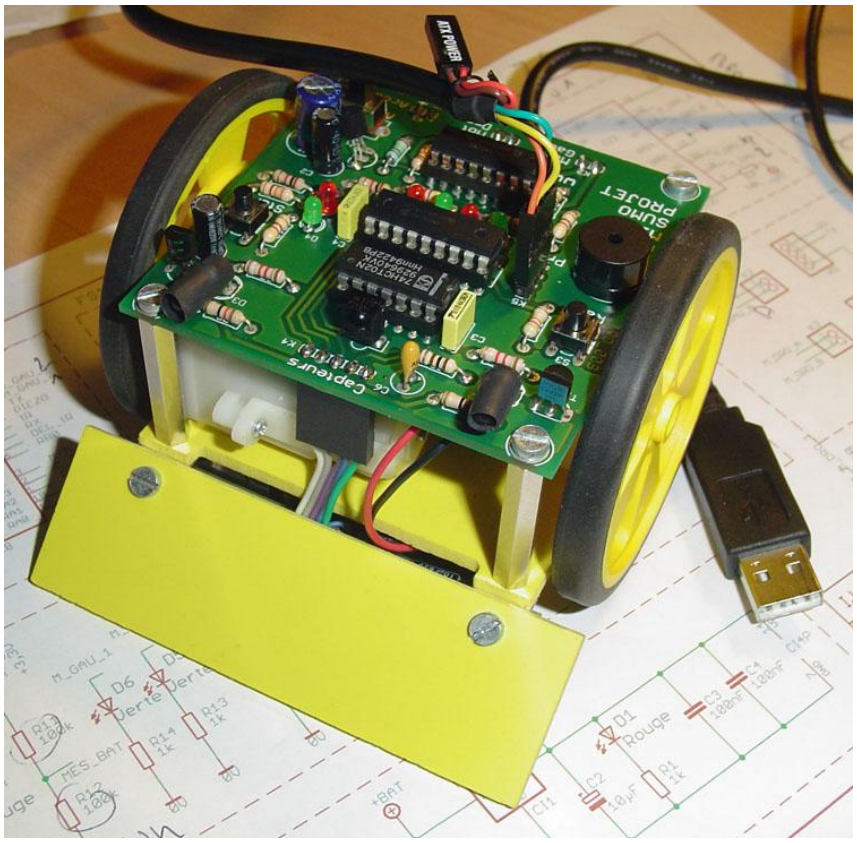
Exercices
de robotique mobile



Semestre 3

2011/2012

Frédéric GIAMARCHI

Département G.E.I.I.

I.U.T. de Nîmes – Université Montpellier II

Sommaire

PRÉSENTATION

Cette série de travaux pratiques doit vous permettre de découvrir les notions de systèmes embarqués, de programmation en C de systèmes temps réels et les premières subtilités de la robotique mobile.

La programmation d'un robot mobile diffère de celle d'un programme pour PC sur divers points.

Le matériel est un ensemble d'éléments mécaniques et électriques dont il est important de connaître les caractéristiques pour bien les utiliser en relation avec la carte électronique.

La carte électronique regroupe toutes les interfaces, les plus adaptées aux contraintes du matériel. Mais aussi le cœur du système, à savoir un microcontrôleur capable de piloter toutes les ressources du matériel proposé.

Le langage de programmation le plus adaptée est le C. Mais il peut être nécessaire d'écrire quelques routines en assembleur, pour réduire la taille d'une fonction critique.

Un environnement de développement est aussi nécessaire. Celui-ci devant être adapté à la programmation des microcontrôleurs utilisés.

Le mode de programmation et le matériel pour programmer le composant sont des éléments qu'il est nécessaire de connaître et de comprendre aussi pour pouvoir se concentrer sur la partie apprentissage.

LE ROBOT MINI SUMO

La partie mécanique du robot est un kit disponible auprès de la société ***Robitec*** [1]. Cette jeune société s'est spécialisée dans la conception de maquette pédagogique de type robotique. Le modèle que nous avons choisi dispose d'une carte détection de ligne ou détection de bordure. Cela permet de s'essayer aux techniques de suivi de ligne en supplément de la version sumo.

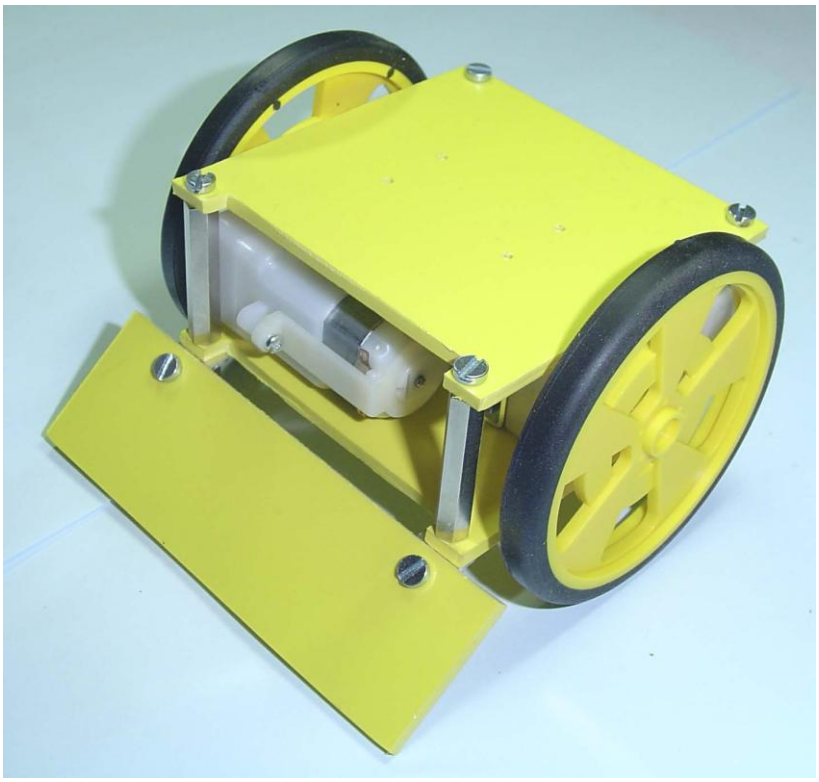


Photo 1 : La mécanique est une base roulante de chez Robitec

La base roulante est équipée de deux motoréducteurs 6Volts. Les roues utilisées ont un fort pouvoir d'adhérence. Elles sont fixées directement sur l'arbre de sortie des motoréducteurs. La vitesse de déplacement du robot n'est pas très élevée du fait de la grande démultiplication 1/143. Mais ce n'est pas ce que l'on attend d'un robot sumo.

