**Projet Robot 2**

**Contrat 7 : Caniche IR**

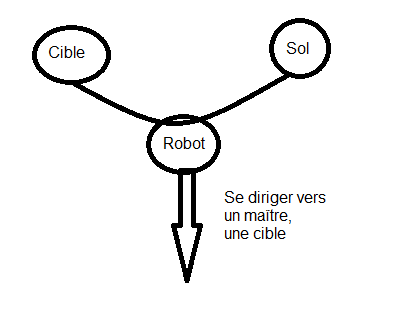


**I- Introduction :**

1) Présentation du contrat :

Le but ce contrat est de faire acquérir au robot un certain nombre de fonction lui permettant de se déplacer dans un environnement prédéfinit. Il doit être capable, grâce aux signaux issus des capteurs infrarouges (IR) gauche et droite, de se déplacer vers une cible mobile devant le robot. Si cette cible est à moins de 1,5m le robot avance vers celle-ci et s’arrête lorsqu’il est à moins de 40cm de la cible. Il doit se déplacer à une vitesse comprise entre 7 et 14 m/min.

2) Diagramme bête à cornes :



Le robot réagit avec la cible, et a comme support le sol.

**I- Les objectifs du contrat 7 :**

1) Objectifs impératifs :

a- mettre en marche et atteindre le robot en appuyant sur le bouton du centre de la télécommande.

b- acquérir les informations à travers le bus I²C, des informations fournies par la télécommande, avec une vitesse de transmission de 50kBauds.

c- afficher les données grâce aux LED, D1 à D8 commandées via le bus I²C.

d- mémoriser la tension issue de la batterie.

e- mesurer la tension issue de la batterie en faisant une moyenne de 4 valeurs sur un intervalle régulier.

f- générer les signaux modulés en largeurs d’impulsions droite et gauche avec une période de 1ms et un rapport cyclique strictement inférieur à 50%. En effet la tension d’alimentation de chaque moteur étant de 12V alors que les moteurs ont une tension nominale de 6V, le moteur tourne donc à sa vitesse maximale avec un rapport cyclique de 50%.

g- changer le sens de rotations des moteurs gauche et droite afin de changer le sens de rotations des chenilles de sorte àtourner.

h- allumer et éteindre les capteurs IR droite et gauche, afin d’alimenter uniquement les capteurs quand ils sont sollicités pour limiter la consommation énergétique de la carte.

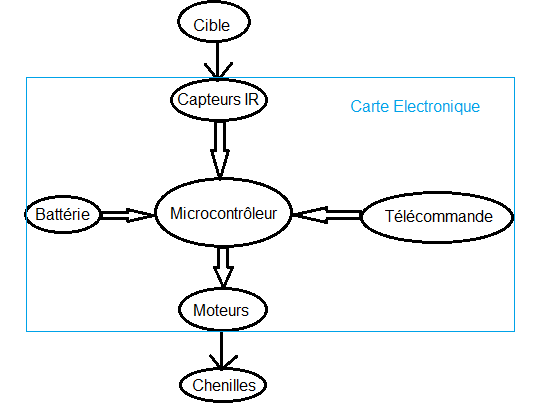
i- acquérir les tensions issues des deux capteurs Infra Rouge droite et gauche donnant une représentation de la distance entre le capteur et l’obstacle.

j- utiliser la liaison série RS232 avec une vitesse de 9,6kBauds, afin d’afficher la tension moyenne de la batterie, la fin de l’initialisions des variables ainsi que leur état.

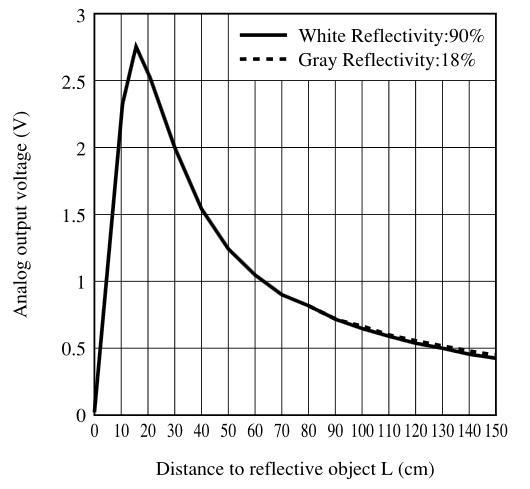
k- réaliser des interruptions via la télécommande et le timer0.

m- Utiliser un minimum de mémoire pour la ram et le code, qui sera sauvegardé sur une ROM programmable et effaçable électriquement. Pour cela, nous portons une attention particulière à la taille des variables en fonction de leur utilisation.

2 ) Diagramme pieuvre :



Pour réagit avec le cible, le robot se sert de deux capteurs infra-rouges, qui fournissent une tension inversement proportionnelle à la distance entre la cible et le capteur, pour une distance comprise entre 15cm et plus 1,5 m, comme l'illustre le courbe si dessous :

