SAM 2 - System Design

Rapport de Conception

Professeurs encadrants :
M. Acacio Marques
M. Jean-Yves Rebouché

Projet Robot 2

Théo Gachet & Cyril Hérail

« Soyez réalistes, exigez l'impossible »
Ernesto "Che" Guevara

Table des matières

1 Introduction

1.1 Cahier des charges : contrat 7

L'objectif du contrat 7 est de réaliser un robot suiveur : le robot détecte un objet, une personne dans une certaine plage de distance (entre 40 et 150 cm), et avance. Ainsi si l'objet détecté avance, le robot peut le suivre à condition que leurs vitesses soient à peu près égales.

D'autres spécifications sont renseignées. La fréquence de l'oscillateur du microcontrôleur doit être réglée à 8 MHz. La tension de la batterie doit être moyennée grâce à quatre mesures et affichée dans l'UART, réglé sur 9600 bauds. Également, si la tension de la batterie devient inférieure à 10V, la LED Test doit s'allumer pour prévenir l'utilisateur, mais les moteurs continuent cependant à tourner.

Les PWM moteurs doivent avoir un rapport cyclique inférieur à 50%. La réception de la télécommande s'effectue à 50 kHz, seul le bouton central est pris en compte. L'appui sur le bouton central provoque une interruption au niveau software et la mise en marche ou l'arrêt du robot, selon son état actuel.

1.2 Répartition du travail

Un projet d'une telle envergure nécessite un vision claire des objectifs à moyen terme. Dès le début, nous avons donc segmenté le travail à effectuer en fonctions de nos compétences individuelles :

Théo :

- Réalisation des algorigrammes
- Rédaction du code en C
- Simulations logicielles sur Protheus

Cyril :

