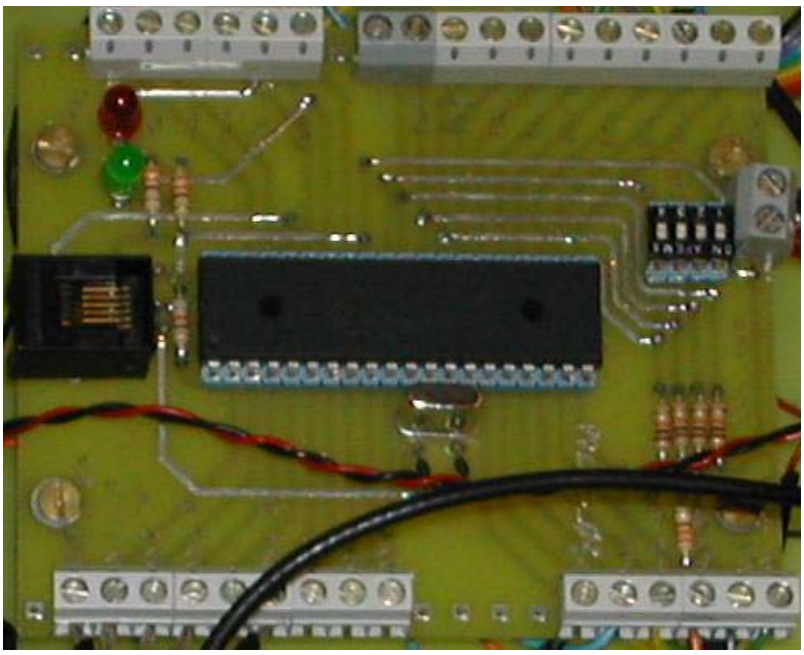
. Régulateur

Il nous permet de générer une tension de +5V pour le PIC.



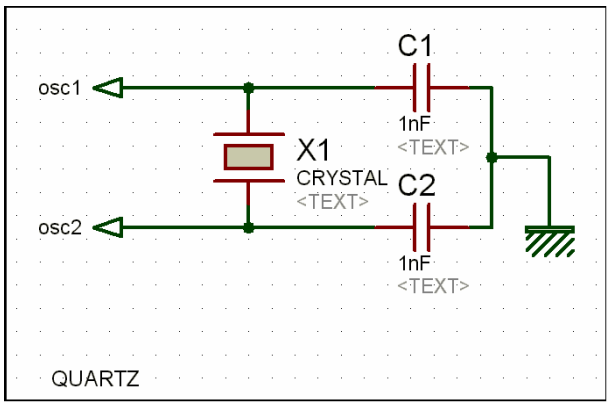
~ Carte micro

Elle supporte le micro contrôleur avec son quartz ainsi que de multiples borniers pour toutes les entrées sorties et connecteurs pour la liaison avec l'ICD*.



. Oscillateur

Le microcontrôleur travaille à une fréquence de 20Mhz. Pour générer ce signal nous utilisons un quartz de 20Mhz selon le montage suivant :



. PIC 16F877

Nous utilisons ce microcontrôleur pour commander le robot, car il peut créer deux signaux carrés de rapport cyclique variable, il gère une liaison I2C, il travaille à une fréquence de 20Mhz ce qui est relativement rapide. Et comporte une importante mémoire.

→ Signal PWM (cf. p. 57 de l'annexe 1).

On le retrouve sur les sorties RC1 et RC2.
Pour créer un signal PWM il faut choisir le mode PWM en écrivant dans le(s) registre(s) CCP1CON (CCP2CON) (cf. p. 58 de l'annexe 1). On précise la période du signal souhaité en écrivant dans PR2 valeur a calculer selon la formule : PWMperiode= [PR2+1]*4*Tosc (cf. p.62 de l'annexe 1)

Dans notre cas PR2=82, nous avons calculé PR2= 100 % du rapport cyclique du signal de 15khz (fréquence à laquelle les moteurs ne vibrent pas d'après le stagiaire allemand).

Pour le mode PWM on indique la valeur du temps haut voulue dans CCPR1L (CCPR1L). Puis on observe le signal sur RC1 (RC2).

Nous générerons les deux signaux grâce à la fonction setdutycycle() qui se trouve dans le fichier PWM.c.

→ Convertisseur A.N. (cf. p. 111 de l'annexe 1).

Pour convertir une tension analogique, il faut configurer le port A en entrée, choisir le canal à convertir, démarrer l'horloge et le convertisseur en écrivant dans ADCON1. On configure les interruptions avec les bits ADIF, ADEI et GEI (cf. p. 117 de l'annexe 1).

Ensuite il faut attendre un temps requis (cf. Equation 11-1 p. 111 de l'annexe 1), qui est d'environ 20μs. Démarrer la conversion attendre le temps d'acquisition et lire la valeur dans ADRESH ADRESL. Nous avons choisi de travailler seulement sur 8 bits donc notre résultat se trouve dans ADRESH (nous nous ne préoccupons pas des bits de poids faible : ADRESL(cf. REGISTER 11-2p. 112).

Pour convertir nous utilisons la fonction convcapt() dans le fichier CAN.c.

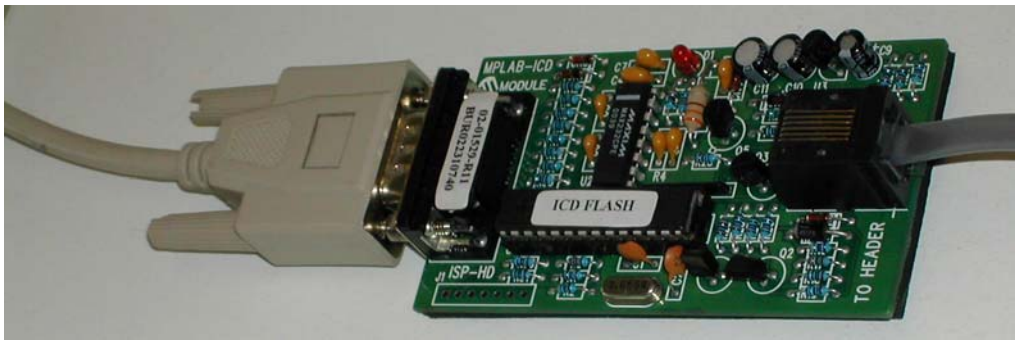
→ Liaison I2C (cf. p. 71 de l'annexe 1).

Pour la liaison I2C nous utilisons les fonctions dans I2C.c

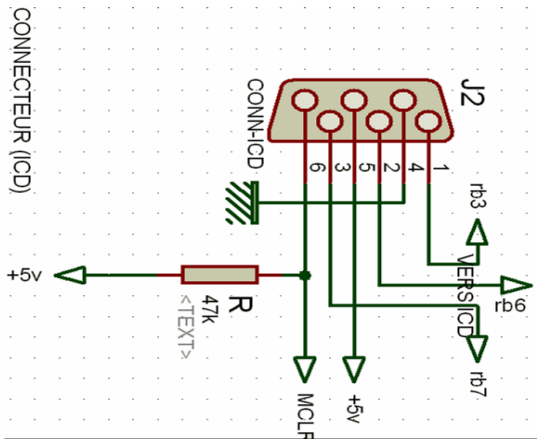
. Liaison ICD

Cette liaison permet de programmer et de discuter (debug mode) avec le PIC par l'intermédiaire du logiciel MPLAB

Carte ICD :

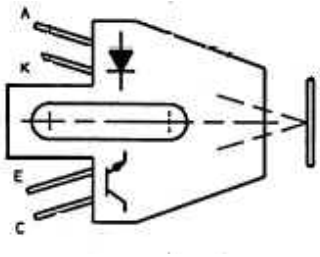


Connecteur :



~ Carte capteur de piste (OPB704)

Elle supporte cinq capteurs infrarouges alimentés en +5V. Ces capteurs sont utilisés pour détecter le scotch blanc grâce à l'émission et à la réception de lumière infra rouge.