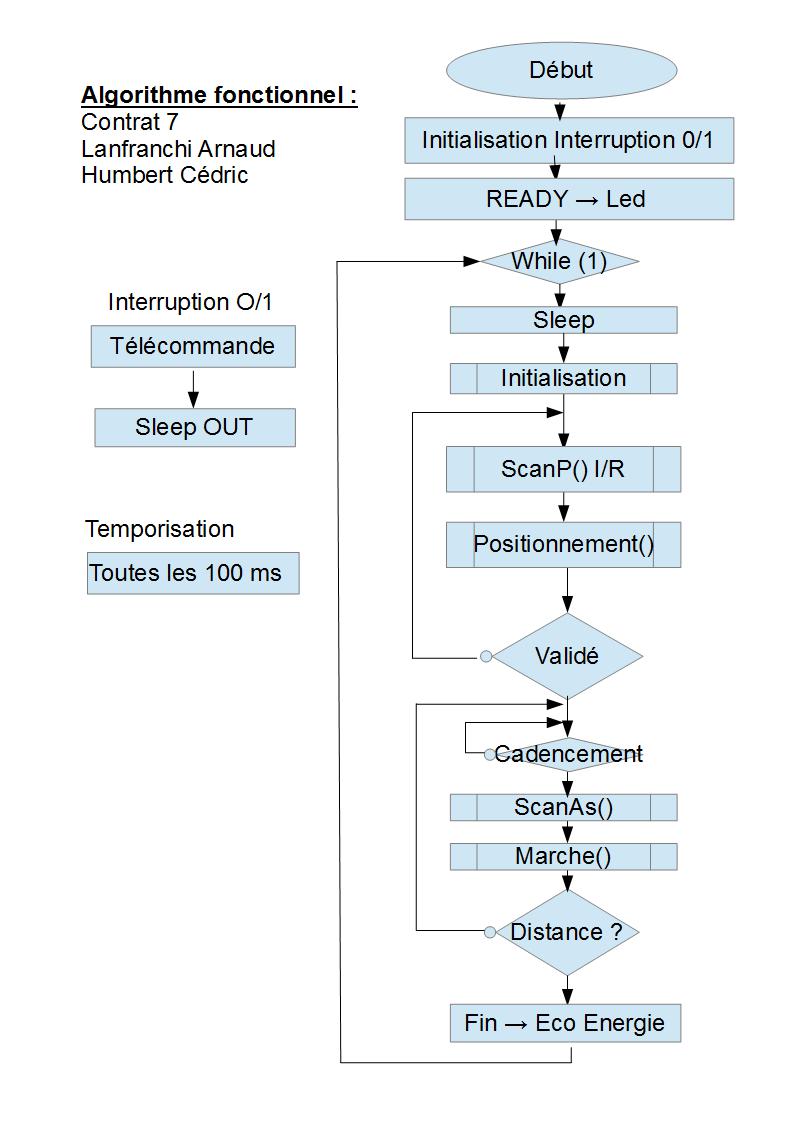


**II- Repérage des composants nécessaires pour atteindre les objectifs impératifs :**

|  |  |
| --- | --- |
| Fonctions | Composants concernés ainsi que leurs broches |
| on/off du robot via bouton centrale de la télécommande via la liaison I²C qui relie le µC et le bloc de réception de la télécommande | RX1 broche 3 in analog  broche 14 out analog  U4 PIC16 broche 2 in analog P2A  broche 6 out digital RC4  broche 9 out digital (data I²C) SDA  broche 10 out digital (clock I²C) SCL  U1PIC18 broche 21 in digital INTO  broche 14 in digital (clock I²C) SCL  broche 15 out digital (data I²C) SDA  broche 21in digital INT0 (IT) |
| afficher données I²C grâce aux LED D1 à D8 | U8 PCF8574 broche 14 in digital clock I²C  broche 15 in digital data I²C  broche 4 out digital LED D1 P0  broche 12 out digital LED D8 P7 |
| mesurer, mémoriser et faire une moyenne de 4 valeurs la tension issue de la batterie | U1PIC18 broche 4 in analog AN2 (SURVBAT) |
| générer signaux PWM avec T=1ms et α<0,5. | U1PIC18 broche 12 out digital CCP2 (PWMD)  broche 13 out digital CCP1 (PWMG)  Timer 1 et 2 |
| changer le sens de rotation des moteurs | U1PIC18 broche 9 out digital RA7 (DIRG)  broche 10 out digital RA6 (DIRD) |
| allumer et éteindre les capteurs IR | U1PIC18 broche 22 out digital RB1 |
| acquérir les tensions issues des deux capteurs IR | U1PIC18 broche 2 in analog AN0 (CAPTD)  broche 3 in analog AN1 (CAPTG) |
| utiliser la liaison série RS232 avec une vitesse de 9,6kBauds, afin d’afficher la tension moyenne de la batterie, la fin de l’initialisions des variables ainsi que leur état. | U1PIC18 broche 17 out analog TX (TX1)  broche 18 in analog RX (RX1)  MAX232 broche 11 in analog T1IN (TX)  broche 12 out analog R1OUT (RX) |
| réaliser des interruptions via la télécommande et le timer0 | U1PIC18 broche 22 out digital INT1  broche 23 out digital INT2 (CAPTARR) |
| faire un système économique en énergie | U1PIC18 broche 22 out digital INT1  broche 23 out digital INT2 (CAPTARR) |

**III- Algorigramme fonctionnel :**

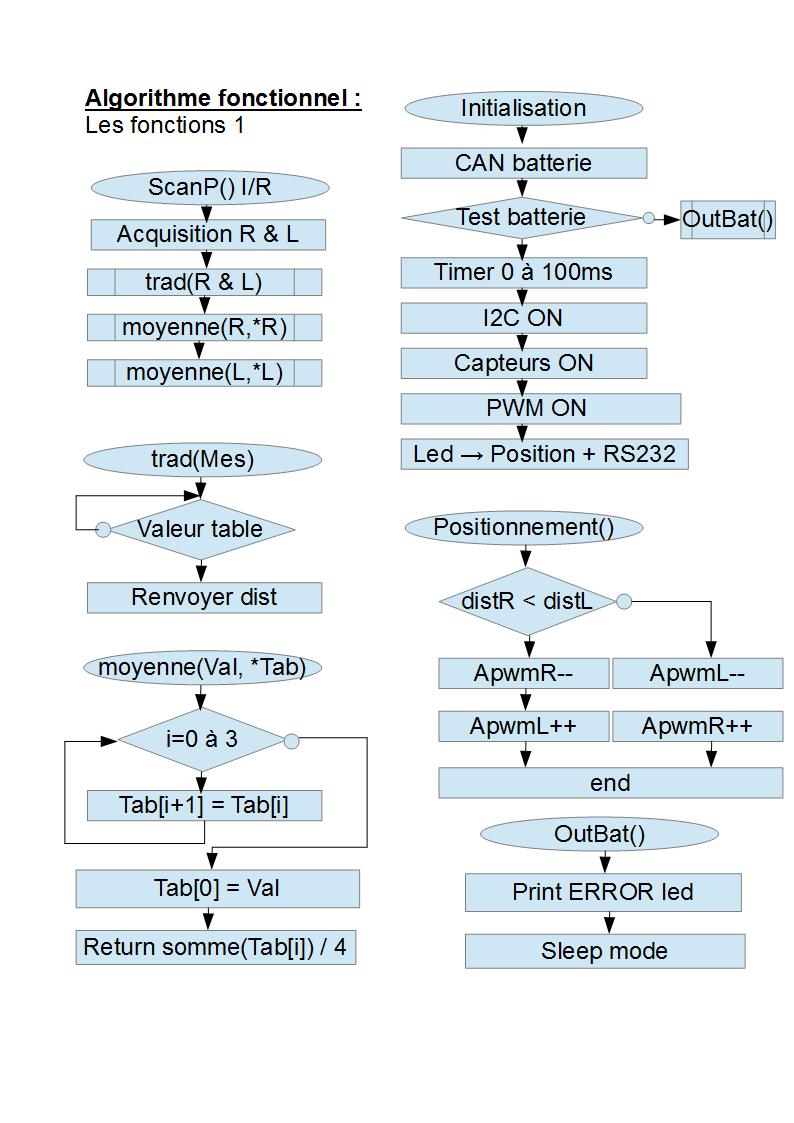
1) Organigramme :