- Debug hardware
- Calculs (timers, ADC, etc.)
- Simulations sur oscilloscope

2 Algorigrammes

2.1 Main

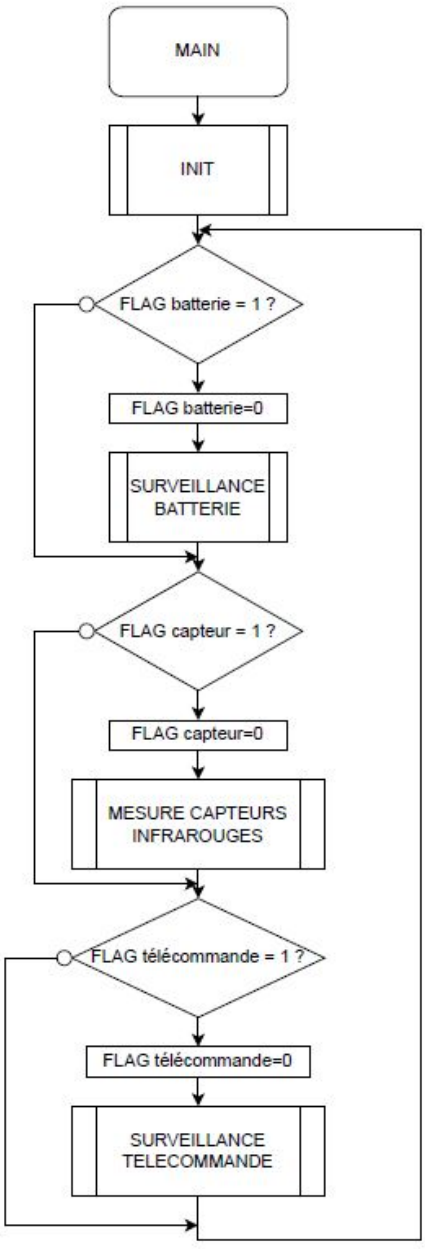

FIGURE 1 – Programme principal

2.2 Interruptions et initialisations

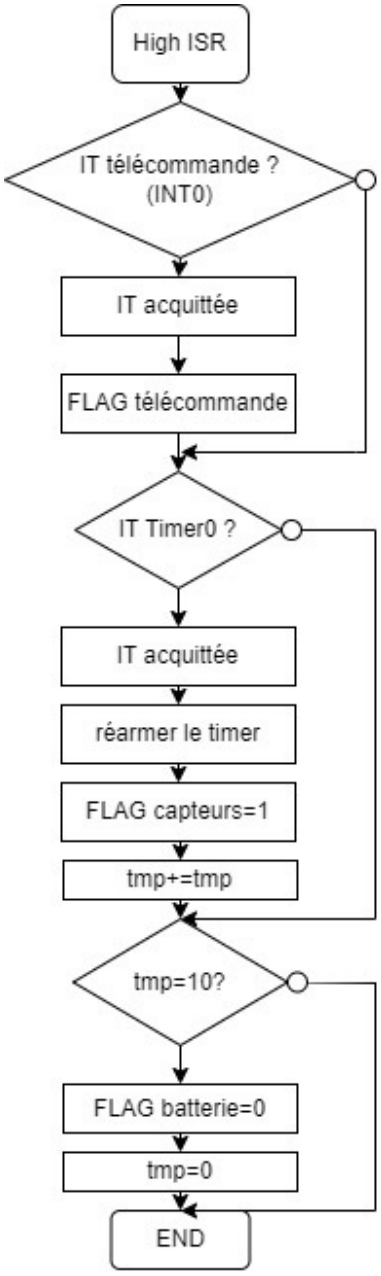

FIGURE 2 – Routine d'IT haute priorité

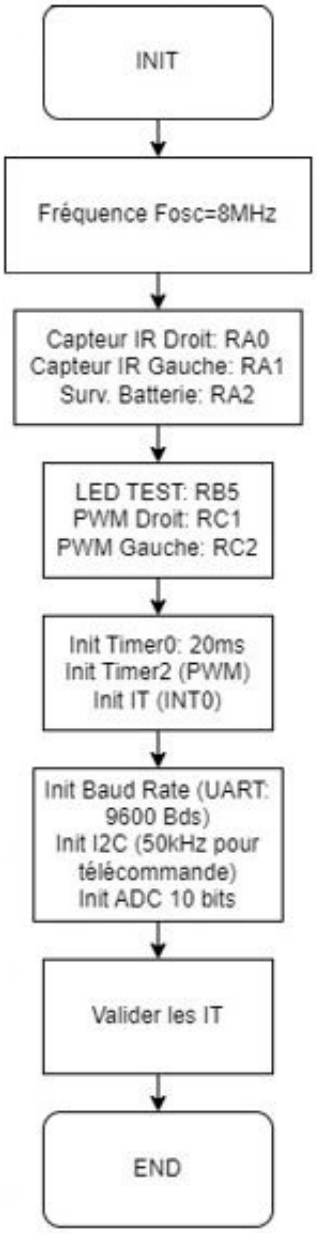

FIGURE 3 – Fonction d'initialisation

2.3 Surveillance batterie

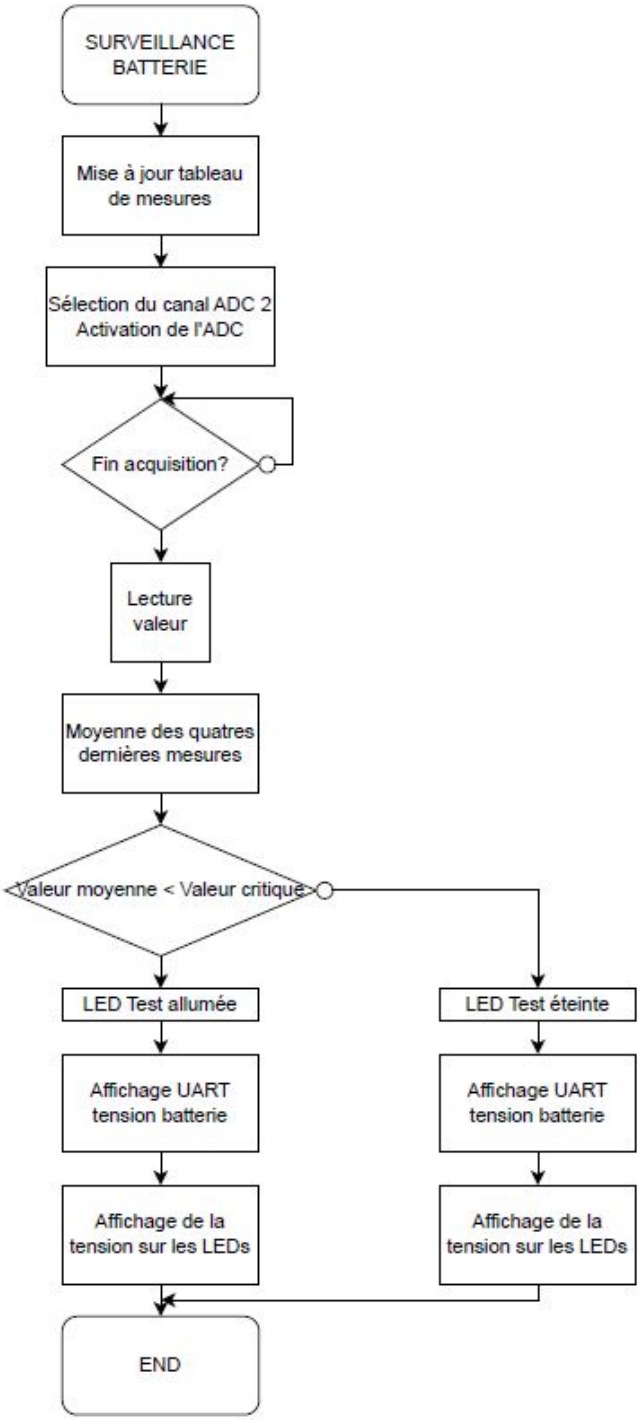

FIGURE 4 – Routine de surveillance batterie

2.4 Surveillance télécommande

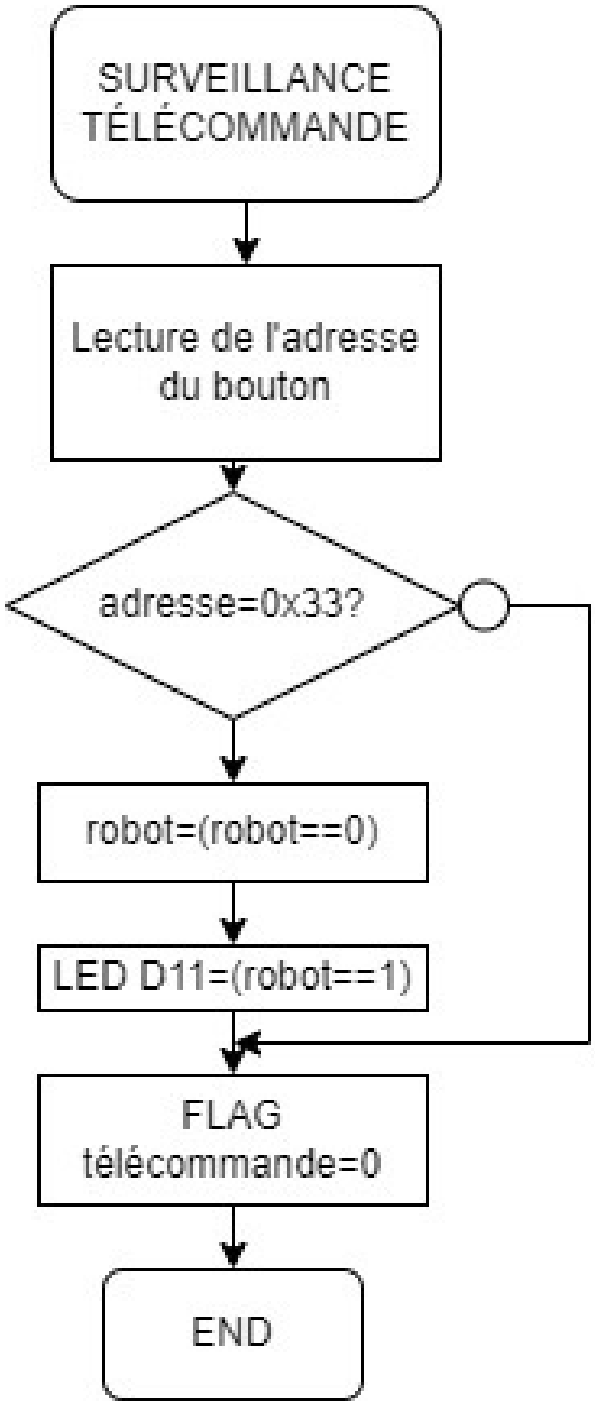

FIGURE 5 – Routine de surveillance télécommande

2.5 Mesures capteurs IR

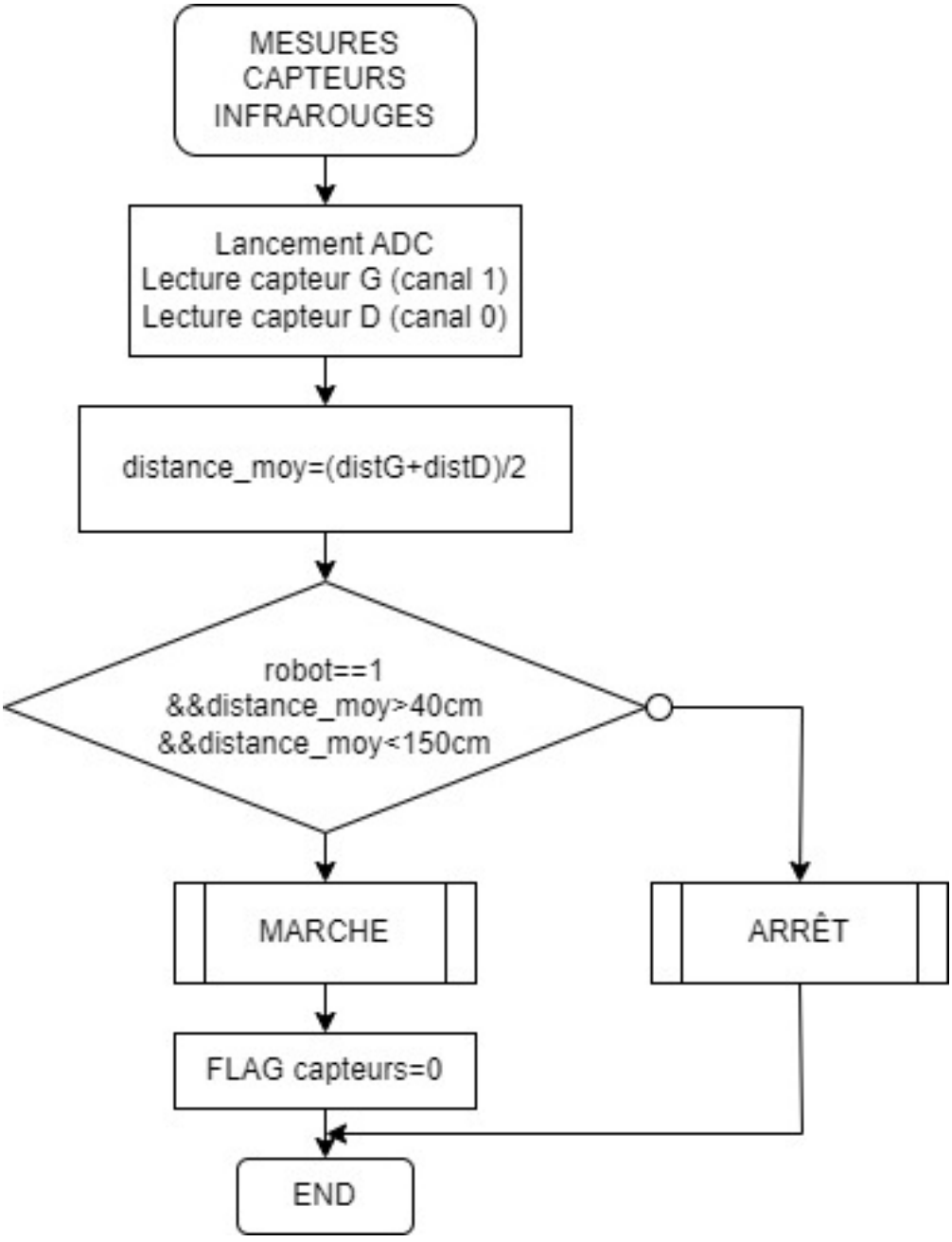

FIGURE 6 – Routine de mesure par les capteurs infrarouges

3 Détail des calculs

3.1 Période du CAN

On configure le CAN tel que :

$$ADRES = (2^{10} - 1) \times \frac{Van2 - Vss}{Vdd - Vss}$$

On choisit T_{AD} tel que :

$$T_{AD} = \frac{4}{F_{OSC}} = \frac{4}{8MHz} = 0.5\mu s$$

On sait d'après la datasheet que

$$T_{ACQ} = T_{AMP} + Tc + T_{COFF}$$

On a toujours $T_{AMP} = 2\mu s$ car on n'ajoute pas de conditionneur externe. D'après la datasheet, on a :

$$Tc = -(CHOLD)(Ric + Rss + Rs)$$

avec $CHOLD = 25pF, Ric = 1k\Omega, Rss = 2k\Omega$ et d'après le schéma $Rs = 2.2k\Omega$.

On trouve $Tc = 0.991\mu s$ soit une microseconde.

On a $T_{COFF} = 0s$ à 25řC et dans le pire des cas (soit 85řC) $T_{COFF} = 1.2\mu s$.

On a donc $T_{ACQ} = 4.2\mu s$ dans le pire des cas.