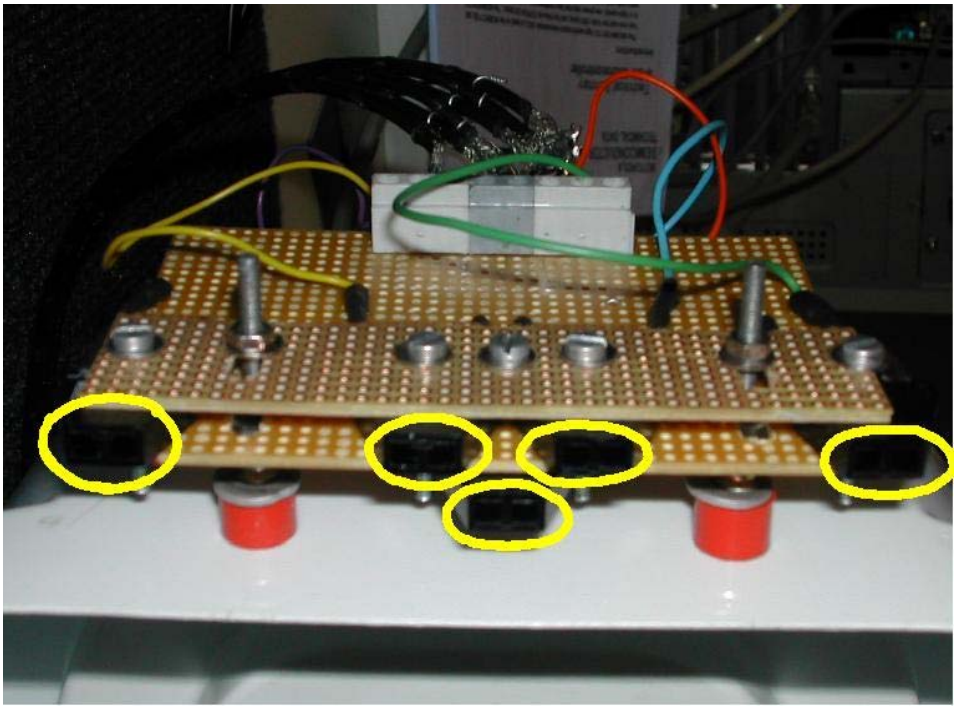


OPB704

Nous avons été confrontés à plusieurs problèmes. La détection ne fut pas facile car le sol sur lequel nous avons fait les essais était plus ou moins gris, la valeur du seuil choisie afin de définir si oui ou non le robot se trouve au-dessus de la piste a été trouvée à la suite de multiples essais. Cette valeur fut confirmée par la suite lorsque nous avons pu faire les essais sur la moquette.

Nous disposons de cinq capteurs dont trois servent pour le suivi de ligne et les deux autres pour détecter les croisements et les raccourcis. Pour éviter d'utiliser le convertisseur analogique (donc pour gagner du temps et obtenir une

valeur instantanée) nous utilisons des trigger de Schmitt afin de déterminer par un +5V ou un 0V (valeurs logiques) si le robot est sur la piste ou non.



~ Carte télémètre à ultra son (SFR08)

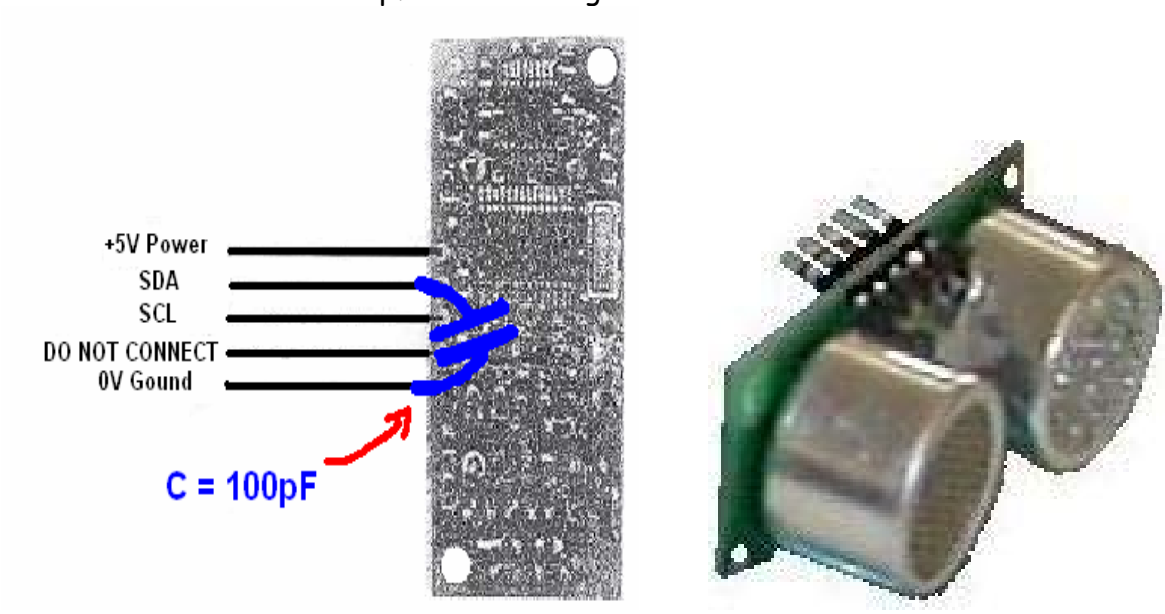
C'est un système complet et autonome, il possède sa propre horloge. Le télémètre à ultra son permet de mesurer une distance entre 43mm et 11m (11 008mm). Il communique par liaison I2C. Il renvoie directement des valeurs en mètres, pouces et microsecondes, jusqu'au 17ème écho. Il est complété par un capteur lumineux qui est utile pour vérifier le bon fonctionnement de la liaison I2C et du module.

Ce capteur nous est nécessaire pour vérifier la présence d'un autre robot lors d'une intersection avec priorité à droite.

Location	Read	*Write*
0	Softwear Revision	*Command Register*
1	Ligth Sensor	*Gain Register*
2	1st Eco Hight Byte	*Rang Register*
3	1st Eco low Byte	*N/A*

~	~	~
34	17st Eco Hight Byte	*N/A*
35	*17st Eco Hight Byte*	*N/A*

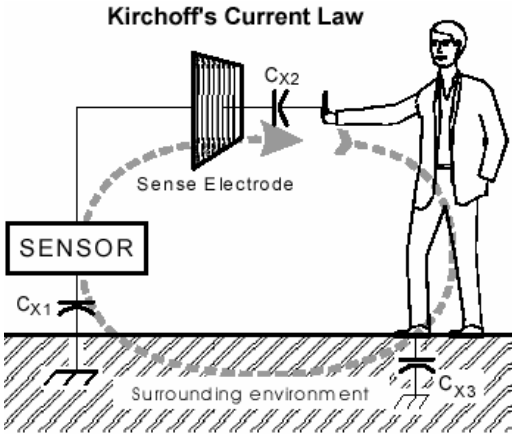
Dans un premier temps nous n'arrivions pas à établir la liaison entre le télémètre et le µc, car le télémètre, étant fixé sur le toit de la carrosserie, engendrait trop de pertes dans le câblage de l'horloge. Il fallait placer un condensateur de 100pf entre l'horloge et la masse.