Le programme de contrôle du robot tel que décrit plus haut dans une modélisation fonctionnelle est décomposé et organisé en mémoire tel que suit.

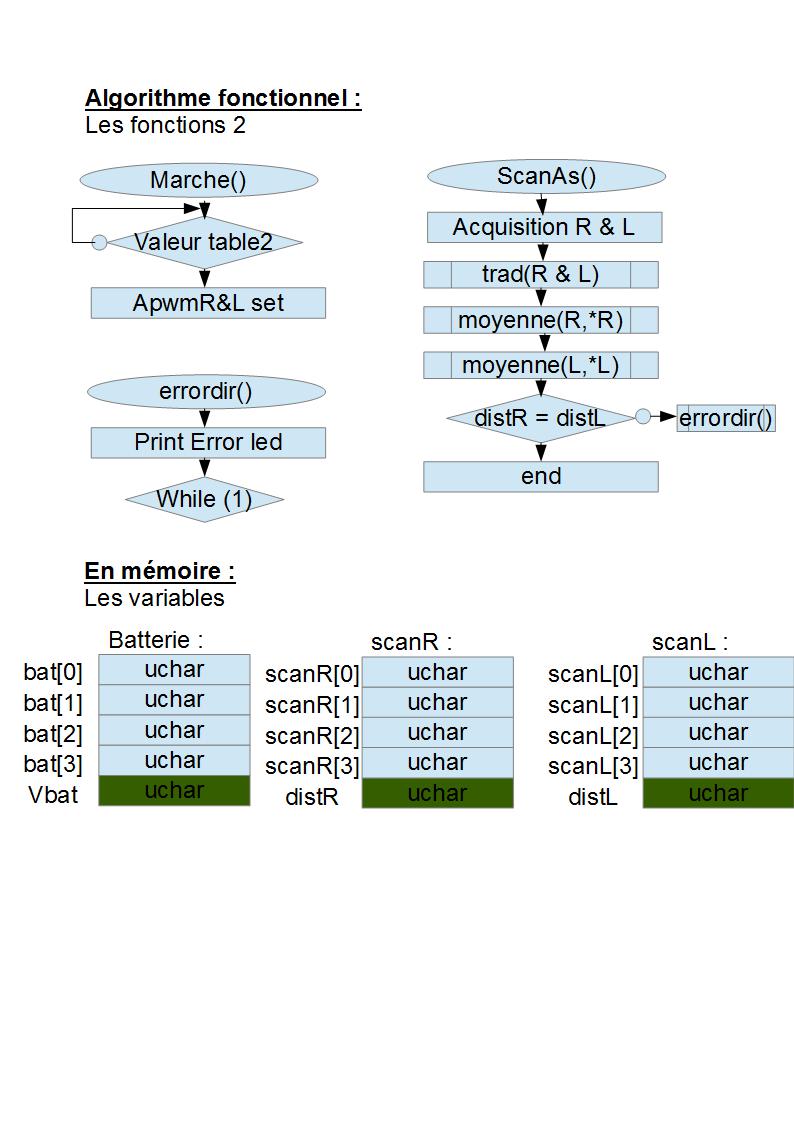
Le projet contiendra 4 fichiers, le fichier principal, mainC7VX.c contenant le main et le programme d’initialisation et les interruptions un header général, headC7VY.h ou l’on trouvera les prototypes des différentes fonctions. De plus, le projet contiendra un fichier C, fonctionC7VX.c contenant les fonctions nécessaires au fonctionnement du robot aussi qu’un fichier header contenant des tableaux de correspondance pour les différentes phases de pilotage du robot.

Ci-joint les algorigrammes des fonctions annexes.



Les fonctions présentées ici sont cependant soumises à des modifications et adaptations suite à des erreurs ou des bugs relevés lors de la phase de test. Les termes décrivant les actions sont volontairement orienté fonction plus que code, ils serviront en effet à commenter et organiser le code, la phase de code consistera à faire la correspondance entre les instructions et ces notions. En somme, certaines parties du code correspondent déjà à des parties vues en TD, il ne s’agit donc pas de les développer outre mesure. Les tables 1 et 2 (cf plus bas) sont des tables de correspondance pour la commande de la machine.

Suite des fonctions :



De plus, dans la fonction moyenne, i est un indice codé donc sur un char.

Les valeurs ApwmR et ApwmL sont les variables correspondant au rapport cyclique de la commande du PWM. Les valeurs L, \*L, R et \*R sont les symboles des tableaux et des valeurs représentés plus haut. Les entrées sur les CAN se feront sur 8bits en effet, notre table de correspondance est indexée sur des pas variables d’environ 25mV, un quantum de 5mV n’aurait donc aucun sens.

Il a été fait le choix d’utiliser des tableaux afin de gommer les erreurs de mesures liés à des fortes demandes de courant sur la batterie ou à un fort changement dans éclairage de la pièce.