On en déduit ACQT2 :ACQT0,

$$ACQT2 : ACQT0 = \frac{T_{ACQ}}{T_{AD}} = 8$$

D'où :

ACQT2=1
ACQT1=0
ACQT0=0

3.2 Surveillance batterie

On stocke la valeur numérique issue de la conversion de l'ADC et on fait une moyenne avec les trois dernières mesures. On a une valeur comprise entre 0 et 255. On affiche cette valeur numérique (appelé valeur_batterie) dans l'UART. On affiche aussi le pourcentage de batterie restant, le calcul pour obtenir ce pourcentage :

$$pourcentage = \frac{100 \times valeur\_batterie}{255}$$

3.3 Génération PWM

Le Timer 2 va servir à générer le signal PWM des moteurs. On a un signal PWM de fréquence f=4000Hz.

Dans un premier temps, on calcule la valeur de PR2 :

$$PR2 = \frac{F_{OSC}}{4 \times FPWM \times prescaler} - 1$$

Nous voulons que PR2 soit le plus grand possible mais également inférieur à 255. Il vient :

$$\frac{F_{OSC}}{4 \times FPWM \times prescaler} - 1 \leq 255$$

Or le prescaler ne peut prendre que trois valeurs possibles : 1, 4 ou 16. Pour que la condition soit vérifiée, il faut que le prescaler soit égal à 4 ou 16. On choisit 4 car cela nous donne la plus grande valeur de PR2. Avec ce choix de prescaler, on obtient donc :

$$PR2 = 124$$

Il reste à calculer les valeurs de CCPRxL, DCxB0 et DCxB1. Ici x vaut 0 ou 1 et permet de différencier le moteur droit du moteur gauche. Comme le robot ne fait qu'avancer en ligne droite sans tourner pas, ces valeurs sont les mêmes pour les deux moteurs.

Lorsque le robot est en marche on définit le rapport cyclique RC à 50% comme suit :

$$CCPRxL : DCxB1 : DCxB0 = \frac{RC \times F_{OSC}}{100 \times prescaler \times FPWM}$$

On a $CCPRxL : DCxB1 : DCxB0 = 250 = 0b11111010$

D'où $CCPRxL = 250 >> 2 = 62 = 0b00111110$, DCxB1=1 et DCxB0=0.

À l'arrêt, le rapport cyclique est nul, d'où CCPRxL = 0, DCxB0 = 0 et DCxB1 = 0.

3.4 Timer0

L'oscillateur est réglé à 8MHz, on a une horloge de $Fclock = \frac{F_{OSC}}{4} = 2MHz$ en entrée du Timer0. On veut une base de temps de 20ms, la formule pour calculer la période est :

$$Periode = (OV - rechargement) \times prescaler \times T_{horloge}$$

On commence par sélectionner le mode du Timer0 (8 bits ou 16 bits), ce choix nous donnera la valeur de l'overflow. On choisit ici le mode 16 bits du Timer0, d'où :

$$Overflow = 2^{16} - 1 = 65535$$

On calcule ensuite le prescaler. Pour cela, on prend un rechargement nul et on obtient un prescaler de 4. Enfin, on calcule la valeur de rechargement :

$$rechargement = Overflow - \frac{T_{horloge} \times periode}{prescaler} = 55535 = 0xD8EF$$

On retrouve ainsi dans le code : TMROH = 0xD8 et TMR0L = 0xEF.