(SFR08)

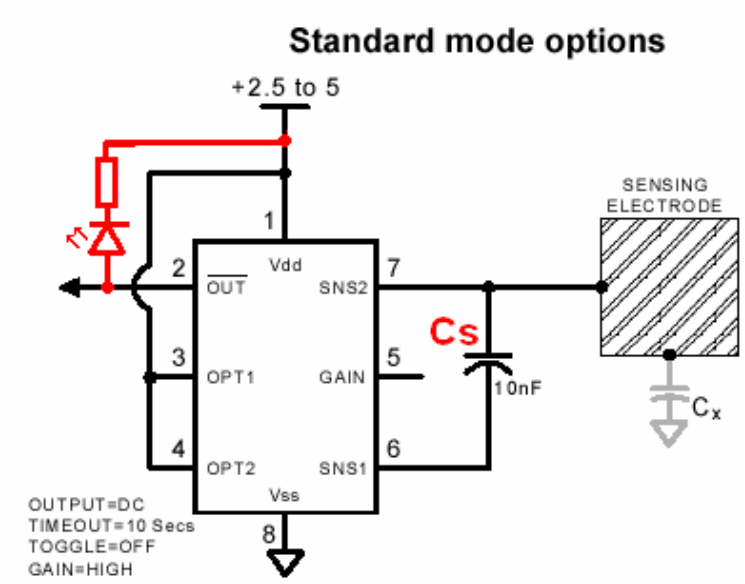
~ Carte capteur tactile (QT110)

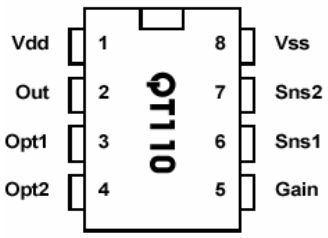
Ce capteur est capable de détecter une proximité ou un contact par une variation de la capacité d'une électrode.



Principe de fonctionnement du QT110

Ce système est auto calibrée, cela évite d'avoir à le régler pour tout type d'utilisation. Un réglage de la sensibilité est tout de même possible, pour cela il suffit de changer la capacité Cs.

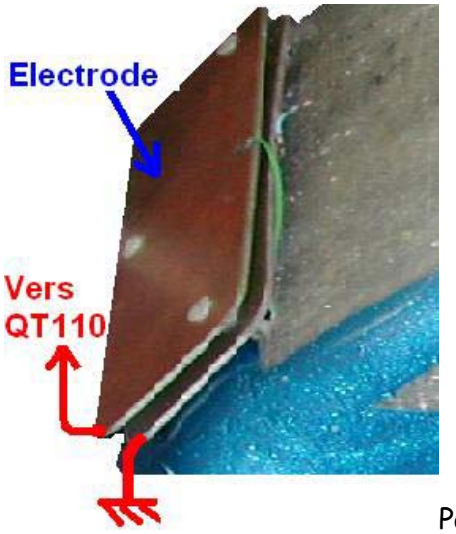




La figure précédente nous montre le schéma de câblage le plus simple. Nous avons rajouter une diode afin d'observer la valeur de la sortie qui est soit de +5V soit de 0V.

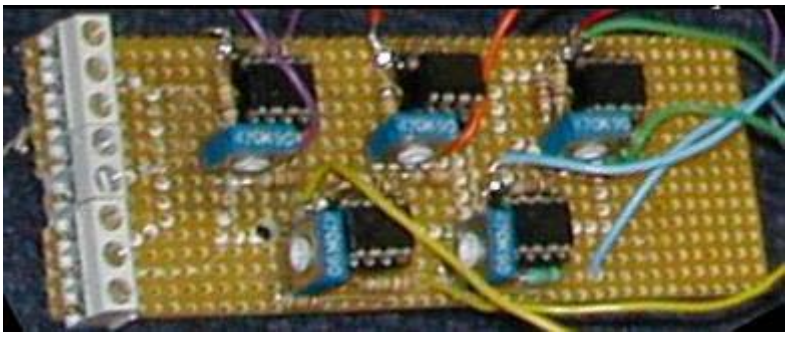
Ce détecteur nous est utile pour la détection de la barre en bois situer en fin de parcours. Nous pouvons le câbler directement sur le PIC mais son signal de sortie n'étant pas réellement parfait, le PIC considère par foi que le capteur a détecté un objet et vie et versa. Ceci nous pose problème, car programmé sous IT (Interruption) cela forcé le programme sur la fonction d'arrivée, donc le robot coupe les moteurs. Pour remédier à cela il reste la solution du capteur de fin de course qui lui fonctionne comme un interrupteur mais il est plus complexe à installer car seule la partie mobile est de ~10cm par ~0.5cm.

Pare choc.



Pare choc.

~ Carte AOP



Cette carte a été développé afin d'améliorer les temps de réponse du robot vis-à-vis de la correction des virages. Elle permet de fournir directement au PIC une valeur logique au lieu de passer par le convertisseur analogique. Elle n'a pas été retenu dans la configuration final pour les capteurs servant au suivi de ligne, mais pour les capteurs optionnels : raccourci et croisement.

Ce sont cinq AOP montés en comparateurs de tension avec un Hysteresis qui permet de régler le seuil analogiquement à l'aide des résistances variables.

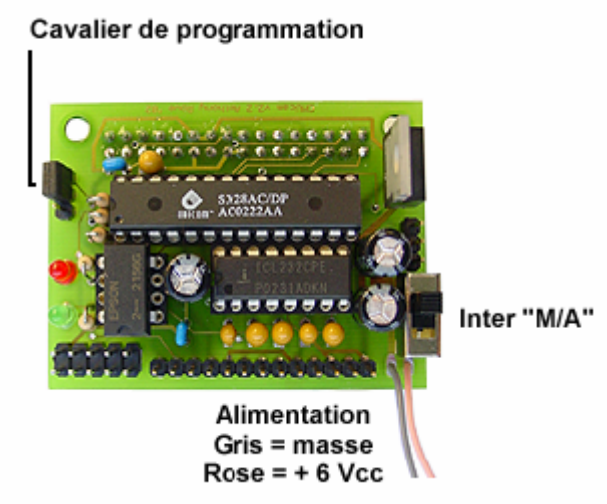
~ Web-cam

Elle a pour utilité d'améliorer les temps de réaction du robot.

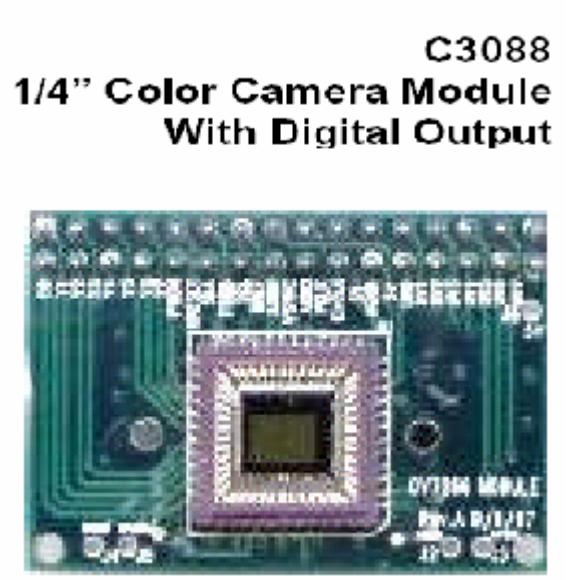
Commandé via Internet, elle a pour fonction d'analyser la piste suite à un traitement d'image. Elle remplace donc la carte des capteurs de piste.

Elle piloté par une carte qui contient le micro processeur C3088. Le tout étant indépendant.

Carte :



Micro processeur :



Les plus :

Le grand plus de ce système est l'anticipation, car si on peut découvrir la piste avant d'avoir « le nez dessus », on pourra répartir les corrections de vitesse plus harmonieusement.

Elle fourni directement les corrections nécessaires. Elle dispose pour cela d'une sortie permettant de commander un cerveau moteur (CF Annexe Camera).

Programmé et testé sur le port série, directement reliée à un ordinateur et gérée par l'Hyper Terminal ou les commandes qu'offre MS-DOS, elle a le grand avantage d'être entièrement autonome.