La table 1 fait une correspondance entre les distances mesurées par les capteurs IR et la valeur en cm, le tableau excel où figurent cette table est joins. Ces tables se trouvent en ROM et seront donc étalonnés durant les tests. La table 2 fait correspondre des valeurs de ApwmR et ApwmL avec la distance retrouvée de sorte à générer une accélération et une décélération amorties dans le temps.

Les signes d’égalité en particulier dans la fonction ScanAs() sont des égalités relatives couvrant les erreurs éventuelles de mesure.

Nous avons fait le choix de diviser la fonction d’initialisation en deux parties en effet, nous désirons pour des raisons d’économie d’énergie mettre notre robot en mode sommeil lorsqu’il n’est pas utilisé, il nous faut donc procéder à un arrêt / allumage d’un certain nombre de ses périphériques que nous plaçons dans une fonction d’initialisation. Cependant, un certain nombre de réglage n’ont pas à être refaits lors de ces arrêt / marche donc nous procéderons aussi à une première initialisation en dehors de la fonction d’initialisation.

2) Découpage du programme en ficher .h .c :

Pour ce projet nous avons choisis la répartition suivante :

|  |  |
| --- | --- |
| Nom de fichier | Description |
| main.c | C'est la ficher principal du programme,  il fait appel à tous les fichers suivants et la plupart des fonctions comprises dans ces fichiers. |
| FonctionPrincipales.c | Ce fichier regroupe toute les fonctionnes indispensables pour respecter le cahier des charges et donc les objectives à atteindre impérativement. |
| SousFonctions.c | Ce fichier contient des fonctionnes utiles, mais facultatives, ou tout simplement non mentionné dans le cahier de charges. Elles permettent de réalisé certaines fonctionnes principales. Ce sont des outils. |
| headall.h | Ce fichier contient le prototype, les déclarations des fonctions principales. |
| headSousFonctions.h | Ce fichier contient le prototype, les déclarations des sous fonctions. |
| PIC18f2520.h | Ce ficher contient la bibliothèque du microcontrôleur utilisé |

3) Déclaration et description des variables et fonctions :

unsigned char Batterie[4] = {180, 180, 180, 180} : c’est un tableau contenant les valeures de la tension de la batterie initialiser au dessus du seuil

unsigned char ScanR[4] = {0, 0, 0, 0} : c’est un tableau pour stocker les valeurs de la tension le capteur droit

unsigned char ScanL[4] = {0, 0, 0, 0} : c’est un tableau pour stocker les valeurs de la tension le capteur gauche

unsigned char Vbat : c’est la valeur de la tension image de la batterie

unsigned char distR : c’est la valeur de la tension distance issu du capteur droit

unsigned char distL : c’est la valeur de la tension distance issu du capteur gauche

unsigned char ApwmR : c’est la valeur de la commande PWM Droite

unsigned char ApwmL : c’est la valeur de la commande PWM Gauche

unsigned char MesBat : Mesure de la tension batterie

unsigned char ComPWM[20] = {13, 14, 14, 15, 17, 18, 20, 22, 24, 26, 26, 26, 26, 26, 26, 23, 21, 18, 15, 13} :

Les fonctions sont réparties en 2 niveau, le premier pour les fonctions directement appelées par le main, un second pour exécuter des taches génériques.

Initialisation() : initialise le micro-controleur

Positionnement() : modules les PWM afin de positionner le robot pour qu'il se mette en face de la plaque par détaction et comparaison

ScanP() : scan avec les IR de la distance entre le robot et la plaque lors du Positionnement, remplit les tableaux scanL et scanR en mémoire.

ScanAs() : scan avec les IR de la distance entre le robot et la plaque lors de la Marche, remplit les tableaux scanL et scanR

Marche1() : génère la commande PWM pour les moteurs lors de la marche du moteur

Pour ce qui est des sous fonctions on a :

errordir1() : arrête le robot à une erreur de déplacement

OutBat() : en cas d’erreur sur la tenstion de la batterie

trad(unsigned char Mes) : converti la valeur retrouvée par les IR en une valeur exploitable pour le tableau de conversion

moyenne(unsigned char Val, unsigned char\* Tab) : fait une nouvelle moyenne des valeurs du tableau avec la nouvelle valeur

**IV- Détails des différentes fonctions :**

On utilise une fréquence de fonctionnement pour le PIC18F de 8MHz. Il faut donc configurer l’oscillateur interne du microcontrôleur via le registre OSCON :

D’après la page 32 de la documentation du PIC18 il faut mettre les bits IRCF2, IRCF1 et IRCF à ‘1’ :

OSCCONbits.IRCF2 = 1;

OSCCONbits.IRCF1 = 1 ;

OSCCONbits.IRCF0 = 1 ;

a- mettre en marche et atteindre le robot en appuyant sur le bouton du centre de la télécommande :

La télécommande génère une interruption sur la broche 21qui réveille le robot.

Le bouton central de la télécommande transmet le code 3h=0011% cela correspond au code ASCII 33h. Si la broche 14 (SDA) reçoit la donnée 33h=0011 0011%.

C’est la broche 0 du port B, c’est une entrée numérique en mode INT0 (interruption).

Il faut configurer cette broche en entrée : TRISBbits.RB0=1;

Pour ce qui est des interruptions, nous pensons qu’une gestion des priorités est adéquate, pour cela il faut mettre le bit IPEN du registre RCON à ‘1’ (RCONbits.IPEN=1 ;). Il faut activer les interruptions de hautes priorités (INTCONbits. GIE=1 ;) et les interruptions de basses priorités (INTCONbits.PEIE=1 ;)

Ensuite via le registre INTCON, il fait les valider (INTCONbits.INT0IE=1 ;) .

Puis il faut activer l’interruption pour la mise en marche du robot avec INTCONbits.INT0IF=0;

b- acquérir les informations I²C fournis par la télécommande, avec une vitesse de transmission de 50kHz=50kBauds :

Il faut mettre les broches 14 et 15 en entrée (TRISCbits.RC3=1; TRISCbits.RC4=1;) I²C (SSPCON1bits. SSPEN=1 ;) Avec cette dernière commande on utilise le mode SCL (respectivement SDA) de la broche 14 (respectivement broche 15). Pour avoir une vitesse de transmission de 50kHz, on sélectionne d’abord le mode Master I²C :

SSPCON1bits.SSPM3=1; SSPCON1bits.SSPM2=0 ; SSPCON1bits. SSPM1=0 ; SSPCON1bits.SSPM0=0 ;

On utilise la formule Avec Fosc=8MHz et SSPADD une valeur numérique.