4 Simulations

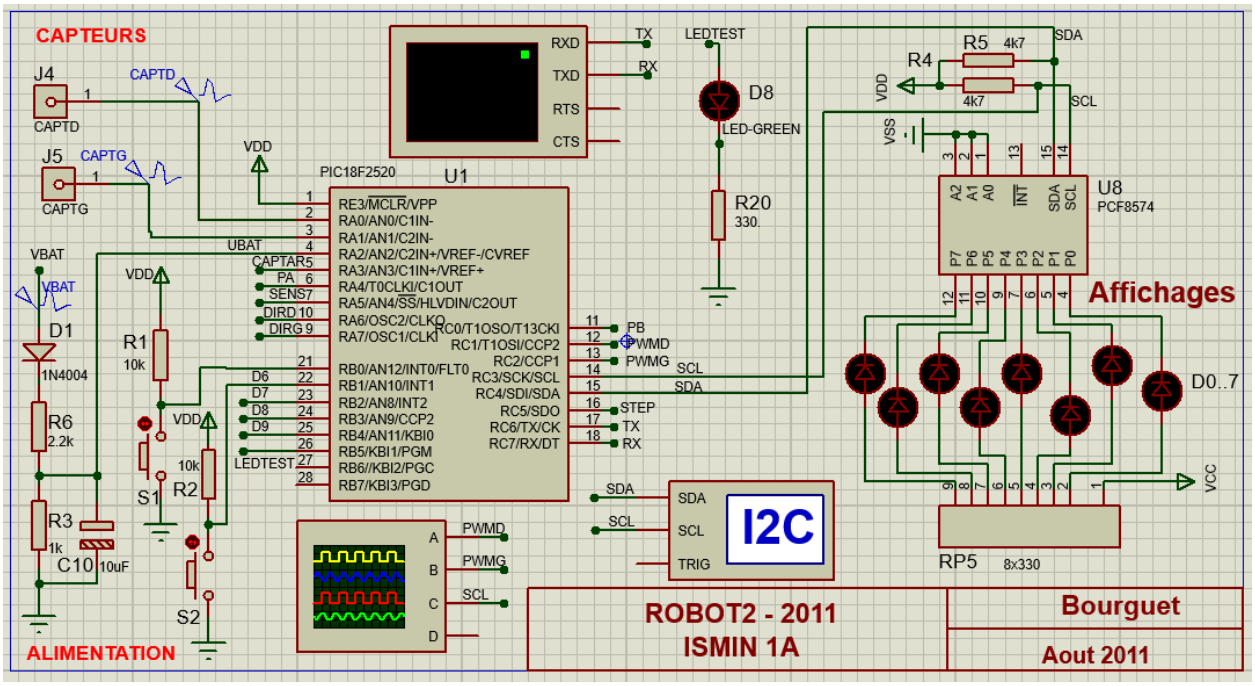

FIGURE 7 – schema électrique - Simulation Protheus

4.1 Période de Timer0

La simulation du timer0 semble correctement fonctionner sous Proteus. Cependant nous avons voulu vérifier que le timer0 respecte scrupuleusement le cahier des charges. En effet, au delà de simplement voir que le timer0 est activé, réalise des interruptions et semble rapide, il est toujours possible qu'une erreur nous échappe. Par exemple, nous voulions une base de temps de 20ms mais en l'absence de vérification poussée, peut-être était-il à 19ms ou 21ms, au quel cas cela est indiscernable au premier abord dans Proteus. Il a fallu légèrement modifier le code. Dans la simulation sous Proteus, on a relié l'oscilloscope à la tension de la LED Test. Cette dernière commute à chaque interruption du timer0. C'est ainsi que nous avons pu mesurer la base de temps du timer0 et vérifier qu'il respectait le cahier des charges

4.2 Génération PWM

Grâce à l'oscilloscope dans Proteus, on a pu vérifier que le signal PWM avait la bonne fréquence, le bon rapport cyclique (0 ou 50%) et était identique pour les deux moteurs. On a utilisé notamment la fonction vitesse_moteur().

4.3 Surveillance batterie

Pour la surveillance batterie, on utilise le virtual terminal sous Proteus pour simuler la liaison UART. On affiche la valeur numérique et le pourcentage de tension correspondant. Ainsi on simule la tension de la batterie grâce à une source de tension constante. On vérifie que la valeur numérique correspond bien à l'image de la tension d'entrée. On fait des tests avec différentes valeurs de tension. La vérification de la surveillance a nécessité de vérifier que la simulation de la liaison UART marchait pour pouvoir afficher les valeurs. Bien sûr d'autre solutions était possible, par exemple en mettant des points d'arrrêts, mais celle-ci était plus rapide une fois la liaision UART programmée. Également, nous avons aussi vérifier que l'affichage des LEDs fonctionnait correctement. Nous avons simplement regarder que pour une tension d'entrée on avait le bon affichage LED correspondait comme défini dans la fonction affichage_batterie().

4.4 Réception télécommande

Dans la simulation Proteus, la télécommande est remplacée par un interrupteur, on ne peut pas vérifier l'adresse de la télécommande. Néanmoins on peut vérifier que l'interruption marche bien sur le patte correspondante du microcontrôleur. Dans le virtual terminal on affiche "IT Telecommande" à chaque fois que l'on détecte une interruption externe INTOIF et donc lorsque l'on appuie sur le bouton. Dans le code final, nous avons mis en commentaire la ligne de code qui permettait de réaliser cet affichage après avoir vérifié que cela fonctionnait en simulation.

4.5 Lecture capteurs IR

Dans la simulation, les capteurs sont remplacés par des sources de tensions constantes qui simule la sortie des capteurs. Pour vérifier que l'ADC fonctionnait, on a affiché les valeurs numériques après conversion des deux capteurs dans l'UART. On a fait plusieurs tests pour des valeurs de tensions différentes. Cela a aussi permi de vérifier que l'algorithme marchait bien i.e. que le robot avance uniquement quand il est dans une certaine plage de distance. On a vérifié cette fonction au début seule puis en interaction avec d'autre fonctions(vitesse_moteur(), surveillance_tele()...). Le code a ainsi évolué petit à petit. On a put aussi simuler certains corner cases comme le cas où un capteur détecterai un objet comme étant à une distance assez proche et l'autre à une distance éloigné (pour rappel notre algorithme prend la moyenne des deux valeurs renvoyées par le capteur).

5 Application

5.1 Explications de notre algorithme

Au début du programme, on initialise l'oscillateur à 8MHz, puis les timers, on configure les entrées-sorties, le signal PWM, l'I2C, l'UART et les interruptions. Avant d'entrer dans la boucle while, on valide les interruptions. Dans la boucle while, on teste de manière séquentiel les flags.

Le premier flag va déclencher un appel à la fonction surveillance_batterie(). Dans cette fonction on affiche grâce à la liaison UART la valeur numérique de la tension batterie et le pourcentage correspondant, l'image de cette tension est affichée sur les 8 LEDs . Les 8 LEDs servent d'interface homme-machine car le nombre de LEDs allumées correspond au pourcentage de la batterie. Dans le cas d'une batterie pleine, toute les LEDs sont allumées. Il est à noter que la valeur numérique provient d'une moyenne de 4 mesures. À chaque nouvelle mesure, on supprime de la mémoire la plus vielle mesure et on la remplace par la plus récente.

Le deuxième flag va déclencher un appel à la fonction surveillance_capteur(). On lance une conversion de l'ADC et on récupère une mesure de la tension de sortie des deux capteurs. On fait une moyenne de ces deux valeurs, puis si la variable robot est à 1 et que cette valeur est dans la bonne plage de distance, on lance les moteurs sinon on les arrêtes.

La variable robot est initialisé à 0, l'idée est que au début tant que l'on n'appuie pas sur la télécommande le robot n'avance pas même s'il est à bonne distance de l'objet. Chaque appuie sur la télécommande entraîne la commutation de cette variable (on alterne entre 0 ou 1).

Le troisième flag va déclencher un appel à la fonction surveillance_tele(). On change l'état de la variable robot.

De même, on utilise dans notre programme uniquement le vecteur de priorité hausse pour nos interruptions. Dans le vecteur de haute interruption , on a plusieurs interruptions. On a l'interruption externe INT0IF qui s'active à chaque appuie de télécommande, on acquitte l'interruption et on met alors le flag télécommande à 1. Ensuite, on a l'interruption du timer0, on acquitte l'interruption puis on met flag_capteurs_IR à 1. Toute les 10 fois, on met flag_batterie à 1. En effet, il est nécessaire de réaliser des acquisitions des mesures des capteurs le plus fréquemment possible d'où le fait qu'on le fasse systématiquement à chaque interruption du timer0. L'acquisition de la tension batterie est certes importante mais ne nécessite pas des temps si courts. Pour éviter de consommer de la charge CPU, on le fait toutes les 200ms (10x20ms).