Le traitement d'image étant fait par son propre micro contrôleur, cela allége de beaucoup notre PIC.

Les moins :

Développer en Java, c'est là où résidé le plus gros problème vu qu'aucune formation nous a été fourni (hors mis les documentations et les cours récupérés sur Internet (CF fichiers zip : cours de java)).

Avec une cadence de 6 images par seconde, cela reste limite.

Une alimentation spécifique comprise entre 6 et 7volts.

Les vibrations engendrées par le déplacement du robot laisse place à quelques centimètres d'incertitude.

Mise en place :

Tout d'abord il faut lancer le logiciel fourni avec celle-ci afin de régler le focus de la camera. Pour cela il faut tout simplement tourner l'objectif, faire des prises de vue, et vérifier si l'image obtenue est correcte (absence de flou).

Puis positionnement sur le chassi. Attention, il faut qu'elle soit exactement alignée avec l'axe du robot.

_ Pendant le concours

Partie mécanique :

Lors du concours, notre carrosserie c'est avérée trop lourde alors on la supprimée. L'inertie du robot était trop importante, donc le robot oscillait sur la piste.

Par ce fait, il a fallu trouvé un moyen rapide et efficace pour remplacer le par-choc. On a donc acheté, chez Farnell, un micro capteur type fin de course. Placé à une distance de 10cm devant le robot, il fut plus efficace que le par choc. Evitant un réglage fastidieux pour les enclenchements parasites dû aux vibrations et secousses que le robot connaît lors de ces déplacements.

Un problème avec une roue, la liaison par encastrement ne tenait plus, nous obligea à changer celle-ci, du même coup, nos réglages du départ en aveugle fûent modifiés.

Partie électrique :

On a supprimer le capteur à infra son qui communique par liaison i2c avec le PIC. Ayant des problèmes de dialogue avec celui-ci, cela ne nous permettait pas d'être fiable pour le parcours du robot et nous faisait rentrer en mode d'attente permanent.

Stratégie :

Comme dit au par avant, on a opté pour la fiabilité du robot. Donc, perdant les quatre premières manches de la compétition, on a laissé de côté les gains de temps de 10s en partant à 20cm de la ligne (départ en aveugle), en supprimant la fonction raccourci et en travaillant uniquement sur le suivi de piste et l'arrivée.

4 Améliorations Envisagées :

_ Pour les capteurs

Il faut absolument que les capteurs « collent » à la piste indépendamment du chassi du robot. La piste étant placée sur une estrade en bois, elle subissait des déformations suite aux grosses chaleurs du moi de juin. Donc, tous les réglages effectués sur les capteurs aux niveaux des seuils (capteurs analogiques) ont été faussés. Ce qui même aussi à proposer une fonction de calibrage des capteurs avant le départ.

Un positionnement dit en « râtelier » permettrait un balayage plus efficace et plus précis. Avec un nombre de huit capteurs indépendants, gérés par un PIC servant uniquement pour la conversion de tension et le renvoie permanent des informations de ces derniers. Positionner en arc de cercle, ils épouserons parfaitement les virages et seront, d'après moi, d'une efficacité redoutable.

De plus, ne surtout pas omettre l'éclairage du parcours. Pour cela un simple cache couvrant la totalité des capteurs sera valable.

Une caméra pourra être utilisé allégrement. Pour cela, réduire au maximum les temps d'analyse d'images en supprimant une bande sur deux ou plus. Pareillement, il faudra à tout prie, s'affranchir des problèmes de luminosités.

_ Partie électrique

Un élévateur de tension sera le bienvenu. La piste ayant de grandes lignes droites, un temps important sera gagné, malgré tout, attention aux virages. Une accélération progressive et un net ralentissement en fin et en début de virage serra nécessaire.

De plus, pour une modularité optimale, un système d'insertion de carte enfichable serait adéquat. Type PCI ou nappe IDE.

_ Stratégie :

Il faut absolument avoir un robot fiable. Cette faille causa notre défaite lors du concours. S'affranchir du moindre problème, si nécessaire, supprimer des composants, des options. Car, un parcours non bouclé donne un temps de 240 secondes au lieu d'un parcours verrouillé environ en trente secondes. Un seul plantage du robot sur le circuit se paye cher. Majoré de 240s, cela a vite fait de nous donné les dernières places à se disputer.

Pour les essais, il faut tenir compte des différences entre les pistes d'essais et le parcours officiel. Le parcours officiel est plus éclairé, il présente des bosses et des parasites permanents (personnes proches de la piste, bordures, passage proche de l'autre robot).

Pour bluffer les autres robots : la couleur, les ultra sons.
Pour les robots utilisant des caméras, une couleur identique à la moquette ne leur permettront pas de faire le distinguo.
Pour les robots utilisant l'ultra son, des émetteurs renverront un signal simulant la présence du robot.

5 Conclusion :

C'est un projet dans lequel il faut s'investir et être motivé.
Il est intéressant par le fait que chaque équipe se doit de réaliser son propre robot, autant dans la partie programmation que fabrication comme l'implantation des différèrent capteurs ou la décoration. Il y a aussi la découverte de nouveau composant comme le QT110 ou le télémètre qu'il faut réussir à mettre en place et à utiliser. Et la découverte de nouveau langage (java), qu'il a fallu acquérir en un mois.
De plus, le fait d'être autonome, nous laisse une large plage de manœuvre
Enfin il faut se servir de sa tête pour trouver la panne, car bien souvent le robot ne réagit pas comme prévus à cause de la surface sur laquelle il roule, la luminosité...

Enfin, ce projet m'a montré à quel point la répartition des divers travaux est importante, en effet je pense que nous aurions pu approfondir notre travail et améliorer les performances du robot s'il y avait eu réellement répartition des tâches au sein de notre équipe.

Nous tenons à remercier tous les professeurs qui ont pu nous éclairer et surtout M ARLOTTO qui fut un tuteur fort sympathique, ainsi que notre magasinier qui a bien voulu accepter l'absence de bon de commande tout au long de cette année.

6 Description du programme et des fonctions :

Pour plus de facilité de modification, nous avons scindé le programme en plusieurs fichiers, réparti en diverses fonctions.

Les « ** » signifie que la fonction présentée n'a pas été utilisée pendant le concours, tandis que l'ensemble du programme ci après est le programme chargé dans le PIC lors du concours.

Programme Principal:

Il gère les différentes fonctions utiles à la navigation du robot.

```
/*********************************************************************/
/*********************************************************************/
/**                        The ROBOT of GEII-Warrior                **/
/**                             The Master Game                     **/
/**                                                                 **/
/**                             XXXX-XXX-XXXX                       **/
/*********************************************************************/
/*********************************************************************/

                          /*Programme*/

/*********************************************************************/
/* erreur au niveau des capteurs: gauche et droit non identique.     */
/* moteur à 100%                                                     */
/* IN 1&2 à utiliser pour régler les capteurs                        */
/* vitesse=25%, delta_vfort=20, delta_vdoux=15                       */
/* vitesse=50%, delta_vfort=35, delta_vdoux=25                       */
/* vitesse=100%, delta_vfort=80, delta_vdoux=60                      */
/*                                                                   */
/* par convention, tout ceux qui touche au niveau des capteurs, c'est*/
/* par une vue de arriére au robot.                                  */
/* Sens de marche.!!!                                                */
/*     pour arthur: gauche == babord /droit == tribord               */
/* moteur gauche = 0                                                 */
/* moteur droit = 1                                                  */
/*********************************************************************/

#include <pic.h>

  #include "télémétre.h"
  #include "can.h"
  #include "pwm.h"
  #include "direction.h"
  #include "démarrage.h"
  #include "arrivée.h"
  #include "delay.h"
  #define JACK RD0

    void jack(void);
   unsigned char capt1;
   unsigned char i;
```

```
void main(void)
{

   TRISD1=0;    TRISB2=0;TRISB1=0;TRISB5=0;TRISB4=1;

   initpwm();          /*****************************************/
   InitCan();          /*    initialisation des fonctions       */
   init_telem();       /*                                       */
   micro_switch();     /*                                       */
   RECUL=0;            /*****************************************/
   jack();             /*                                       */
   démarrage();        /*****************************************/

   capt1=0;
   while(1)
   {
     for(i=0;i<5;i++)
     {

         capt1 = (capt1 << 1) | seuil(convcapt(i));
                     /* on récupère les valeurs des capteurs de*/
                     /* gauche à droite, sens de course       */
         capt1=0x1f & capt1;

     }
                     /* ici on détermine la position du robot par*/
                     /* rapport a la ligne                    */
    direction (capt1);
                     /* on décide de la direction et on déduit  */
                     /* les valeurs de la pwm                   */


     if(RB4==0)
     {
           arrivée();  /* pour l'arrivée, détection par contact de*/
                       /* type interrupteur (Tout ou Rien)       */
     }
   }
}

/*********************************************************************/
/*                                                                   */
/* cette fonction permet de faire démarrer le robot environ une      */
/* secondes après/*avoir enlevé le jack. Elle force la position sur */
/* stop en attendant la consigne de départ.                          */
/*                                                                   */
/*********************************************************************/
   void jack(void)
   {
        while(JACK==0)
        {
              stop();           /* on attend la consigne de départ   */
              DJAUNE=1;DROUGE=1;DVERT=1;
        }
        for(i=0;i<5;i++)
        {
              DelayMs(50);     /* on attend 5 fois 50ms              */
        }
   }
```

Fonctions :

_Démarrage.c :

Il permet d'effectuer un démarrage dit en aveugle. C'est à dire à 20 cm en avant de la piste.

```
#include <pic.h>
#include <stdlib.h>
#include "pwm.h"
#include "can.h"


void démarrage(void)
{

      unsigned char i,capt=0;       /* départ avant la ligne */

      while(capt==0)          /* tant qu'on ne trouve pas de */
                              /* ligne on avance tout droit  */
      {
            for(i=0;i<5;i++)
            {
                  capt = (capt << 1) | seuil(convcapt(i));
                  capt=0x1f & capt;
            }
            setdutycycle( 1 , 100); /* moteur à fond        */
            setdutycycle( 0 , 100); /* moteur à fond        */
      }
}
```

Démarrage.h :

```
#ifndef _DEMARRAGE_H
#define _DEMARRAGE_H

void démarrage(void);

#endif
```

_Arrivée.c :

C'est cette fonction qui gère la barre à faire tomber à l'arrivée du robot en bout de piste. La fonction recul est utile ici, à fin de stopper net le mobil, tout ceux-ci pour éviter de faire tomber une seconde barre distante de 20 cm par rapport à la première.

```
#include <pic.h>
#include <stdlib.h>
#include "pwm.h"
#include "direction.h"
#include "delay.h"

void arrivée(void)
{
```

```
        char i;
        RECUL=1;
        setdutycycle(1,25);
        setdutycycle(0,25);
        for(i=0;i<3;i++)
        {
              DelayMs(25);
        }
        DROUGE=0;
        DVERT=1;

        while(1)
        {
              RECUL=0;                /* lorsqu'on est arrive, */
              setdutycycle(1,0);      /* le robot stop et fait */
              setdutycycle(0,0);      /* une marche arriére    */
              for(i=0;i<6;i++)
              {
                    DelayMs(25);
              }
              DROUGE=!DROUGE;
              DVERT=!DVERT;
        }
}
```

Arrivée.h :

```
#ifndef _ARRIVEE_H
#define _ARRIVEE_H

void arrivee(void);
void DelayMs(unsigned char);


#endif
```

_Direction.c :

On dira que c'est le cœur du robot. Là, on gère les deux rapports cycliques présents sur les deux moteurs selon les consignes reçues par la fonction Can.

```
/*************************************************************/
/*                                                           */
/*    DIRECTION , douxdroite, fortdroite , douxgauche ,      */
/*    fortgauche , droit , recul , raccourci , stop ,        */
/*    croisement                                             */
/*                                                           */
/*  ces fonctions sont utilisées pour les commandes moteur   */
/*                                                           */
/*                                                           */
/*************************************************************/

#include <pic.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
```

```
#include "pwm.h"
#include "direction.h"
#include "télémetre.h"
#include "can.h"
#include "delay.h"

int          dist, intcroi,    intrac,     inttele,    intrec;
unsigned int      perdu=0,     rac=0 ;
unsigned char     vitesse=100,delta_vfort=82,delta_vdoux=70;

/* le fort est terriblement bien, vue les grosses courbures*/
/* le doux est enclenché pendant  un temps trop court:     */
/* solution :                                              */
/* faire une boucle pour un doux se décrémentant petit à   */
/* petit mais le laisser suffisamment de temps             */

/*   initialisation de la vitesse (max 100%)               */
/*   initialisation du rapport des virages                 */
/*   il faut que le % soit plus faible en fort             */



/*********************************************************/
/* on doit faire varier la valeur à envoyer sur le moteur  */
/* 2 cas considerés comme impossibles 101 , 000
     */
/* on a num=1 quand on est sur la ligne
/* et on a num=0 quand on est sur la moquette (normalement)*/
/* attention les diodes sont actives sur le 0  (négatif)   */
/*********************************************************/

   void direction (unsigned char cap)
   {
      if (cap!=0)
      {
         RECUL=0; perdu=0;
      }
      switch(cap)
      {
         case 0x01: fortgauche();  DJAUNE=1;DROUGE=0;DVERT=0;
            break;
         case 0x02: douxgauche();  DJAUNE=1;DROUGE=1;DVERT=0;
      break;
         case 0x03: fortgauche();  DJAUNE=1;DROUGE=0;DVERT=0;
      break;
         case 0x04: toutdroit();   DJAUNE=0;DROUGE=1;DVERT=0;
      break;
         case 0x06: douxgauche();  DJAUNE=1;DROUGE=1;DVERT=0;
      break;
         case 0x08: douxdroite();  DJAUNE=0;DROUGE=1;DVERT=1;
      break;
         case 0x10: fortdroite();  DJAUNE=0;DROUGE=0;DVERT=1;
      break;
         case 0x0c: douxdroite();  DJAUNE=0;DROUGE=1;DVERT=1;
      break;
         case 0x18: fortdroite();  DJAUNE=0;DROUGE=0;DVERT=1;
      break;
```

```
    case 0x00: recul();          DJAUNE=1;DROUGE=0;DVERT=1;
   break;
      default : stop();          DROUGE=1;DVERT=1;DJAUNE=1;
        break;

   }
}

void douxgauche(void)
{
   setdutycycle( 0 , vitesse-delta_vdoux);
   setdutycycle( 1 , vitesse);
}


void fortgauche(void)
{
   setdutycycle( 0 , (vitesse-delta_vfort));
   setdutycycle( 1 , vitesse);
}

void douxdroite(void)
{
   setdutycycle( 1 , (vitesse-delta_vdoux));
   setdutycycle( 0 , vitesse);
}

void fortdroite(void)
{
   setdutycycle( 1 , (vitesse-delta_vfort));
   setdutycycle( 0 , vitesse);
}

void toutdroit(void)
{
   setdutycycle( 1 , vitesse);
   setdutycycle( 0 , vitesse);
}

void recul (void)
{
   if(intrec==0)
   {
               perdu++;
   if(perdu>1000)
   {
   RECUL=1;
   setdutycycle(1 ,50 );
   setdutycycle(0 ,50 );
   }
   if(perdu>3000)
   {
   perdu=0;
               RECUL=0;
               }
   }
}

void stop(void)
{
   setdutycycle( 1 , 0);
```

```
    setdutycycle( 0 , 0);
}

void ralenti(void)
{
   setdutycycle( 1 , 75);
   setdutycycle( 0 , 75);
}
```