= 0010 0111%

Puis on met la valeur 39 dans SSPADD : SSPADD=39 ;

d- mémoriser la tension issue de la batterie et e- mesurer la tension issue de la batterie en faisant une moyenne de 4 valeurs sur un intervalle régulier :

La tension de la batterie arrive sur la broche 4 du PIC18 (au fait c’est la tension de la batterie divisé par 3,2), il faut donc d’abord configurer la broche 4 en entrée (TRISAbits.RA2=1 ;) et analogique (PORTAbits.RA2=1 ;).

f- générer les signaux modulés en largueur d’impulsions :

Les signaux modulés en largeur d'impulsions, qui permettent des contrôles la vitesse de rotation des moteurs sont en sortie des broches 12 et 13 du microcontrôleur.

Il faut dans un premier temps configurer ces broches en sortie (TRISCbits.RC1=0;

TRISCbits.RC2=0;).

On souhaite un période PWM de 1ms soit une fréquence de 1kHz.

Le choix de la période PWM se fait grâce à la valeur stockée dans le registre

PR2 = 1. Cette valeur doit être inférieure à 255, le prescaler peut être seulement de 1,4 ou 16. Avec un prescaler de 4 PR2=499 ce qui est trop grand donc il faut un prescaler de 16 (T2CONbits.T2CKPS1=1;) PR2=124 avec ce prescaler.

Ensuite pour générer les signaux PWM il faut utiliser le mode CCP2 pour la broche 12 et CCP1 pour la broche 13.

CCP2CONbits.CCP2M3=1;

CCP2CONbits.CCP2M2=1;

CCP1CONbits.CCP1M3=1;

CCP1CONbits.CCP1M2=1;

CCP1CONbits.CCP1M1=0;

CCP1CONbits.CCP1M0=0;

h- allumer et éteindre les capteurs IR droite et gauche :

Le contrôle de l'alimentation des capteurs IR de fait grâce à la broche 22 du microcontrôleur. Il faut d'abord la configurer en sortie (TRISBbits.RB1=0;).

Ensuite pour alimenter les capteurs IR, il faut saturer le transistor pnp, donc mettre un niveau bas sur la base de ce transistor (PORTBbits.RB1=0;) et pour éteindre le capteur il faut bloquer le transistor avec un niveau haut sur sa base (PORTBbits.RB1=1;).

**V- le code :**

1) headall.h :

// Variables ApwmL & ApwmR 1 les tableaux en mémoire :

extern unsigned char Batterie[4] = {180, 180, 180, 180}; // Tableau des valeures Batterie initialisation au dessus du seuil

extern unsigned char ScanR[4] = {0, 0, 0, 0}; // Tableau pour le capteur Droit initialisation Sur petite distance

extern unsigned char ScanL[4] = {0, 0, 0, 0}; // Tableau pour le capteur Gauche initialisation Sur petite distance

extern unsigned char Vbat; // Valeur de la tension Batterie

extern unsigned char distR; // Valeur de la tension distance à Droite

extern unsigned char distL; // Valeur de la tension distance à Gauche

extern unsigned char ApwmR; // Valeur de la commande PWM Droite

extern unsigned char ApwmL; // Valeur de la commande PWM Gauche

extern unsigned char MesBat; // Mesure de la tension batterie

// Ici sont toutes les fonctions utilisée dans le fichier FonctionPrincipales.c

extern char trad(unsigned char Mes);

// Fonction d'initialisation :

void Initialisation(void);

// Positionnement du robot pour qu'il se mette en face de la plaque par modulation du rapport cyclique

void Positionnement(void);

// Scan avec les IR de la distance entre le robot et la plaque lors du Positionnement, remplit les tableaux scanL et scanR en mémoire

void ScanP(void);

// Scan avec les IR de la distance entre le robot et la plaque lors de la Marche, remplit les tableaux scanL et scanR

void ScanAs(void);

// Commande le PWM pour les moteurs lors de la marche du moteur

void Marche1(void);

2) headSousFonctions.h :

// Ici sont toutes les fonctions présentes dans le fichier SousFonctions.c

// Arret du robot suite à une erreur de déplacement

void errordir1();

// Sort la tenstion de la batterie

void OutBat();

// Transforme la valeure retrouvé par les IR en une valeur exploitable via le tableau de conversion

char trad(unsigned char Mes);

// Fait une nouvelle moyenne dans le tableau avec la nouvelle valeur

unsigned char moyenne(unsigned char Val, unsigned char\* Tab);

3) FonctionPrincipales.c :

const unsigned char ComPWM[20]; // = {13, 14, 14, 15, 17, 18, 20, 22, 24, 26, 26, 26, 26, 26, 26, 23, 21, 18, 15, 13};

// Fonction d'initialisation :

void Initialisation(){ //Description de la fonction Initialisation, qui ne revoit rien.