5.2 Démonstration et résultats réels

5.2.1 I2C

À l'oscilloscope, on observe la liaison I2C qui permet l'affichage LED. Sur la sonde J24, on a SCL (voie 1) et sur la sonde J25 on a SDA (voie 2). En l'absence de réception d'un signal de la télécommande, le microcontrôleur envoie régulièrement une valeur sur U8. En effet, on fait appel à la fonction affichage_batterie après avoir effectuer une mesure de la tension batterie qui envoie une valeur correspondant à l'image de la tension mesurée. Cette valeur permet de régler l'affichage des 8 LEDs. On observe la trame suivante en binaire : 01000 0000 0 1100 0000. Les 8 premiers bits correspondent à l'adresse de U8 : 0x40. Le neuvième bit indique une écriture. Les 8 bits restants indique que le pourcentage de la tension batterie est entre 63% et 75%, ainsi les deux premières LEDs sont éteintes, les six autre sont allumées.

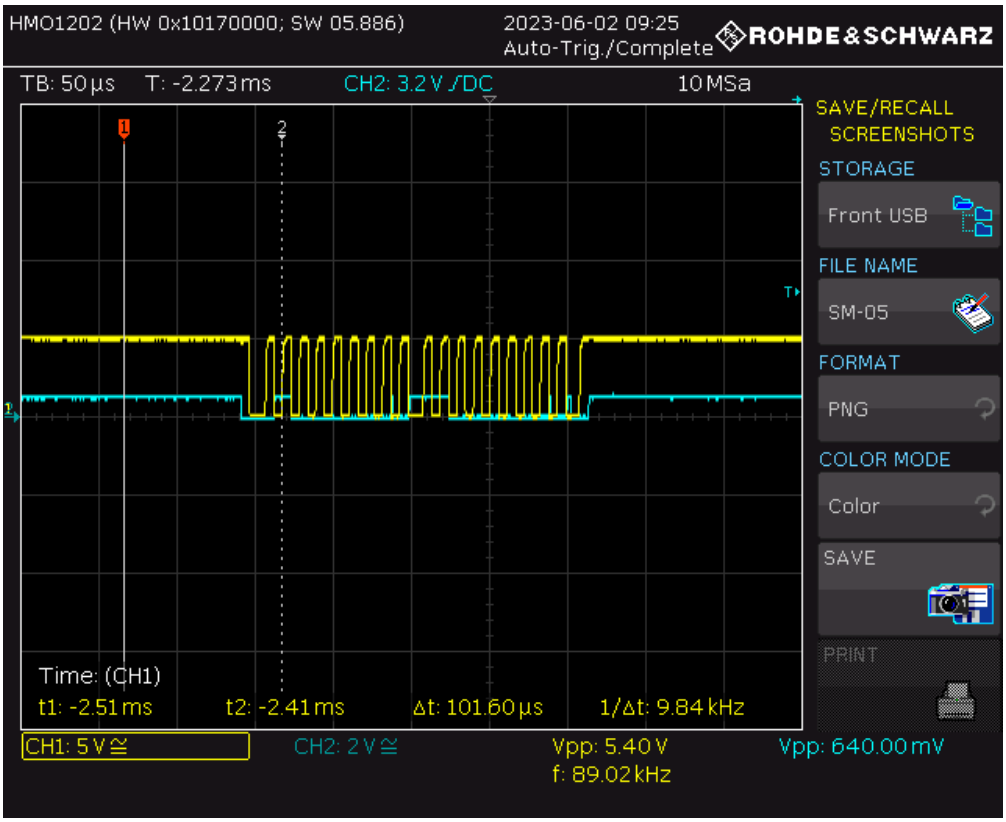

Figure 8 – signal I2C observé en J24 (SCL) et J25 (SDA)

5.2.2 Lecture capteurs IR

Au niveau hardware, on observe bien en sortie du capteur une tension analogique. Pour un objet placé à distance fixe du robot, cette tension est fixe. La datasheet du capteur donne une tension d'environ 1.5V pour un objet placé à 40 cm. On retrouve bien cette tension à l'oscilloscope. Au niveau software, on affiche grâce à la liaison UART la valeur numérique correspondante sur les deux capteurs. Comme les capteurs sont légèrement tournés, cette valeur numérique est proche mais pas totalement identique.