Direction.h :

```
#ifndef _REDIRECT_H
#define _REDIRECT_H
#define RECUL RD1              /* pour la marche arrière     */
#define DROUGE RB2
#define DVERT RB1
#define DJAUNE RB5

void douxdroite(void);
void fortdroite(void);
void douxgauche(void);
void fortgauche(void);
void toutdroit(void);
void recul (void);
void stop(void);
void ralenti(void);

void direction (unsigned char cap);


#endif
```

_PWM.c :

Ici, on s'occupe de générer les deux rapports cycliques variables et indépendants qui permettent d'alimenter les moteurs à travers l'amplificateur de puissance.
Converti en pourcentage, cela facilite les réglages.

```
/*************************************************************/
/*                                                           */
/*                      PWM ET INIT_PWM                      */
/*                                                           */
/*ces fonctions sont utilisées pour les commandes moteur     */
/*                                                           */
/*                                                           */
/*************************************************************/

#include <pic.h>
#include <stdlib.h>

/*************************************************************/
/* initialisation CCP1 et CCP2 en mode PWM avec une          */
/* frequence de 15khz et un quartz de 20 MHz,on a calculer   */
/* PR2= 100 % du rapport cyclique du signal de 15khz         */
/*************************************************************/
```

```
   void initpwm(void)
   {
      PR2= 82;
      CCPR1L=0;
      CCP1CON=0x0c;     /* on met a 0 les bits 4 et 5 pour */
                        /* se positionner en mode PWM      */
      TRISC2=0;         /*                mise en sortie   */
      TRISC1=0;         /*                mise en sortie   */
      CCPR2L=0;
      CCP2CON=0x0c;
      T2CON=0x05;       /*    les bits 0 et 1 de t2con sont */
                        /*    le prescaleur                 */
   }


/*********************************************************/
/*                                                       */
/*                      setdutycycle                     */
/*                                                       */
/* cette fonction permet de mettre un signal ayant un    */
/* rapport cyclique choisi sur le moteur choisi.         */
/* le numéro du moteur et le rapport cyclique sont entres */
/* en paramètre.                                         */
/* si le numéro du moteur ou le rapport présente une erreur*/
/* alors les sorties ccp1 et ccp2 sont mises a 0.        */
/*********************************************************/

   void setdutycycle(char num , unsigned char dutycycle )
   {
      char val1=PR2;
      /*RECUL=0;*/  /*pour avancer*/
      unsigned int dut;
      if ((dutycycle > 100))
      {
         CCPR1L =0;
         CCPR2L =0;
         return;
      }
      dut= ( (unsigned int) val1 * (unsigned int) dutycycle ) /
100;             /* on transpose en % */
      dut = dutycycle;
      if ( num==0)                              /*moteur 1*/
      {
         CCPR1L=dutycycle;
         return;
      }
      if ( num==1)                              /*moteur 2*/
      {
         CCPR2L=dutycycle;
         return;
      }
      CCPR1L =0;
      CCPR2L =0;
      return;
   }
```

PWM.h :

```
#ifndef _PWM_H
```

```
#define _PWM_H

void initpwm(void);
void setdutycycle(char  , unsigned char  );

#endif
```

_Can.c :

Le second point névralgique du robot : la lecture de piste. Grâce au balayage des capteurs par le programme principal, on s'occupe, ici, d'affecter telle ou telle entrée analogique afin de la convertir en valeur numérique codé sur 8 bits. Il est possible d'utiliser deux autres bits supplémentaires, mais la précision est suffisamment importante.
De plus, une fois cette valeur convertie, un seuil, calibré au préalable, type à hystérésis, nous permet d'affirmer ou non la présence de la piste.

```
/*************************************************************/
/*                                                           */
/*                      CONVCAPT , INITCAN ET waitRAT        */
/*                                                           */
/*          ces fonctions sont utilisées pour le CAN         */
/*                                                           */
/*                                                           */
/*************************************************************/

#include <pic.h>
#include <stdlib.h>

void waitRAT(void);
unsigned char counter;


void InitCan(void)
      {
        ADCON1 = 0x00 ;             /* Port A&E analogique ,*/
                                    /* justification gauche */
        ADCON0 = 0x81 ; /* Tad = 32*Tosc = 32 * 50ns =      */
                        /* 1.6µs >= 1.6µs(cf p116), ADON ;  */
        ADIF = 0 ;            /* interdit les interruptions */
         }

unsigned char convcapt(unsigned char ch)
      {
            ADCON0 = ADCON0 & 0xC7 ;
                              /* RAZ CHSx Sélection du canal */

      if(ch==4)
      {
            CHS2=1;
      }
      if((ch==3) || (ch==2))
      {
            CHS1=1;
      }
      if((ch==1) || (ch==3))
      {
```

```
            CHS0=1;
      }


       waitRAT();          /* Attendre le temps d'acquisition */

        ADGO = 1 ;       /* Démarre la conversion              */
                         /* Attend la fin de conversion        */
        while (ADGO && ADON) ;
                         /* Par sécurité on teste aussi ADON   */
            /* car sinon on rentre dans une boucle infinie */
        return ADRESH;
      }


/*************************************************************/
/* Required acquisition time for A/D module form Max.        */
/* Heise 2002 p. 115 equation 11-1 Tacq=19,72,e-6 seconds,   */
/* approx. 2,e-5 seconds. One instruction is executed in     */
/* one cycle, one cycle at 20MHz takes 200ns. Thus we must   */
/* idle for 100 instructions. Formula for this loop          */
/* Inst(I)=1+1+( (I-1)*(1+2) )+2+2=6+3I-3=3I+3+2             */
/* Inst.Cycles for the call of this function =>Inst(I)=3I+5*/
/* Inst(I) must be 400 100=3I+5 => I=29                      */
/* Timing critical, so this is still in assembler.           */
/*************************************************************/

void waitRAT(void)
{

      #asm
      I EQU 54;   /*    nous avons doublé la valeur de I    */
                  /*    pour plus de sécurité               */
      movlw  I;         /*    1 top d'horloge machine       */
      movwf  _counter;  /*    1 top d'horloge machine       */
      WaitRAT_Loop
      decfsz _counter,f  ;    /* 1 top d horloge machine     */
                              /* si I!=0, autrement deux     */
                              /* tops d horloge              */
      goto   WaitRAT_Loop ;   /* 2 tops d horloge            */
      #endasm
}

/*************************************************************/
/*                                                           */
/* cette fonction permet de déterminer si le robot se        */
/* trouve sur la piste, en fonction de la variable des       */
/* capteurs.                                                 */
/*                                                           */
/*************************************************************/

unsigned char seuil(unsigned char v )
   {
      int num;
      if((0x01 <= v)  && (v < 0xA0) )
      {
         num=1;
      }
      else
      {
         if((0x90 <=v) && (v< 0xFE))
```

```
            {
               num=0;
            }
            else num = 0x0f;
         }
         return num;
      }
```

Can.h :

```
#ifndef _CAN_H
#define _CAN_H

void InitCan(void);
unsigned char convcapt(unsigned char ch);
unsigned char seuil(unsigned char v );

#endif
```

_I2C.c :