La pelle placée à l'avant sert aussi de troisième point d'appui pour l'équilibre de l'ensemble. Les cartes de détection du sol sont fixées sous cette pièce.

Les piles ou accumulateurs sont placés sous le robot dans un bloc support piles.

CARACTÉRISTIQUES :

La tension d'alimentation ne doit pas dépasser 6Vols. Donc quatre accumulateurs en série délivrent une tension de 4,8Volts tout à fait adaptée au montage.

LA CARTE ELECTRONIQUE

Cette carte a été réalisée spécialement pour la base robotique. Il s'agit d'une version adaptée de la carte Picky 1 développée à l'IUT en 2004. Le contrôle de la motorisation a été simplifié sans nuire à l'efficacité. Les cartes de détection de la bordure ont été développées pour le châssis avec l'aide des étudiants ainsi que la partie détection de l'adversaire.

Le schéma détaillé est fourni en fin de document. Il vous permettra de repérer les liaisons entre les diverses interfaces et le microcontrôleur.

LE MICROCONTRÔLEUR

Le microcontrôleur choisi, un PIC16F88, est un modèle très performant dont les caractéristiques sont indiquées dans le tableau suivant, avec le PIC16F84, son prédécesseur pour comparaison.

Composant	Mémoire Programme		RAM	EEPROM	E/S	CAN	MLI	MSSP		UART	Timer 8/16bit	Oscillateur
	Octets	Instructions						SPI	I2C			
PIC16F84	1k	1024	68	64	13	0	0	Non	Non	Non	1	10MHz
PIC16F88	7168	4096	368	256	16	7	1	Oui	Esclave	Oui	2/1	20MHz

LES CONNECTEURS DE PROGRAMMATION

Le microcontrôleur ne peut pas être programmé par une interface spécialisée appelé ICSP (In Circuit Serial Programming). En cas de perte du programme implanté appelé bootloader, il est nécessaire de sortir le coposant de son support et de le placer sur un programmateur dédié pour reprogrammer le programme initial.

La programmation du composant se fait en indirect par l'interface série RS232.

L'INTERFACE ÉLÉMENTAIRE

On nomme, ainsi, le bouton poussoir, les Dels et le bruiteur de type piezo.

L'INTERFACE DE COMMANDE DES MOTEURS

On va utiliser un circuit spécialisé pour la commande de moteurs à courant continu. Il s'agit d'un double pont en H à transistors bipolaires, le L293D ou le SN754410N.

LES INTERFACES SÉRIE

La liaison RS232 entre la carte et un PC un moyen simple pour dialoguer ou pour mettre au point les programmes. Elle peut être filaire ou HF, à l'aide de module de type XBee par exemple. C'est un outil de développement très utilisé qui permet de visualiser des résultats de mesure et de modifie des paramètres internes.

Elle est aussi utilisée, ici, pour programmer le composant.

La liaison I2C est un autre moyen simple de dialoguer avec des cartes spécialisées (caméra, ultra-sons, odomètre, capteurs de ligne).

LA TÉLÉCOMMANDE IR

Le photo-module infrarouge permet de piloter le robot par une télécommande de type TV, c'est très pratique pour modifier les paramètres du robot pendant qu'il se déplace, lorsque la liaison RS232 n'est plus utilisable.

L'ALIMENTATION

La source de tension est réalisée à partir de 4 accumulateurs de 1,2Volt en série.

Cette tension de 4,8Volts est régulé à 3,3Volts. Le régulateur est un module à faible chute de tension LDO (Low Drop Output voltage) de type MCP1702-33. La tension non régulée en entrée du régulateur ne doit pas descendre sous 4 Volts.

DIVERS

Il a été prévu de pouvoir suivre la tension à l'entrée de la carte. Un pont diviseur de tension réalisé avec deux résistances de précision de même valeur permet de diviser par deux la tension d'entrée. Les entrées du composant n'appréciant pas les tensions supérieures à 3,3Volts.

L'ENVIRONNEMENT DE DÉVELOPPEMENT

L'environnement de développement ***MPLAB*** de ***Microchip*** est un formidable outil d'édition de programmes, de simulation, de débogage et de programmation de composants [3]. Il est gratuit et peut être associé à des compilateurs C d'autres sources que ***Microchip***, comme celui que nous utilisons ***CCS*** [4].

Il est compatible avec des programmateurs classiques ou de type ICSP, comme l'***ICD2*** ou le ***PicKit2*** de ***Microchip*** et les dernières versions ***ICD3*** et ***PicKit3.***

MPLAB

L'environnement de développement ***MPLAB*** de ***Microchip*** est un logiciel d'édition de fichiers texte au format *.asm ou *.c, mais paramétré pour des fichiers de programmation pour les composant ***PIC*** de ***Microchip***. En plus de l'édition de fichiers, il réalise l'assemblage de fichiers assembleur *.asm, ou de fichiers *.c si un compilateur C est installé.

Il crée divers fichiers utiles dont le fichier *.hex permettant de programmer un composant ***PIC***.

COMPILATEUR C

Nous utilisons le compilateur C de ***CCS*** qui est compatible pour tous les composants de ***Microchip*** depuis plusieurs années.

Le compilateur ***CCS*** doit être installé sur le PC, puis un plug-in doit être lancé pour qu'il soit reconnu par ***MPLAB***.

Le gros avantage de ce compilateur est d'être efficace à la compilation, de disposer de beaucoup d'exemples de programmes et d'être suffisamment ancien pour avoir un des forums les plus actifs.

PROGRAMMER UN PIC

Il y a deux moyens de programmer un composant PIC. Soit avec un programmateur classique comme ***ICD2*** de ***Microchip*** qui est reconnu dans l'environnement ***MPLAB***, soit en utilisant une des interfaces de dialogue comme l'USB ou le port série RS232.

C'est cette dernière méthode qui a été choisie ici en raison de sa simplicité d'utilisation qui ne nécessite pas le module ***ICD2***.

Pour cela, il est nécessaire d'installer un programme permettant le dialogue avec un PC. Celui-ci devra être programmé une seule fois avec un programmateur classique comme l'***ICD2*** ou un autre programmateur artisanal.

Ce fichier est un programme d'amorce appelé ***bootloader***.

Dans le cas présent, c'est une liaison série qui sera utilisé entre le robot et un PC. Il est nécessaire de disposer d'une carte additionnelle de conversion TTL/RS232 et d'un câble non croisé.