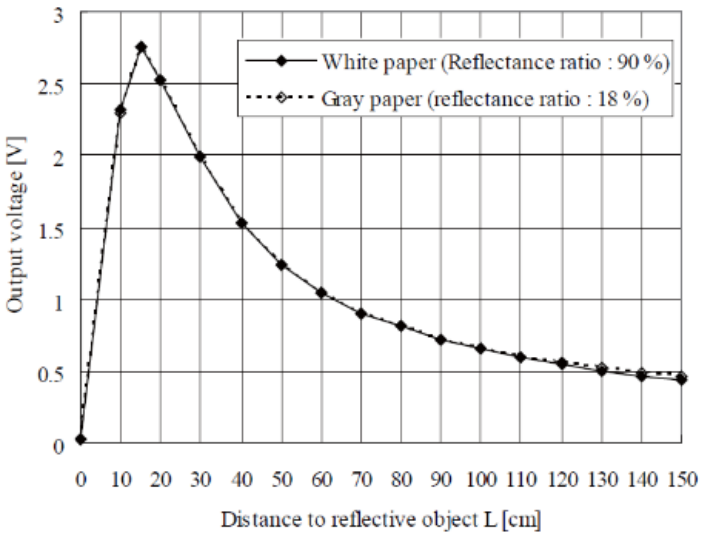

FIGURE 9 – extrait datasheet capteurs IR

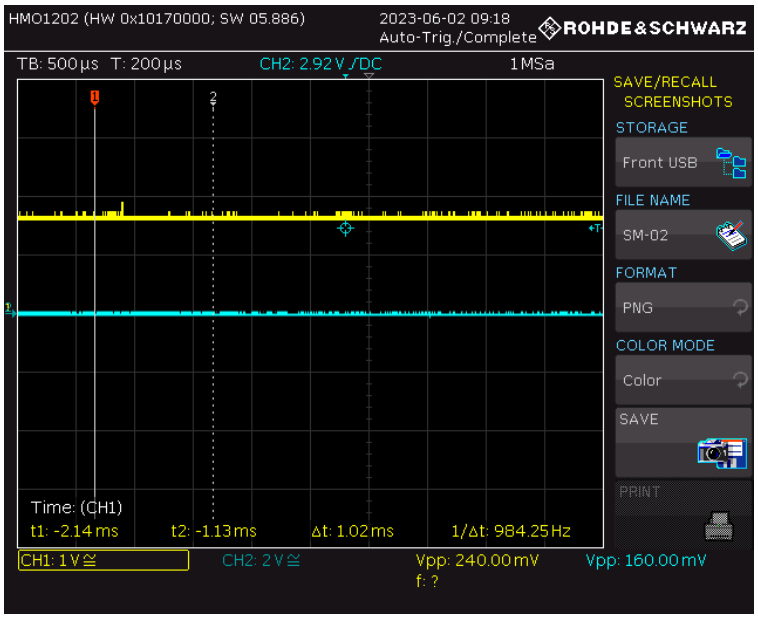

FIGURE 10 – tension en sortie du capteur droit pour un objet à 40 cm

5.2.3 Génération PWM

Dans le cas où les moteurs sont à l'arrêt, le signal PWM a un rapport cyclique RC=0%. On observe alors du bruit d'amplitude de quelques mV sur les sondes J17 et J18. Lorsque le robot est en marche, on observe à la fois un signal PWM du côté du moteur droit et du moteur gauche. Le signal PWM a dans les deux cas un rapport cyclique RC=50%. Le signal PWM a une fréquence de 4000Hz. On a ci-dessous une mesure du signal PWM à l'oscilloscope :

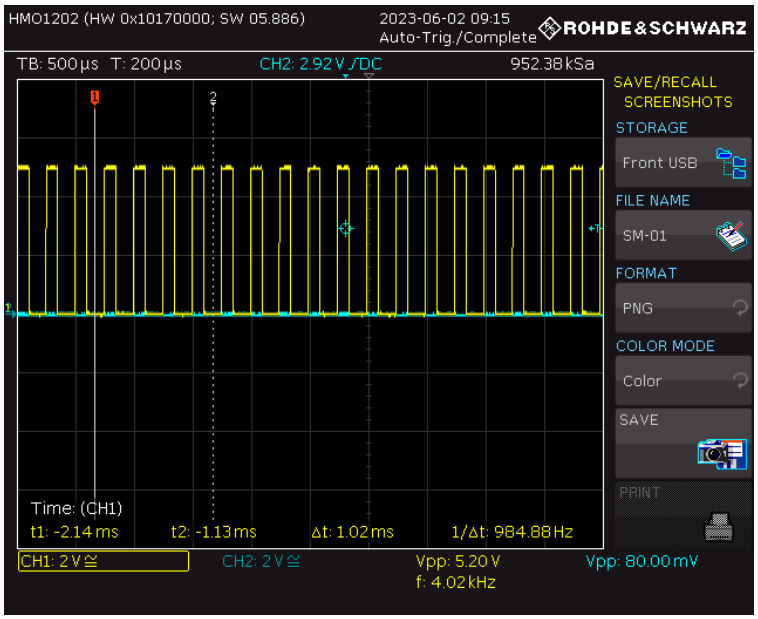

FIGURE 11 – le signal PWM observé sur la sonde J17 (moteur droit)

6 Conclusion

6.1 Différence par rapport au Cahier des Charges

Le cahier des charges est respecté. La seule différence notable qui persiste entre notre robot et les consignes du cahier des charges concerne le signal PWM. En effet, celui-ci a une fréquence de 4000Hz au lieu d'être à 1000Hz. Le cahier des charges précise en effet une base de temps de 1 ms.

6.2 Améliorations possibles

De nombreuses amélioration pourraient améliorer la précision et l'efficacité du système :

Éteindre les LEDs en cas de batterie faible : C'est une suggestion logique pour économiser de l'énergie. Si la batterie est faible, il est préférable d'économiser de l'énergie pour les composants essentiels du système. Les éteindre permettrait donc d'économiser de l'énergie.

Ajout d'un filtre anti-rebond pour la télécommande : Un "rebond" (bounce en anglais) est un phénomène courant dans les circuits électroniques où un signal peut fluctuer rapidement avant de se stabiliser à sa nouvelle valeur après un changement d'état. Ceci est souvent observable avec les boutons ou les interrupteurs. Dans le contexte de la télécommande, un filtre anti-rebond pourrait aider à s'assurer que les signaux de la télécommande sont correctement reçus et interprétés par le PIC18, en éliminant les fluctuations temporaires qui peuvent être interprétées comme des signaux multiples.

Utilisation d'un testeur de batterie : Un testeur de batterie mesure la charge restante dans une batterie. Il peut donner une indication plus précise de l'état de la batterie que l'estimation basée uniquement sur la tension. C'est particulièrement utile dans des systèmes où le niveau de la batterie peut avoir un impact significatif sur le fonctionnement.

Utilisation d'un bipeur en cas de tension critique : Un bipeur produit un signal sonore lorsqu'il est activé. Ici, il serait utilisé pour alerter l'utilisateur lorsque la tension de la batterie tombe en dessous d'un certain seuil critique (10V). Cette amélioration permet une réaction rapide à un état de batterie faible, qui pourrait passer inaperçu.