/\*Cette fonction initialise :

\* le convertisseur chargé de traité la tension issue de la battérie,

\* le timer0,

\* l'I²C,

\* les capteurs infrarouges,

\* les signaux PWM,

\* les 8 LEDs,

\* L'RS232

\* \*/

OSCCONbits.IRCF0 = 1 ; // On utilise l'oscillateur interne du microcontroleur

OSCCONbits.IRCF1 = 1 ; // Qui est de 8MHz

OSCCONbits.IRCF2 = 1 ;

OSCCONbits.SCS1 = 1 ;

/\*Inilisation du Conversisseur Analogique Numérique (CAN)\*/

//Configuration entrée/sortie de la broche 2 du port A

TRISAbits.RA2 = 1 ; // Broche 2 du port A en entrée

TRISBbits.RB5 = 0 ; //Broche de la LED test

TRISBbits.RB0 = 1 ; //Broche 0 du port B(INT0) en entrée

/\*Initialisation des capteurs infrarouges\*/

TRISAbits.RA0 = 1 ; // Broche 0 du port A en entrée (tension de sortie du capteur droit)

TRISAbits.RA1 = 1 ; // Broche 1 du port A en entrée (tension de sortie du capteur gauche)

TRISBbits.RB1 = 0 ; // Broche 1 du port B en sortie

// Broche configuré en numérique plus haut

/\*Initialisation des signaux PWM\*/

TRISCbits.RC1 = 0 ; //Broche 1 du port C en sortie (PWMD)

TRISCbits.RC2 = 0 ; //Broche 1 du port C en sortie (PWMG)

// Configuration analogique/numérique de la broche 2 du port A

ADCON1bits.PCFG0 = 0 ;

ADCON1bits.PCFG1 = 0 ; // Indique que le broche 2 du port A

ADCON1bits.PCFG2 = 1 ; // est une grandeure analogique

ADCON1bits.PCFG3 = 1 ; // Sélection de AN0 à AN2, les autres broches sont en numériques

//Sélection des tensions de référence

ADCON1bits.VCFG0 = 0 ; // On n'utilise pas de tension de référence

ADCON1bits.VCFG1 = 0 ; //(ni Vref+, ni Vref-)

/\*

//Configuration de la fréquence de fonctionnement du CAN et du temps de conversion

ADCON2bits.ADCS0 = 0 ;

ADCON2bits.ADCS1 = 0 ; // Il faut que Tad (temps de conversion) soit le plus faible possible met supérieur à 0,7 µs

ADCON2bits.ADCS2 = 1 ; // Tad = 1 /(Fosc/Préscaler) Fosc = 8MHz, donc il faut un préscaler de 8. Tad = 1µs > Tadmin = 0,7 µs

//Configuration du temps d'acquisition Tacq = Tamp + Tc + Tcoff

ADCON2bits.ACQT0 = 0 ; // Tamp = 2µs, c'est la temps de mise en place, Tc = 1,05µs(dans le pire des cas), c'est le temps de chargement

ADCON2bits.ACQT1 = 0 ; // Tcoff = 1,2µs(dans le pire des cas), c'est temps supplémentaire dépendant de la température

ADCON2bits.ACQT2 = 1 ; // Tacq = 2 + 1 + 1,2 > 4,2 µs, ici Tacq = 8.Tad = 8 µs > 4,2µs\*/

ADCON2 = 0x1E;

// Résultat de la conversion droite justifié ou gauche justifié

ADCON2bits.ADFM = 0 ; //Gauche justifié (on prend les 8 bits de poids forts,le plus significatifs)

ADCON0bits.ADON = 1 ; // Active le module de version analogique/numérique

/\*Initialisation du Timer0\*/

//Ce timer sera utiliser pour l'intérruption

T0CONbits.T08BIT = 1 ; // Configuration du timer0 en 8 bits

T0CONbits.T0CS = 0 ; // Utilisation de l'horloge interne pour le timer0

T0CONbits.PSA = 1 ; // Utilisation d'un préscaler

T0CONbits.T0PS2 = 1 ; // Valeur du prescaler :

T0CONbits.T0PS1 = 0 ; // On veut avoir la moyenne de la tension de la battérie toute les 250ms (on fait 4 mesures)

T0CONbits.T0PS0 = 1 ;

T0CONbits.TMR0ON = 1 ; // Activation de timer0

// Prescaler = 64 => Ftimer0 = Fosc.Préscaler/4 => t1mesure = N/Ftimer0 = 32.8ms

// => tmesuremoy = 4.t1mesure = 131ms

/\*Initialisation du Timer2\*/

//Ce timer sera utiliser pour générer les signaux PWM

/\*T2CONbits.T2OUTPS3 = ;

T2CONbits.T2OUTPS2 = ;

T2CONbits.T2OUTPS1 = ;

T2CONbits.T2OUTPS0 = ;\*/

T2CONbits.T2CKPS1 = 1 ; //Choix d'un prescaler de 16

//T2CONbits.T2CKPS0 = 1; // Inutile si T2CONbits.T2CKPS1 = 1

PR2 = 124 ; //On veut une fréquence Fpwm de 1 kHz, PR2 = [Fosc/(4.Fpwm.16)]-1 = 8M/(4.1k.16)-1 = 124 , PR2 <255

//pour générer les signaux PWM mode CCPi

CCP2CONbits.CCP2M3 = 1 ; // Mode CCP2 pour la broche 12

CCP2CONbits.CCP2M2 = 1 ;

CCP1CONbits.CCP1M3 = 1 ; // Mode CCP1 pour la broche 13

CCP1CONbits.CCP1M2 = 1 ;

CCP1CONbits.CCP1M1 = 0 ; // Broche 12(PWMD) et 13(PWMG)en simple sortie.

CCP1CONbits.CCP1M0 = 0 ;

T2CONbits.TMR2ON = 1 ; //Mettre en route le timmer 2

/\*Initialisation de l'RS232\*/

TXSTAbits.SYNC = 0 ;// Liaison asynchrone

BAUDCONbits.BRG16 = 0 ; // Transmission du 8bits

TXSTAbits.BRGH = 0 ;

SPBRG = 12 ;// BaudRate = Fosc/[64](SPBRG+1]

// SPRBG = Fosc/[64.BaudRate] -1 = 8M/[64.9,6k] -1 = 12

RCSTAbits.FERR = 0 ; // Pas d'erreur de cadrage

RCSTAbits.OERR = 0 ; // Pas d'erreur de dépassement

RCSTAbits.CREN = 1 ; // Activer la réception

TRISCbits.RC6 = 1 ; //Broche 6 du port C TX en entrée

TRISCbits.RC7 = 1 ; //Broche 7 du port C RX en sortie

RCSTAbits.SPEN = 1 ; //Validation de la broche TX et RX

TXSTAbits.TXEN = 1 ; //Validation de la transmission

SSPCON2bits.RCEN = 1 ; //Validation de la translission

PIR1bits.TXIF = 1 ; //Effacement de l'intérruption TX

PIE1bits.TXIE = 1 ; //Valider l'intérrruption TX

PIR1bits.RCIF = 1 ; //Effacement de l'intérruption RX

PIE1bits.RCIE = 1 ; //Valider l'intérrruption RX

/\*Initialisation de l'I²C\*/

TRISCbits.RC3 = 1 ; //Broche 3 du port C en entrée (SCL)