La disparition des interfaces séries RS232 sur les PC actuelles va entrainer une évolution vers l'utilisation de PIC intégrant une interface USB. Mais le principe ne changera pas.

LE BOOTLOADER

Le ***bootloader*** est un programme qui démarre à la mise sous tension du composant et scrute la liaison de dialogue sélectionnée : RS232, I2C ou encore USB.

Si un PC essaie de dialoguer avec le composant, alors le ***bootloader*** transfert le nouveau programme dans le PIC sans écraser son propre programme.

Puis à la fin du transfert, le ***bootloader*** lance le nouveau programme.

Si au démarrage, il n'y a pas de dialogue avec un PC, alors le ***bootloader*** saute directement au programme précédemment installé.

Cette méthode est plus souple et plus rapide, mais ne permet pas un débogage en profondeur du PIC.

Mais elle nécessite un programme préalablement installé sur le PC pour dialoguer avec le ***booloader*** dans le composant.

TINY BOOTLOADER

Ce programme gratuit pour PC permet de dialoguer avec un ***bootloader*** dans un composant de type PIC [5].

Le fichier à transférer est sélectionner par Browse et apparait dans la ligne en haut.

La vitesse est de 19200 baud.

Le port série doit être choisi en fonction de la situation. Si vous avez un port série, alors cela sera certainement le COM1. Mais si vous n'avez plus de port série, il faut en créer un avec un convertisseur USB/RS232. Ce qui va générer un port COM virtuel, qui apparaitra dans la liste des COM. Si ce n'est pas le cas, mettre à jour les ports COM dans l'onglet Options en cliquant sur ***Search for COMs.***

Cliquer sur ***Write Flash*** pour transférer votre programme dans le composant, puis rapidement, mettre sous tension la carte Picky.

Vérifier les indications de bon transfert ou d'erreur.

Ce programme possède un onglet terminal permettant de dialoguer avec le composant si nécessaire (voir chapitre sur le dialoguer RS232).

MODULE XBEE

Ce module XBee permet de supprimer la liaison filaire entre deux systèmes, sans nuire au fonctionnement. Suivant le module utilisé, la portée va de 10m à 1500m. [6]

Chaque module est paramétré par un programme spécifique pour dialoguer avec un autre module. Dans notre application, les modules sont couplés et plusieurs couples de module peuvent fonctionner ensemble sans risque de perte de dialogue.

L'insertion de la carte ***DROIDS*** avec le ***XBee*** sur un PC par un câble USB génère un port COM virtuel dont il faut connaître le numéro.

PREMIER PROGRAMME

OBJECTIFS

Apprendre à utiliser l'environnement ***Mplab*** de ***Microchip***.

Apprendre à programmer un composant par bootloader

DOCUMENTS

Annexe des fonctions C, schéma du robot en annexe

EXERCICE 1

- Créer un répertoire de travail pour vos exercices sur le stage de robotique, nommé ce répertoire par exemple Robotic.
- Lancer Mplab et créer un nouveau fichier Exo_01.c dans votre répertoire.
- Copier le programme suivant dans le fichier Exo_01.c et créer un projet Exo_01 dans votre répertoire avec le fichier Exo_01.c.

```
#include <16F88.h>
#fuses intrc_io,nowdt,noprotect,nolvp,ccpb3,brownout,put
#use delay (clock=8000000)
void    main(void)
{
        while (1)
        {
                output_high (Pin_A6);
                output_high (Pin_A7);
                delay_ms (500);
                output_low (Pin_A6);
                output_low (Pin_A7);
                delay_ms (500);
        }
}
```

REMARQUES :

Les 3 premières lignes du code source définissent la configuration du µC

While (1)	permet de réaliser une boucle sans fin.
output_high (Pin_A6)	met à 1 la ligne 6 du port A.
delay_ms (500)	marque une temporisation de 500ms.
output_low (Pin_A6)	met à 0 la ligne 6 du port A.
output_toggle (Pin_A6)	bascule l'état de la ligne.

TESTER LE PROGRAMME SUR LE ROBOT

- ✓ Compiler votre programme sous ***MPLAB***. Un fichier exo_01.hex a été créé.
- ✓ Le robot doit être éteint et le câble RS232 ou USB branché entre le PC et le robot. Lancer ***TinyBootloader*** sur le PC et sélectionner le fichier exo_01.hex, ensuite cliquer sur ***Write Flash***, puis allumer le robot.
- ✓ Vous devez lire sur l'interface ***TinyBootloader*** les indications d'un téléchargement valide. Le programme se lance automatiquement.
- ✓ Vous devez observer le clignotement des Dels verte et rouge connectées aux lignes A6 et A7 du microcontrôleur.
- ✓ Le programme est arrêté en coupant l'alimentation du robot.

EXERCICES COMPLÉMENTAIRES

- ➢ Modifier le programme pour allumer en alternance soit les 2 Dels rouge et verte reliées aux lignes B6 et B7 soit celles reliées à A6 et A7.
- ➢ Générer une fréquence de 1kHz sur le piezo relié à la ligne B4.
- ➢ Réaliser un déplacement du robot en marche avant pendant 1s.

NOTES

Sélectionner les 3 premières lignes du code source puis faites un copier-coller dans un nouveau fichier. Nommer ce fichier config.h. Remplacer les 3 précédentes lignes par :

```
#include "config.h"
```

Ce nouveau fichier va être utilisé pour ajouter des constantes propres au robot.

Exemple :

Ajouter la ligne : #define Piezo pin_B4 // Piezo sur ligne B4

Compléter avec les 4 lignes pour les deux moteurs

Ajouter des commentaires (nom du fichier, auteur, dernière date, contenu)

A la suite des 3 premières lignes, ajouter la ligne suivante :

#org 0x0F00, 0x0FFF {} // Bootloader Protection

CAPTURER UNE ENTRÉE

OBJECTIFS

Apprendre à utiliser les entrées logiques.