C'est la fonction qui s'occupe de la liaison I2C avec le télémètre.

```
#include    <pic.h>
#include    "delay.h"
#include    "i2c.h"

/*************************************************************/
/*    I2C functions for HI-TECH PIC C - master mode only     */
/*                                                           */
/* TIMING -see Philips Sem. 80C51-based 8-bit ucontrollers,*/
/*             or e.g. Philips PCF8574 data sheet            */
/*    Send stop condition - data low-high while clock high   */
/*************************************************************/

i2c_Stop(void)
   {              /* don't assume SCL is high on entry      */
      SCL_HIGH();
      SDA_LOW();               /* ensure data is low first */
      DelayUs(I2C_TM_DATA_SU);
      SCL_DIR = I2C_INPUT;          /* float clock high     */
      DelayUs(I2C_TM_STOP_SU);
      SDA_HIGH();         /* the low->high data transistion */
      DelayUs(I2C_TM_BUS_FREE);     /* bus free time before */
                                    /* next start           */
      SDA_DIR = I2C_INPUT;          /* float data high      */
      return;
   }

/*************************************************************/
/*    Send (re)start condition - ensure data is high then    */
/* issue a start condition - see also i2c_Start() macro      */
/*************************************************************/

i2c_Restart(void)
   {
      SCL_LOW();                     /* ensure clock is low */
      SDA_HIGH();                    /* ensure data is high */
```

```
      DelayUs(I2C_TM_DATA_SU);
      SCL_DIR = I2C_INPUT;          /* clock pulse high */
      DelayUs(I2C_TM_SCL_HIGH);
      SDA_LOW();               /* the high->low transition */
      DelayUs(I2C_TM_START_HD);
      return;
   }

/*********************************************************/
/*    Send a byte to the slave - returns true on error     */
/*********************************************************/

unsigned char
i2c_SendByte(unsigned char byte)
   {
      signed char i;

      for(i=7; i>=0; i--)
      {
         SCL_LOW();                    /* drive clock low   */
                /* data hold time = 0, send data now     */
         SDA_DIR = ((byte>>i)&0x01);
         if ((byte>>i)&0x01)            /* bit to send     */
         {
            SDA_HIGH();
         }
         else
         {
            SDA_LOW();
         }
         DelayUs(I2C_TM_DATA_SU);
         SCL_DIR = I2C_INPUT;      /* float clock high     */

         if(i2c_WaitForSCL())    /* wait for clock release  */
            return TRUE;                 /* bus error      */

         DelayUs(I2C_TM_SCL_HIGH); /* clock high time      */
      }

      return FALSE;
   }

/*********************************************************/
/*    send an address and data direction to the slave      */
/*      - 7-bit address (lsb ignored)                      */
/*      - direction (FALSE = write )                       */
/*********************************************************/

unsigned char
i2c_SendAddress(unsigned char address, unsigned char rw)
   {
      return i2c_SendByte(address | (rw?1:0));
   }

/*********************************************************/
/*    Check for an acknowledge                           */
/*      - returns ack or ~ack, or ERROR if a bus error     */
/*********************************************************/

signed char
i2c_ReadAcknowledge(void)
```

```
   {
      unsigned char ack;

      SCL_LOW();                        /* make clock is low */
      SDA_DIR = I2C_INPUT;          /* disable data line   */
                                    /* -listen for ack     */
      DelayUs(I2C_TM_SCL_TO_DATA);       /* SCL low to data*/
                                         /* out valid      */
      SCL_DIR = I2C_INPUT;          /* float clock high    */
      DelayUs(I2C_TM_DATA_SU);
      ack = SDA;                    /* read the acknowledge */
                     /* wait for slave to release clock  */
                     /* line after processing byte       */
      if(i2c_WaitForSCL())
      return I2C_ERROR;
      return ack;
   }

/*********************************************************/
/*    Read a byte from the slave                         */
/*      - returns the byte, or I2C_ERROR if a bus error  */
/*********************************************************/

int
i2c_ReadByte(void)
   {
      unsigned char i;
      unsigned char byte = 0;

      for(i=0; i<8; i++)
      {
         SCL_LOW();                        /* drive clock low */
         DelayUs(I2C_TM_SCL_LOW);  /* min clock low  period */
         SDA_DIR = I2C_INPUT;      /* release data line */

         SCL_DIR = I2C_INPUT;      /* float clock high */
         if(i2c_WaitForSCL())
            return I2C_ERROR;
         DelayUs(I2C_TM_SCL_HIGH);
         byte = byte << 1;         /* read the next bit */
         byte |= SDA;
      }
      return (int)byte;
   }

/*********************************************************/
/*    Send an (~)acknowledge to the slave
/*      - status of I2C_LAST implies this is the last byte */
/*    to be sent                                         */
/*********************************************************/

void
i2c_SendAcknowledge(unsigned char status)
   {
      SCL_LOW();
      if ( status & 0x01) {
         SDA_LOW();       /* drive line low -> more to come */
      }
      else
      {
         SDA_HIGH();
```

```
      }
      DelayUs(I2C_TM_DATA_SU);
      SCL_DIR = I2C_INPUT;              /* float clock high */
      DelayUs(I2C_TM_SCL_HIGH);
      return;
   }

/*************************************************************/
/* Send a byte to the slave and acknowledges the transfer  */
/*      - returns I2C_ERROR, ack or ~ack                     */
/*************************************************************/

signed char
i2c_PutByte(unsigned char data)
   {
      if(i2c_SendByte(data))
         return I2C_ERROR;
      return i2c_ReadAcknowledge();/* returns ack, ~ack */
   }

/*************************************************************/
/* Get a byte from the slave and acknowledges the transfer */
/*      - returns true on I2C_ERROR or byte                  */
/*************************************************************/

int
i2c_GetByte(unsigned char more)
   {
      int byte;

      if((byte = i2c_ReadByte()) == I2C_ERROR)
         return I2C_ERROR;

      i2c_SendAcknowledge(more);

      return byte;
   }

/*************************************************************/
/* Send an array of bytes to the slave and acknowledges the*/
/* transfer                                                  */
/*  - returns number of bytes not successfully transmitted */
/*************************************************************/

int
i2c_PutString(const unsigned char *str, unsigned char length)
   {
      signed char error;

      while(length)
      {
         if((error = i2c_PutByte(*str)) == I2C_ERROR)
            return -(int)length;         /* bus error */
         else
            if(error)
               return (int)length; /* non acknowledge */
         str++;
         length--;
      }
```

```
      return FALSE;                              /* everything OK
*/
   }

/*
 *    Reads number bytes from the slave, stores them at str and
acknowledges the transfer
 *      - returns number of bytes not successfully read in
 */

   unsigned char
   i2c_GetString(unsigned char *str, unsigned char number)
   {
      int byte;

      while(number)
      {
         if((byte = i2c_GetByte(number-1)) == I2C_ERROR)
            return number;               /* bus error */
         else
            *str = (unsigned char)byte;
         str++;
         number--;
      }

      return FALSE;                      /* everything OK */
   }

/*************************************************************/
/*    Opens communication with a device at address. Mode     */
/*    indicates I2C_READ or I2C_WRITE.                       */
/*      - returns TRUE if address is not acknowledged        */
/*************************************************************/

   unsigned char
   i2c_Open(unsigned char address, unsigned char mode)
   {
      i2c_Start();
      i2c_SendAddress(address, mode);
      if(i2c_ReadAcknowledge())
      return TRUE;

      return FALSE;
   }

/*************************************************************/
/*    wait for the clock line to be released by slow slaves*/
/*      - returns TRUE if SCL was not released after the     */
/*        time out period.                                   */
/*      - returns FALSE if and when SCL released             */
/*************************************************************/

unsigned char
i2c_WaitForSCL(void)
   {                    /* SCL_DIR should be input here */
      if(!SCL)
      {
         DelayUs(I2C_TM_SCL_TMO);
              /* if the clock is still low -> bus error */
         if(!SCL)
            return TRUE;
```

```
      }
      return FALSE;
   }

unsigned char i2c_read(unsigned char ucAdr)
   {
      unsigned char ucDat;

      if (i2c_ReadFrom(ucAdr)==0)
      {
         ucDat=i2c_GetByte(I2C_MORE);
         i2c_Stop();

      }

      return(ucDat);
   }
```