TRISCbits.RC4 = 1 ; //Broche 4 du port C en entrée (SDA)

SSPCON1bits.SSPEN = 1 ; // Broche 14(SCL) et (SDA) en mode I²C

SSPCON1bits.SSPM3 = 1 ;

SSPCON1bits.SSPM2 = 0 ; // Utilisation de mode master

SSPCON1bits.SSPM1 = 0 ;

SSPCON1bits.SSPM0 = 0 ;

SSPSTATbits.SMP = 1 ; //Echange standart sur front montant

SSPADD = 39 ; // Pour une fréquence 50kHz pour l'I2C :

//SSPADD=Fosc/(4.CLK)-1=(8.?10?^6)/(4.50.?10?^3 )-1=39

/\*Initialisation des intérruptions\*/

INTCONbits.TMR0IE = 1 ; // Autoriser le débordément du timer0

INTCONbits.INT0IE = 1; // Activation de INT0

INTCON2bits.INTEDG0 = 1 ; // Inérruption0 active sur front montant

INTCONbits.PEIE = 1 ; // Valider les périphériques d'intérruptions

INTCONbits.GIE = 1 ; // Autoriser les intérrruptions

}

// Positionnement du robot pour qu'il se mette en face de la plaque par modulation du rapport cyclique

void Positionnement() {

if (distR <= distL){

ApwmL = ComPWM[0];// IncrÃ©mentation ApwmL

ApwmR = 0; // DÃ©crÃ©mentation ApwmR

}

else {

ApwmR = ComPWM[0];// IncrÃ©mentation ApwmR

ApwmL = 0;// DÃ©crÃ©mentation ApwmL

}

}

// Scan avec les IR de la distance entre le robot et la plaque lors du Positionnement, remplit les tableaux scanL et scanR en mÃ©moire

void ScanP(){

unsigned char R, L; // Variables pour la mesure gauche et droite

// Aquisition L & R via les capteurs

// Mesurer sur le Can de la tension Batterie AN2

PORTBbits.RB1 = 0; // Capteur on

ADCON0bits.CHS0 = 0 ; // Sélection broche en analogique

ADCON0bits.CHS1 = 1 ; // Broche 2 du port A (AN2)

ADCON0bits.CHS2 = 0 ;

ADCON0bits.CHS3 = 0 ;

ADCON0bits.GO = 1;

while(ADCON0bits.GO);// Busy Flag

R = ADRESH ; //read Can

R = moyenne(R, ScanR); // Fait une nouvelle moyenne dans le tableau avec la nouvelle valeur

distR = trad(R); // Correspondance avec le tableau

// Mesurer sur le Can de la tension Batterie AN2

ADCON0bits.CHS0 = 1 ; // Sélection broche en analogique

ADCON0bits.CHS1 = 0 ; // Broche 2 du port A (AN1)

ADCON0bits.CHS2 = 0 ;

ADCON0bits.CHS3 = 0 ;

ADCON0bits.GO = 1;

while(ADCON0bits.GO);// Busy Flag

L = ADRESH ; //read Can

L = moyenne(L, ScanL); // Fait une nouvelle moyenne dans le tableau avec la nouvelle valeur

distL = trad(L); // Correspondance avec le tableau

PORTBbits.RB1 = 1; // Capteur off

}

// Scan avec les IR de la distance entre le robot et la plaque lors de la Marche, remplit les tableaux scanL et scanR

void ScanAs(){

unsigned char MesR; // Variables pour la mesure droite

unsigned char MesL; // Variables pour la mesure gauche

// Aquisition de L & R via les capteurs

PORTBbits.RB1 = 0; // Capteur on

ADCON0bits.CHS0 = 0 ; // Sélection broche en analogique

ADCON0bits.CHS1 = 1 ; // Broche 2 du port A (AN2)

ADCON0bits.CHS2 = 0 ;

ADCON0bits.CHS3 = 0 ;

ADCON0bits.GO = 1;

while(ADCON0bits.GO);// Busy Flag

MesR = ADRESH ; //read Can

ADCON0bits.CHS0 = 1 ; // Sélection broche en analogique

ADCON0bits.CHS1 = 0 ; // Broche 2 du port A (AN1)

ADCON0bits.CHS2 = 0 ;

ADCON0bits.CHS3 = 0 ;

ADCON0bits.GO = 1; // Capteur off

while(ADCON0bits.GO);// Busy Flag

MesL = ADRESH ; //read Can

// Transforme la valeure retrouvÃ© par les IR en une valeur exploitable via le tableau de conversion

MesR = moyenne(MesR,ScanR); // Fait une nouvelle moyenne dans le tableau avec la nouvelle valeur

distR = trad(MesR) ;

MesL = moyenne(MesL, ScanL); // Fait une nouvelle moyenne dans le tableau avec la nouvelle valeur

distL = trad(MesL) ;

// Test en cas d'erreur sur une déviation de chemin

if ( distR != distL )

{

errordir1();

}

}

// Commande le PWM pour les moteurs lors de la marche du moteur

void Marche1(){

// Regarder dans la table 2 ( conrrespondance distance / vitesse ) via distL et distR

// Set ApwmL & ApwmR

ApwmL = ComPWM[distL];

ApwmR = ComPWM[distR];

}

4) SousFonctions.c :

unsigned char dist[20] = { 11, 11, 12, 14, 15, 16, 18, 20, 22, 24, 27, 31, 35, 40, 46, 55, 66, 83 ,110, 135};

//const rom unsigned char ComPWM[20] = {51, 54, 56, 61, 66, 71, 79, 87, 94, 102, 102, 102, 102, 102, 102, 92, 82, 71, 61, 51};

// Arret du robot suite à une erreur de déplacement

void errordir1(){

// Afficher Erreur Marche Led

// Terminaison du programme car erreur:

while(1);