DOCUMENTS

Instructions en C. Schéma du montage

EXERCICE 2

- Enregistrer l'exercice suivant sous le nom Exo_02.c
- Lancer ***Mplab*** et créer un projet Exo_02 dans votre répertoire avec le fichier Exo_02.c, config.h.
- Modifier votre programme avec le listing suivant.

```
#include "config.h"
void    main(void)
{
        output_A (0);                   // Mise à 0 des lignes en sortie
        output_B (0);
        set_tris_A (0b00011111);        // Direction des lignes du port A
        set_tris_B (0b00000101);        // Direction des lignes du port B

        while (1)
        {
                set_tris_a (0x1F);      // mise en entrée de la ligne A4
                if (input(BP))
                {
                        set_tris_a (0x0F);      // 0000 1111
                        output_toggle (Mot_Gau_Av);
                        output_toggle (Mot_Gau_Ar);
                }
        }
}
```

REMARQUES :

Les 4 premières instructions permettent de définir la direction des lignes des ports du µC et de mettre à 0 les lignes en sortie, afin de réduire la consommation du montage global.

if (input(BP)) permet de tester l'état du bouton. Si la condition est vraie, les instructions entre les accolades sont exécutées.

if (input(BP)) est équivalent à : ***if (input(BP)==1)***

TESTER VOTRE PROGRAMME

- ✓ Vous devez observer l'allumage ou l'extinction des Dels du moteur gauche à chaque appui du bouton poussoir. Mais le programme n'est pas stable.

EXERCICES COMPLÉMENTAIRES

- Ajouter une seule instruction pour que le programme soit stable.
- Modifier le programme pour activer le robot avec une télécommande infrarouge. Détecter la réception d'une commande quelconque issue d'une télécommande TV pour faire avancer le robot. En l'absence d'une commande, le robot s'arrête.
- Créer une fonction Appui_BP qui renvoie l'état du bouton poussoir.
- Créer un fichier fonctions.h afin d'y placer les fonctions créées.
- Créer une fonction Init_Robot pour lancer les 4 premières lignes du programme.

NOTES

```
#define     Rec     Pin_B0                  // déclaration du récepteur infrarouge
```

Exceptionnellement, la fonction Init_Robot sera placée dans le fichier config.h, mais ce sera la seule. Toutes les autres seront placées dans le fichier fonctions.h

SOUS PROGRAMME D'INTERRUPTION

OBJECTIFS

Rappel sur les interruptions

DOCUMENTS

Instructions en C. Timer0 en interruption

EXERCICE 3

- Enregistrer l'exercice suivant sous le nom Exo_03.c. Lancer ***Mplab*** et créer un projet Exo_03 dans votre répertoire avec les fichiers Exo_03.c, config.h et fonctions.h
- Modifier votre programme avec le listing suivant :

```
#include "config.h"
#include "fonctions.h"
#include "interruptions.h"
//-----------------------------------------------
void    main(void)
{
        Init_Robot ();
        Init_Interruptions ();
        while (1)
        {
                Synchro = 0;
                while (!Synchro);               // Drapeau venant de l'interruption

                Output_toggle (Del_Ve);         // Alterne la Del verte
        }
}
```

REMARQUES :

La fonction ***Init_Interruptions***, paramètre le timer0 pour déclencher une interruption sur débordement de son compteur.

Synchro = 0;

while (!Synchro); ces deux lignes permettent de bloquer le programme en attendant la validation par l'interruption. Cette technique permet de synchroniser les fonctions qui suivent afin qu'elles soient exécutées à intervalle de temps fixe.

SOUS PROGRAMME D'INTERRUPTION

Créer un fichier interruptions.h dans lequel vous allez placer le listing suivant :

```
void    Init_Interruptions (void)
{
        setup_timer_0 (RTCC_INTERNAL|RTCC_DIV_1);
        enable_interrupts(INT_TIMER0);
        enable_interrupts(GLOBAL);
}
//-----------------------------------------------
int     _100us;                 // déclaration d'une variable temps
short   Synchro;                // déclaration d'une variable drapeau
#Int_Timer0
void    _OS(void)               // Interruptions toutes les 100µs
{
        Set_Timer0 (??);        // Réinitialise le timer0 pour la prochaine interruption
        if (++_100us == 1000)           // Compte jusqu'à ??
        {
                _100us = 0;
                Synchro = 1;            // Valide la boucle principale
        }
//              Vous ajouterez ici les instructions pour générer les signaux MLI, voir Exo_4
}
```

MISE AU POINT DU PROGRAMME

- Compléter l'instruction Set_Timer0 (??) avec une valeur afin d'obtenir une interruption toutes les 100µs. Utiliser le mode Debugger et un StopWatch pour ajuster précisément cette valeur. Attention à la vitesse du quartz en mode simulation.
- Modifier la valeur 1000 pour obtenir un clignotement de la Del verte à 500ms.

NOTES :

Les timers sont des registres internes qui s'incrémentent tout seul, et redémarre à 0 lorsque la valeur maximale est atteinte. On peut les assimiler à des compteurs.

Le timer 0 est un registre 8 bits qui va de 0 à 255. Avec un oscillateur à 8MHz, le maximum est atteint au bout de 128 µs.

L'horloge qui incrémente le timer 0 est reliée à l'oscillateur interne (RTCC_Internal). L'horloge peut passer par un prédiviseur avant d'incrémenter le timer. T1_Div_By_2 divise par 2 l'horloge et donc multiplie par 2 le temps final.

Il y a 2 autres timers dans le PIC16F88, le timer 1 et le timer 2.

Le timer 0 est souvent utilisé pour générer l'horloge temps réel. Le timer 1 est un compteur 16 bits à usage général. Le timer 2, registre 8 bits, est utilisé pour générer les signaux périodiques comme les MLI de commande des moteurs.

DÉPLACEMENTS ÉLÉMENTAIRES

OBJECTIFS

Créer deux fonctions MLI logicielles pour les moteurs.

DOCUMENTS

Instructions en C.

EXERCICE 4

- Enregistrer l'exercice suivant sous le nom Exo_04.c. Lancer ***Mplab*** et créer un projet Exo_04 dans votre répertoire avec les fichiers Exo_04.c, config.h, fonctions et interruption.h
- Modifier votre programme avec le listing suivant et créer la fonction :

  void Moteurs (signed Vit_Gau, signed Vit_Dro)
- Compléter le fichier interruption.h avec les routines MLI logicielles.

```
#include "config.h"
#include "fonctions.h"
#include "interruptions.h"
//-----------------------------------------------
void    main(void)
{
        Init_Robot ();
        Init_Interruptions ();
        while (1)
        {
                if (Appui_BP())                 // Appui sur le bouton poussoir
                {
                        delay_ms (500);         // Attente bouton poussoir relaché
                        Moteurs (10,10);        // Robot avance
                        delay_ms (1000);
                        Moteurs (-10,-10);      // Robot recule
                        delay_ms (1000);
                }
                else
                {
                        Moteurs (0,0);          // Arrêt des moteurs
                }
        }
}
```

REMARQUES :

Moteurs (10,10) envoie un ordre de vitesse aux moteurs gauche et droit, mais ne peut pas modifier les variables directement dans la routine d'interruption.