I2C.h :

```
#ifndef     _I2C_H_
#define _I2C_H_

/*************************************************************/
/*    SDA (data) and SCL (clock) bits                        */
/*                                                           */
/*                     Special note!!!                       */
/*                                                           */
/*    If the clock and data lines are in the same port, you*/
/*    will need to beware of the Read/Modify/Write issue in*/
/*    the PIC - since a bit set or clear on any one bit in */
/*    a port will read and write back all other bits, any  */
/*    bits configured as input which                         */
/*************************************************************/


/* Uncomment the next line to use the PIC's SSP Module*/
#define I2C_MODULE 1

#ifdef I2C_MODULE            /* I2C module uses PORT C */
#define SCL             RC3   /* clock on port C bit 2 */
#define SCL_DIR         TRISC3
#define SDA     RC4            /* data on port C bit 1 */
#define SDA_DIR         TRISC4
#define I2CTRIS TRISC
#define MASTER_MODE     0B1011 /* I2C firmware controlled */
                               /* Master Mode (slave idle) */
#define SSPMode(val)   SSPCON &=0xF0; SSPCON|=(val & 0xf)

#else       /* Change port as required - defaults to port b */
#define     SCL         RB2         /* clock on port B bit 2 */
#define     SCL_DIR           TRISB2

#define SDA             RB1         /* data on port B bit 1 */
#define     SDA_DIR           TRISB1
#define I2CTRIS TRISB

#endif
```

```
#define M_SDA_INP 0x02
#define M_SDA_OUT   0xFD
#define M_SCL_INP   0x04
#define M_SCL_OUT 0xFB

#define I2C_INPUT 1           /* data direction input */
#define I2C_OUTPUT     0      /* data direction output */

#define I2C_READ  0x01        /* read bit used with address */
#define I2C_WRITE 0x00        /* write bit used with address */

#define FALSE          0
#define TRUE           !FALSE

#define I2C_ERROR (-1)

#define I2C_LAST  FALSE /* SendAck: no more bytes to send */
#define I2C_MORE  TRUE        /* SendAck: more bytes to send */

#define i2c_Start()           i2c_Restart()
#define i2c_WriteTo(address)  i2c_Open((address), I2C_WRITE)
#define i2c_ReadFrom(address) i2c_Open((address), I2C_READ)

#ifdef I2C_MODULE
#define SCL_HIGH() SCL_DIR = I2C_INPUT
#define SCL_LOW()  SCL_DIR = I2C_OUTPUT
#define SDA_HIGH() SDA_DIR = I2C_INPUT
#define SDA_LOW()  SDA_DIR = I2C_OUTPUT
#else
#define SCL_HIGH() SCL = 1; SCL_DIR = I2C_OUTPUT
#define SCL_LOW()  SCL = 0; SCL_DIR = I2C_OUTPUT
#define SDA_HIGH() SDA = 1; SDA_DIR = I2C_OUTPUT
#define SDA_LOW()  SDA = 0; SDA_DIR = I2C_OUTPUT
#endif

/*********************************************************/
/* Timings for the i2c bus. Times are rounded up to the    */
/* nearest micro second.                                 */
/*********************************************************/

#define I2C_TM_BUS_FREE      5
#define    I2C_TM_START_SU        5
#define I2C_TM_START_HD      4
#define I2C_TM_SCL_LOW       5
#define    I2C_TM_SCL_HIGH        4
#define I2C_TM_DATA_SU       1
#define I2C_TM_DATA_HD          0
#define I2C_TM_SCL_TO_DATA   4     /* SCL low to data valid */
#define    I2C_TM_STOP_SU         4
#define I2C_TM_SCL_TMO       10    /* clock time out */

extern signed char     i2c_ReadAcknowledge(void);
extern unsigned char   i2c_SendAddress(unsigned char, unsigned
char);
extern unsigned char   i2c_SendByte(unsigned char);
extern int       i2c_ReadByte(void);
extern void      i2c_Restart(void);
extern void      i2c_Stop(void);
extern void      i2c_SendAcknowledge(unsigned char);
extern signed char     i2c_PutByte(unsigned char);
extern int       i2c_GetByte(unsigned char);
```

```
extern unsigned char    i2c_Open(unsigned char, unsigned char);
extern unsigned char    i2c_GetString(unsigned char *, unsigned
char);
extern int        i2c_PutString(const unsigned char *, unsigned
char);
extern unsigned char    i2c_WaitForSCL(void);
extern void             i2c_Free(void);
#endif                  /* _I2C_H_ */
```

_Telemétre.c :

Cette fonction est issue du mode croisement, inexploitée durant le concours. Elle permet de faire un « appel » au télémètre à infra son pour savoir si oui ou non un robot est présent sur notre flan droit.

```
/*********************************************************/
/* Programme de test lcd+i2c+sonar sur 16F877            */
/* A compiler avec lcd.c  delay.c i2c.c                  */
/* Ce programme affiche la distance mesurée par le sonar*/
/* la température lue par le DS1621 et la luminosité     */
/* ambiante vue par le module sonar sur l'écran lcd      */
/* le lcd est en mode 4 bits(seuls D4-D7 sont utilisees)*/
/* Voir le câblage dans lcd.c                            */
/* Auteur : Arlotto     Date : 03/05/02   création       */
/*       Arlotto         12/02/03                        */
/*********************************************************/

#include <pic.h>
#include <stdlib.h>

#include "delay.h"
#include "i2c.h"

   void WriteSonar(unsigned char location, unsigned char byte);
   unsigned char ReadSonar(unsigned char location);

#define DS1621   0x48<<1 /* adresse i2c du ds1621 */
#define SONAR    0xE0    /* adresse i2c du module sonar */

void init_telem(void)
      {
          TRISC3=0;
 /*************** initialize i2c module ****************/
      #ifdef I2C_MODULE
        SSPMode(MASTER_MODE);
        SSPEN = 1;
        CKP = 1;
      #else
                #error "I2C NON INITIALISE"
      #endif

      }

   int telem(void)
   {

      unsigned char light,distH,distL,Cpt;
      int dist;
```

```
      /* Sonar */
/*Lecture luminosité pour s'assurer que le sonar répond */
/* (pour l'instant on recommence tant que ça ne marche pas!) */

      WriteSonar(0,0x51);
      DelayMs(25);
      Cpt=0;
      do {
         light=ReadSonar(0);
         Cpt++;
       }while(light==255 && Cpt<255);



      /* Lecture Distance en cm */

      DelayMs(25);
      distH=ReadSonar(2);
      DelayMs(25);
      distL=ReadSonar(3);
      dist = (distH<<8) + distL ;

      DelayMs(25);

          return dist;
   }

/* ecrit un octet sur le sonar */
   void WriteSonar(unsigned char location, unsigned char byte)
   {
      i2c_WriteTo(SONAR);
      i2c_PutByte(location);
      i2c_PutByte(byte);
   }

/* lit un octet sur le sonar */
   unsigned char ReadSonar(unsigned char location)
   {
      i2c_WriteTo(SONAR);
      i2c_PutByte(location);
      i2c_ReadFrom(SONAR);
      return i2c_GetByte(I2C_LAST);
   }
```

Telemétre.h :

```
#ifndef     _telem_H_
#define _telem_H_

void init_telem(void);
int telem(void);

#endif
```

_Raccourci.c :

```
void raccourci (void)
{
            for(i=0 ;i<10 ;i++)
            {
                  DelayMs(50);
            }
            while(( 0x20 != ( capt && 0x20 )))
            {
                  recul();
                  RECUL=0;
            }
            fortgauche(); RB2=0;RB1=1;
            DelayMs(25) ;
}
```