}

// Sort la tenstion de la batterie

void OutBat(){

// Afficher Erreur Batterie Led

// Entrer en Mode Sleep

}

// Transforme la valeure retrouvé par les IR en une valeur exploitable via le tableau de conversion

char trad(unsigned char Mes){

char Trans = 0; // Variable de parcours du tableau qui est retournée

// Comparaison de la valeur mesurée avec un tableau de conversion dans un header

while (Mes <= dist[Trans]){

Trans++;

}

return Trans; // retourner la distance

}

// Fait une nouvelle moyenne dans le tableau avec la nouvelle valeur

unsigned char moyenne(unsigned char Val, unsigned char\* Tab){

// Boucle de parcour du tableau :

char i ; // Variable de parcours du tableau

unsigned char Moy = Val;

for (i = 0 ; i <= 2 ; i++ ){

Tab[i+1] = Tab[i]; // Passer la valeur i du tableau dans la valeur i + 1

Moy = Tab[i] + Moy; // Somme pour la moyenne

}

Tab[0] = Val; // Donner à la valeur 0 du tableau la valeur de la mesure

Moy = Moy>>2; // Division par 4 en décalant à droite 2 fois

return Moy; // Retourne la moyenne du tableau

}

5) main.c :

#include <p18f2520.h>

#include "headall.h"

#include "headSousFonctions.h"

#include "FonctionPrincipales.c"

#include "SousFonctions.c"

#pragma config OSC = INTIO67

#pragma config PBADEN = OFF, WDT = OFF, LVP = OFF, DEBUG = ON

#pragma code HighVector=0x08

void IntHighVector(void){

// Intéruptions

// Pour la Batterie Appel toutes les 100 ms

if( INTCONbits.TMR0IF ) {

INTCONbits.TMR0IF = 0; //Flag Off

PORTBbits.RB5 = 1; // signale que l’interruption a eu lieu

// Mesurer sur le Can de la tension Batterie AN2

ADCON0bits.CHS0 = 0 ; // Sélection broche en analogique

ADCON0bits.CHS1 = 1 ; // Broche 2 du port A (AN2) en analogique

ADCON0bits.CHS2 = 0 ;

ADCON0bits.CHS3 = 0 ;

ADCON0bits.GO = 1;

while(ADCON0bits.GO);// Busy Flag

MesBat = ADRESH ; //read Can de batterie

// Interruption télécomande

}

}

void main (void)

{

Initialisation(); // Appel de la fonction d'initialisation

PORTBbits.RB5 = 1 ;

// Initialisation des Interruptions

// Affichage des Leds -> Pret

while (1) {

// Sleep mode // Attente de commande par la télécommande

// Déclaration des Variables

//Positionnement & Détection de l'obstacle

distL = 0;

distR = 0;

while (distL == 0 && distR == 0 )

{ // test sur la distance, à plus de 150 cm +/- 7 cm

ScanP(); // détection de la plaque à distance

Positionnement(); // Mise en Position du robot

CCPR1L = ApwmL ;// Commande moteur Gauche

CCPR2L = ApwmR ;// Commande moteur Droit

}

while (distL <= 18 && distR <= 18)

{ // Test sur la distance ici avec le model : à 24 cm

if(PORTBbits.RB1 == 0 )

{ // Cadencement sur l’interruption

PORTBbits.RB1 = 1;

moyenne(MesBat, Batterie); // Faire la moyenne dans le tableau

// test de valeur Bon :

if (Vbat <= 159 ){ // valeur correspondant sur la simulation ISIS

OutBat(); // batterie insuffisante

}

}

ScanAs(); // Détection de la distance

Marche1(); // Controle des PWM pour la vitesse de déplacement

CCPR1L = ApwmL ;// Commande moteur Gauche

CCPR2L = ApwmR ;// Commande moteur Droit

}

// Fin -> Economie d'énergie

while(1); // terminaison du programme pour le debug

}

}

**VI- Les tests et le Debug :**

Pour ce qui est de la partie test, nous avons de testé nos fonctions au cours du processus de développement sur les modèles numériques de simulation Proteus afin de s’assurer du bon fonctionnement de notre commande et de procéder aux réglages des constantes pour les timers de scan. Pour cela, nous construirons les modèles avec les signaux correspondants aux tests des fonctions spécifiques.

La phase de debug avec la liaison RS232 sera cadencée sur les temps de bouclage des différents instants du système. Ces temps ne sont encore pas connus à l’heure où nous écrivons ce texte et seront des paramètres importants de nos premiers tests sur simulateur et sur le robot.

Lors des premières simulations ISIS, pour tester la fonction initialisation, nous avons constaté un problème avec l’I²C, nous voulions allumer les 8 LEDs mais le microcontrôleur ne recevait pas le premier acknowledge, le bit d’acquittement qui indique que le récepteur est prêt à recevoir. En utilisant le débugger I²C, on remarque que seul le bit start et l’adresse du récepteur du récepteur sont transmis.

Nous avons eu aussi un problème pour générer les signaux PWM, car le timer2 qui est utilisé spécialement pour les signaux PWM, n’était pas activé il manquait le flag suivant T2CONbits.TMR2ON = 1 ;

Lors de la création du projet, nous avons fait une erreur sur le choix du microcontrôleur, nous avons en effet sélectionné le PIC18F2550, de ce fait le compilateur indiquait une erreur sur la ligne suivante

#pragma config OSC = INTIO67, car « OSC » n’était pas dans le PIC18F2550.

**VII- Conclusion :**

Malheureusement beaucoup d'objectif n'ont pas été atteints, alors que nous étions partis sur de bonnes bases. En effet nous avions bien réussi le projet robot 1 avec le soudage de la carte et sa compréhension, la partie "hard".

De plus nous avons passé beaucoup de temps sur les organigrammes que nous avons pris soins de faire et sur des aspects théoriques qui se sont révélés inutiles car nous n’avons pas réussi à atteindre ses problématiques.

Cependant, ce projet était l’occasion de mettre en place un code C important et de l’implémenter sur une carte.