TESTER VOTRE PROGRAMME

- ✓ Vous devez observer que le robot avance pendant une seconde sur appui du bouton poussoir puis recule pendant une seconde et s'arrête.
- ✓ Mesurer le temps réel. Que pouvez en conclure ?

EXERCICES COMPLÉMENTAIRES

- ➢ Modifier le programme pour que le robot dessine un carré de 20cm de coté environ sur appui sur le bouton poussoir. Utiliser une variable pour effectuer 4 fois avance puis tourne.
- ➢ Modifier le programme précédent afin de faire dessiner une spirale au robot. Utiliser une variable pour incrémenter les vitesses.

NOTES :

La vitesse maximale sera obtenue pour la valeur 20. Donc l'instruction Moteurs (10,10) demande aux moteurs de tourner à 50% de leur vitesse maximale.

Mesurer la distance parcourue par le robot en ligne droite en une seconde à différentes vitesses.

Mesurer le temps que met le robot pour faire un tour sur lui-même.

Quelle est la vitesse minimale pour laquelle le robot n'avance plus ?

UTILISATION DE LA LIAISON SÉRIE

OBJECTIFS

Apprendre à dialoguer avec un PC par la liaison série RS232.

Apprendre à mettre en forme les variables.

EXERCICE 5

- Ajouter la ligne suivante au fichier config.h à la suite des autres #use :

  #use rs232 (baud=19200, xmit=Pin_B5, rcv=Pin_B2)

- Compléter votre programme avec le listing suivant.

```
#include "config.h"
#include "fonctions.h"
#include "interruptions.h"
//-----------------------------------------------
void    main(void)
{
        Init_Robot ();
        Init_Interruptions ();
        printf ("\r\n Controle Robot\r\n");
        printf ("\r\n (A)vance, (R)ecule, (D)roit, (G)auche\r\n");
        while (1)
        {
                switch (toupper(getc()))
                {
                        case 'A':
                                Del_On;
                                Moteurs (10,10);          // Robot avance
                                break;
                        case 'R':
                                Del_On;
                                Moteurs (-10,-10);        // Robot recule
                                break;
                        case 'D':
                                Del_On;
                                Moteurs (10,-10);         // Rotation vers la droite
                                break;
                        case 'G':
                                Del_On;
                                Moteurs (-10,10);         // Rotation vers la gauche
                                break;
                }
                delay_ms (200);
                Del_Off;
                Moteurs (0,0);              // Arrêt des moteurs
        }
}
```

REMARQUES :

La directive #use rs232 (baud,xmit,rcv) permet de valider les fonctions suivantes:

printf ("\r\n Contrôle Robot\r\n")	envoie un texte vers le PC
getc()	attend un caractère (ex : touche du clavier)
printf(" Robot : %c\r\n",Etat)	envoie un texte et le contenu d'une variable

TESTER VOTRE PROGRAMME

- ✓ Vous pouvez utiliser le Terminal intégré à ***TinyBootloader*** pour envoyer des commandes du PC au robot. Mais Windows possède un logiciel intégré, ***Hyperterminal***, qui est plus agréable à utiliser

EXERCICES COMPLÉMENTAIRES

- ➢ Modifier le programme pour afficher l'état du robot sur le PC, en fonction des commandes clavier. Utiliser l'instruction : *printf(" Robot : %c\r\n",Etat)*
- ➢ Améliorer votre programme en modifiant la vitesse par les touches P(lus) et M(oins).
- ➢ Afficher le mouvement et la vitesse sur le PC.

NOTES :

HyperTerminal est accessible depuis Programmes/Accessoires/Communications

Il devra être paramétré en COMX (dépend de l'USB), 19200 baud, pas de contrôle de flux.

Si vous utilisez ***HyperTerminal***, il faut arrêter ***TinyBootloader.*** Aussi, il faut ensuite arrêter ***HyperTerminal*** pour reprogrammer la cible avec ***TinyBootloader***.

SUIVRE UNE LIGNE NOIRE

OBJECTIFS

Utilisation de capteurs optiques, traitement logique

EXERCICE 6

- Modifier la valeur de durée de la boucle d'interruption pour 100ms.
- Modifier votre programme avec le listing suivant.

```
#include "config.h"
#include "fonctions.h"
#include "interruptions.h"
//-----------------------------------------------
Byte Const Table [7] = {0,5,3,4,1,0,2};
Int       Ligne;

void      main(void)
{
          Init_Robot ();
          Init_Interruptions ();
          Moteurs (0,0);              // Moteurs stoppés
          printf ("\r\n Robot Suiveur de Ligne\r\n");
          printf ("\r\n Lecture des capteurs\r\n");

          while (1)
          {
                    Synchro = 0;
                    while (!Synchro)  // Boucle exécutée toutes les 100ms

                    Ligne = input_A();
                    Ligne = Ligne & 0b00001110;
                    Ligne >>=1;
                    Ligne = Table [Ligne];
                    printf (" Ligne : %u\r\n",Ligne);
          }
}
```

REMARQUES :

Byte const table[7] = {0,5,3,4,1,0,2};	Déclaration d'une table en ROM
Y = Table [X];	Lecture d'une valeur dans la table
Y = input_A();	Lecture d'un port en entier
Ligne >>=1	Décalage d'un bit, équivalent à diviser par 2

TESTER VOTRE PROGRAMME

- ✓ Déplacer le robot sur une ligne noire et observer les valeurs envoyées sur le PC.

EXERCICES COMPLÉMENTAIRES

- Modifier le programme pour que le robot suive une piste noire sur fond blanc. Mettez en commentaire l'instruction printf dans la boucle et réduisez à 10ms la durée de la boucle. Utiliser la variable Ligne pour commander indirectement les moteurs.
- Êtes-vous satisfait du comportement du robot suiveur de ligne ?
- Que se passe t-il en cas d'intersection ? Corriger le programme.

NOTES :

L'instruction printf est très pratique pour débugger un programme, mais ralentit la boucle infinie. Il est conseillé de la mettre en commentaires pour les tests réels.

UTILISATION DES ENTRÉES ANALOGIQUES

OBJECTIFS

Apprendre à utiliser les entrées analogiques. Apprendre à mettre en forme les variables.

EXERCICE 7

- Compléter votre programme avec le listing suivant.

```
#include "config.h"
#include "fonctions.h"
#include "interruptions.h"
//-----------------------------------------------
int     Mesure_Bat;

void    main(void)
{
        Init_Robot ();
        Init_Interruptions ();

        setup_adc_ports( sAN0 | sAN1 | sAN2 | sAN3);
        setup_adc(ADC_CLOCK_INTERNAL );

        printf ("\r\n Entrees Analogiques\r\n");
        printf ("\r\n Lecture de la tension batterie\r\n");
        delay_ms (1000);

        while (1)
        {
                Synchro = 0;
                while (!Synchro)          // Boucle exécutée toutes les 500ms

                set_adc_channel(0);
                delay_us (10);
                Mesure_Bat = read_adc();
                printf (" Valeur lue = %u\r\n",Mesure_Bat);
        }
}
```

REMARQUES :

setup_adc_ports (sAN0 | sAN1 | sAN2 | sAN3); sélectionne les lignes AN0 à AN3

setup_adc (Adc_Clock_Internal); sélectionne la vitesse de lecture

set_adc_channel (0) sélectionne la ligne AN0 qui est reliée au convertisseur, car il n'y a qu'un seul convertisseur dans le composant.

Y = read_adc() lit la valeur numérique convertie

TESTER VOTRE PROGRAMME

- ✓ Vous pouvez lire la valeur de la tension batterie sur votre PC. Cette valeur n'est pas simple à lire.

EXERCICES COMPLÉMENTAIRES

- Compléter le programme afin de convertir la valeur numérique lue en Volt avec un chiffre après la virgule.
- Ajouter une condition qui déclenche le piezo si la tension de la batterie est inférieure à 4,4V. Et créer la fonction Mesure_Bat() qui regroupe les diverses possibilités.
- Modifier le programme afin de lire les tensions issues des capteurs optiques du suiveur de ligne. Afficher les valeurs numériques sur le PC. Observer la sensibilité lors des déplacements latéraux du robot. En conclure sur une amélioration du programme de suivi de ligne.

NOTES :

setup_adc_ports(sAN0 | sAN1 | sAN2 | sAN3);

setup_adc(ADC_CLOCK_INTERNAL);

Déplacer les deux instructions précédentes dans une fonction Init_CAN () dans le fichier fonctions.h.

DÉTECTER UN OBSTACLE

OBJECTIFS

Apprendre à utiliser le Timer 2 et la ressource interne MLI

EXERCICE 8

➢ Compléter votre programme avec le listing suivant.

```
int     IR_Etat;                                        // Variable d'état IR

void    main(void)
{
        Init_Robot ();
        Init_Interruptions ();
        Moteurs (0,0);
        printf ("\r\n Exo Timer\r\n");
        printf ("\r\n Detection Obstacle\r\n");
        delay_ms (1000);

        setup_ccp1(CCP_PWM);                            // Configure RB3 en sortie MLI
        setup_timer_2(T2_DIV_BY_1, 53, 1);              // Fréquence 36 kHz
        set_pwm1_duty(28);                              // rapport cyclique = 50%

        while (1)
        {
                Synchro = 0;
                while (!Synchro)                // Boucle exécutée toutes les 100ms

                IR_Etat = 0;
                output_high (Del_IR);       // Emission sur Del IR droite
                delay_us(600);
                if (!input(Rec))            // Si obstacle
                        IR_Etat = 1;
                output_low (Del_IR);        // Emission sur Del IR gauche
                delay_us(600);
                if (!input(Rec))            // Si obstacle
                        IR_Etat += 2;
                printf (" Zone : %u\r\n",IR_Etat);
        }
}
```

REMARQUES :

setup_ccp1 (CCP_PWM) configure la ligne CCP1 pour générer un signal MLI.

Setup_timer_2 (T2_DIV_BY_1, 53, 1) définit la période du signal.

set_pwm1_duty (28) définit le temps haut.

TESTER VOTRE PROGRAMME

- ✓ Déplacer votre main devant le robot pour constater la détection à droite à gauche, devant ou pas de votre main.

EXERCICES COMPLÉMENTAIRES

- ➢ Modifier le programme pour que le robot avance lorsqu'il détecte votre main.
- ➢ Modifier à nouveau le programme pour qu'il suive votre main.

NOTES :

Le timer 2 est un registre 8 bits qui va de 0 à 255. Avec un oscillateur à 8MHz, on peut obtenir un signal de période 128 µs maximum. Mais si on prédivise avant d'utiliser le compteur, on peut augmenter la période encore. Pour réduire la période, il faut donner une valeur maximale avec la commande setup_timer_2

L'horloge qui incrémente le timer 2, est reliée à l'oscillateur interne. T2_Div_By_2 divise par 2 l'horloge et donc multiplie par 2 la période du signal.

Par la suite, les 3 lignes de configurations doivent être déplacées dans une fonction Init_IR().

TÉLÉCOMMANDE INFRAROUGE RC5

OBJECTIFS

Utilisation du Timer 1, utiliser une routine fournie par une tierce personne.

EXERCICE 9

- Compléter votre programme avec le listing suivant. Un nouveau fichier est appelé par le programme. Copier ce fichier : RC5.h dans le répertoire avec les autres fichiers.

```
#include "config.h"
#include "fonctions.h"
#include "interruptions.h"
#include "RC5.h"
//-----------------------------------------------
void    main(void)
{
        Init_Robot ();
        Init_Interruptions ();
        Moteurs (0,0);
        Init_RC5();
        printf ("\r\n Exo Robot\r\n");
        printf (" Code RC5\r\n");
        delay_ms (1000);
        Del_Off;
        RC5_Etat = 0;

        while (1)
        {
                Synchro = 0;
                while (!Synchro)            // Boucle exécutée toutes les 200ms

                Process_RC5();
                if (RC5_Valid)
                {
                        RC5_Valid = 0;
                        printf("RC5 : %1u %2u %2u\r\n",RC5_Rep,RC5_Adr,RC5_Code);
                }
        }
}
```

REMARQUES :

Setup_Timer_1(T1_internal|T1_div_by_1)	Le timer 1 est configuré
Set_Timer1(0)	On met à zéro le timer 1
Temps = get_timer1();	On lit le contenu du timer 1

TESTER VOTRE PROGRAMME

- ✓ Utiliser une télécommande avec codage RC5 (Phillips). Vous devez lire sur votre PC, les valeurs des touches sélectionnées.

EXERCICES COMPLÉMENTAIRES

- ➢ Modifier le programme pour piloter le robot dans les quatre directions.
- ➢ En vous aidant d'un document sur la trame RC5, essayer d'analyser la machine d'états dans l'interruption #int_ext.

NOTES :

Le décodage d'une trame RC5 est obtenu par une machine d'état dans l'interruption déclenché par chaque front descendant.

Le fichier RC5.h est constitué de plusieurs fonctions.

Init_RC5() initialise le timer 1 utilisé et l'interruption sur l'entrée RB0.

Process_RC5() permet de valider un code reçu et de le décoder.

#int_Timer1 contrôle les erreurs de temps.

#int_ext reçoit la séquence de bits correspondant au code tout en décodant chaque bit.

ARCHITECTURE À INTERACTIONS PRIORITAIRES

OBJECTIFS

Programmation d'une architecture à interactions prioritaires.

Utilisation de machine d'états finis.

Programmation d'un robot sumo.

EXERCICE 10

- La structure des comportements est donnée au chapitre Listing.
- Compléter chaque comportement indépendamment les uns des autres.
- L'écriture la plus adaptée pour écrire les comportements et les fonctions de lecture de certaines entrées est la machine d'états

```
void    main(void)
{
        Init_Robot ();
        Init_Interruptions ();
        Init_CAN ();
        Init_IR ();

        while (1)
        {
                Synchro = 0;
                while (!Synchro)            // Boucle exécutée toutes les 500µs

                Process_Capteurs_Sol();
                Process_Capteurs_IR();

                Process_Bordure();
                Process_Cherche();
                Process_Defaut();

                Process_Arbitre();
        }
}
```

TESTER VOTRE PROGRAMME

- ✓ Tester chaque bout de programme aussi souvent que possible.

EXERCICES COMPLÉMENTAIRES

- Améliorer le programme.

LISTING

Trame des comportements et sous-programme de l'arbitre :

```
//=============================================================
//        Comportement par défaut                    Durée du processus : 4,5µs
//=============================================================
short Defaut_Drap = 0;
signed int Defaut_Vit_Gau, Defaut_Vit_Dro;

void    Process_Defaut()
{
        Defaut_Vit_Gau = 10;
        Defaut_Vit_Dro = 10;
        Defaut_Drap = 1;
}
//=============================================================
//        Comportement pour chercher l'adversaire    Durée du processus :
//=============================================================
short Cherche_Drap = 0;
signed int Cherche_Vit_Gau, Cherche_Vit_Dro;

void    Process_Cherche()
{
        if (Cherche_Etat == Attente)            // rien en vue
        {
                Cherche_Drap = 0;

//                              Écrire votre code ici afin de changer d'état
        }
        else if (Cherche_Etat == Droite)        // Obstacle à droite
        {
                Cherche_Vit_Gau = 10;
                Cherche_Vit_Dro = -10;
                Cherche_Drap = 1;

//                              Écrire votre code ici afin de revenir à l'état initial
        }
        else if (Cherche_Etat == Gauche)        // Obstacle à gauche
        {
                Cherche_Vit_Gau = -10;
                Cherche_Vit_Dro = 10;
                Cherche_Drap = 1;

//                              Écrire votre code ici afin de revenir à l'état initial
        }
        else                                    // Obstacle devant
        {
                Cherche_Vit_Gau = 10;
                Cherche_Vit_Dro = 10;
                Cherche_Drap = 1;

//                              Écrire votre code ici afin de revenir à l'état initial
        }
}
```

```
//=============================================================
//        Comportement pour éviter de tomber du dohyo       Durée du processus :
//=============================================================
short Bordure_Drap = 0;
signed int Bordure_Vit_Gau, Bordure_Vit_Dro;
int       Bordure_Tempo, Bordure_Mem;

void      Process_Bordure()
{
          switch (Bordure_Etat)
          {
          case Attente:
                    Bordure_Drap = 0;                         // Pas de contrôle du robot

//                                  Écrire votre code ici afin de changer d'état

                    break;
          case Recule:
                    Bordure_Vit_Gau = -10;              // en arrière 50%
                    Bordure_Vit_Dro = -10;
                    Bordure_Drap = 1;                         // demande de contrôle du robot

//                                  Écrire votre code ici afin de changer d'état

                    break;
          case Tourne:
                    Bordure_Vit_Gau =
                    Bordure_Vit_Dro =
                    Bordure_Drap = 1;

//                                  Écrire votre code ici afin de revenir à l'état initial

                    break;
          default:
                    Bordure_Etat = Attente;
                    break;
          }
}
```

```
//=============================================================
//        Sous-Programme d'arbitrage                    Durée du processus :
//=============================================================
void    Process_Arbitre()
{
signed int Arbitre_Vit_Gau, Arbitre_Vit_Dro;
        if (Bordure_Drap)
        {
                Arbitre_Vit_Gau = Bordure_Vit_Gau;
                Arbitre_Vit_Dro = Bordure_Vit_Dro;
        }
        else if (Cherche_Drap)
        {
                Arbitre_Vit_Gau = Cherche_Vit_Gau;
                Arbitre_Vit_Dro = Cherche_Vit_Dro;
        }
        else
        {
                Arbitre_Vit_Gau = Defaut_Vit_Gau;
                Arbitre_Vit_Dro = Defaut_Vit_Dro;
        }
        Moteurs (Arbitre_Vit_Gau, Arbitre_Vit_Dro);
}
//=============================================================
```

SCHÉMA DE LA CARTE PRINCIPALE

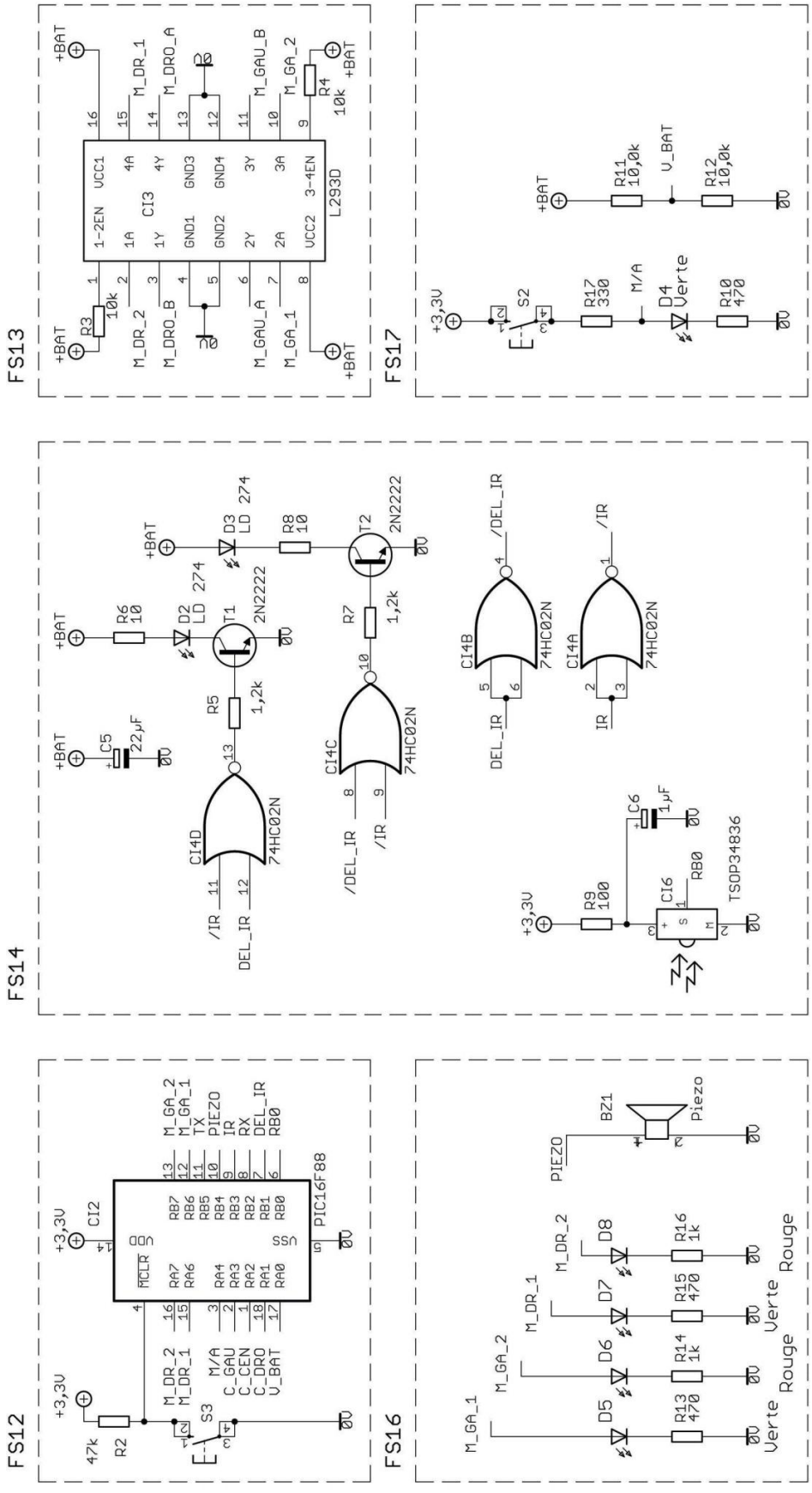


SCHÉMA DES CARTES DE DÉTECTION SOL

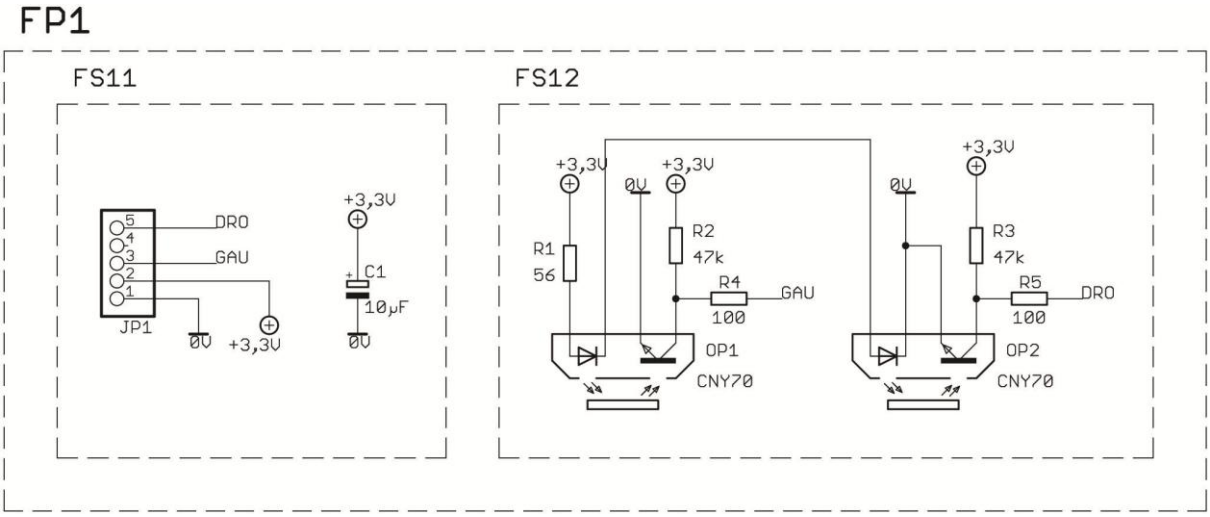


Schéma de la carte de détection de la bordure

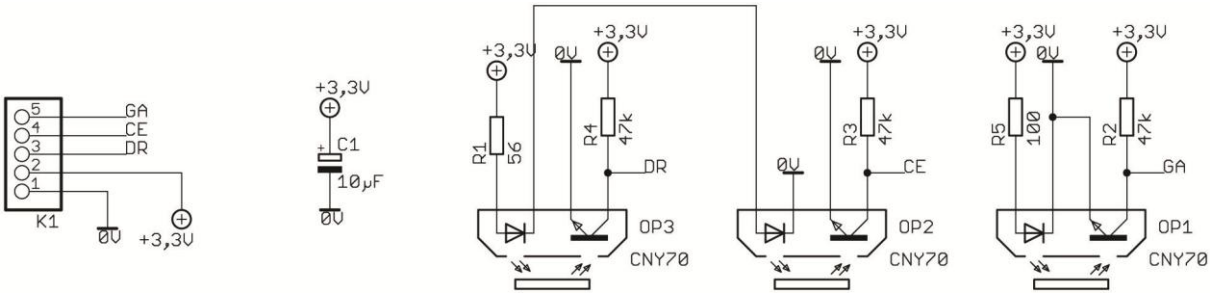


Schéma de la carte de détection de la ligne

RÉFÉRENCES

[1] Base mécanique : www.robitec.fr

[2] Site de l'auteur : www.geii.iut-nimes.fr/fg

[3] Composant Microchip (data sheet) et logiciel MPLAB : www.microchip.com

[4] Compilateur C de CCS : www.ccsinfo.com

[5] Tiny PIC Bootloader : www.etc.ugal.ro/cchiculita/software/picbootloader.htm

[6] module XBee de Maxstream: www.maxstream.fr