Utilisation d'une batterie lithium polymère 3 cellules : Les batteries au lithium polymère (ou LiPo) sont des types de batteries rechargeables qui offrent plusieurs avantages par rapport aux autres types de batteries, y compris une densité d'énergie plus élevée, un taux de décharge plus élevé, et la possibilité de prendre presque n'importe quelle forme. L'utilisation d'une batterie LiPo 3 cellules fournirait une tension nominale de 11.1V, ce qui serait suffisant pour alimenter le système en 12V.

Utilisation d'un testeur de batterie qui déclenche un signal sonore en cas de sous-alimentation : Comme mentionné précédemment, un testeur de batterie peut fournir une indication précise de l'état de la batterie. Cependant, ce testeur particulier serait également capable de déclencher un signal sonore si la tension de la batterie tombe en dessous de 10V. Cela offrirait un double avantage : non seulement il donnerait une indication précise de l'état de la batterie, mais il alerterait également l'utilisateur de toute sous-alimentation, permettant une intervention rapide.

6.3 Ce que nous avons retenu

Ce projet s'est avéré être une expérience d'apprentissage très enrichissante pour nous deux, à la fois en termes de connaissances acquises et de compétences pratiques développées. En effet, la première phase du projet robot au Semestre 5 nous a permis de développer des compétences manuelles, comme l'apprentissage de la soudure. De plus, la seconde phase développée au Semestre 6 nous a permis d'appliquer les connaissances et les techniques acquises pour résoudre des problèmes pratiques que nous pourrions rencontrer dans notre futur professionnel. Par exemple, Théo a particulièrement apprécié la partie codage, trouvant l'apprentissage du C embarqué très gratifiant. De son côté, Cyril a beaucoup aimé gérer la carte et chercher des bugs matériels. Ce projet a donc été une excellente occasion d'apprendre et de progresser dans un contexte différent de celui, plus formel, des cours magistraux.

7 Code C

7.1 Programme principal

main.c

```
#include "initialisations.h"
#include <p18f2520.h>
#include <stdio.h>
#include "MI2C.h"

#pragma config OSC = INTIO67
#pragma config PBADEN = OFF, WDT = OFF, LVP = OFF, DEBUG = ON

void main(void)
{
    init_uart();
    init_frequence();
    init_entrees_sorties();
    init_moteurs();
    init_timer();
    init_I2C();
    init_ADC();
    valider_IT();

    while (1)
    {
        if (flag_batterie == 1)
        {
            flag_batterie = 0;     // le flag est remis a 0
            surveillance_batt();   // on démarre la surveillance batterie
        }
        if (flag_capteurs_IR == 1)
        {
            flag_capteurs_IR = 0;  // le flag est remis a 0
            surveillance_capt();   // on démarre la lecture des capteurs
    IR
        }
        if (flag_telecommande == 1)
        {
            flag_telecommande = 0; // le flag est remis a 0
            surveillance_tele();   // début surveillance télécommande
        }
    }
}
```

7.2 Initialisations : header

initialisations.h

```
#ifndef INITIALISATIONS_H
#define INITIALISATIONS_H

#include <p18f2520.h>
#include "MI2C.h"

#include "fonctions.h"
#include "interruptions.h"

void init_frequence(void);
void init_entrees_sorties(void);
void init_moteurs(void);
void init_timer(void);
void init_uart(void);
void init_I2C(void);
void init_ADC(void);
void valider_IT(void);

#endif INITIALISATIONS_H
```

7.3 Initialisations : source

initialisations.c

```
#include "initialisations.h"

void init_frequence(void) // fréquence d'horloge = 8MHz
{
    OSCCONbits.IRCF0 = 1;
    OSCCONbits.IRCF1 = 1;
    OSCCONbits.IRCF2 = 1;
}

void init_entrees_sorties(void)
{
    // entrées
    TRISCbits.TRISC3 = 1; // SCL en entree (p176)
    TRISCbits.TRISC4 = 1; // SDA en entree (p176)
    TRISAbits.RA0 = 1;  // IR droit
    TRISAbits.RA1 = 1;  // IR gauche

    // sorties
    TRISCbits.RC1 = 0; // PWM Droit
    TRISCbits.RC2 = 0; // PWM Gauche
    TRISBbits.RB5 = 0; // LED Test
    TRISBbits.RB1 = 0; // IR_ON (alim)
    PORTBbits.RB1 = 0;
}

void init_moteurs(void)
{
    T2CONbits.T2CKPS1 = 0; // Prescaler = 4 (p135)
    T2CONbits.T2CKPS0 = 1;
    PR2 = 124;
    CCPR1L = 0; // le robot est initialement immobile
    CCPR2L = 0;
    CCP1CONbits.DC1B0 = 0; // p149
    CCP1CONbits.DC1B1 = 0;
    CCP2CONbits.DC2B0 = 0;
    CCP2CONbits.DC2B1 = 0;

    CCP1CONbits.CCP1M3 = 1; // mode PWM
    CCP1CONbits.CCP1M2 = 1;
    CCP2CONbits.CCP2M3 = 1;
    CCP2CONbits.CCP2M2 = 1;
```

```
    T2CONbits.TMR2ON = 1; // Timer2
    T2CONbits.T2OUTPS = 9;
    PIE1bits.TMR2IE = 1;
}

void init_uart(void)
{
    BAUDCONbits.BRG16 = 1;
    TXSTAbits.BRGH = 1; // p204
    TXSTAbits.SYNC = 0; // p207

    // p207
    SPBRG = 207;
    SPBRGH = 0;

    TRISCbits.RC6 = 1;
    TRISCbits.TRISC6 = 1; // TX1 en entree
    TRISCbits.TRISC7 = 1; // RX1 en entree
    RCSTAbits.SPEN = 1;   // Port srie activ (p205)
    TXSTAbits.TXEN = 1;   // Activation transmission (p204)
    RCSTAbits.CREN = 1;   // Activation rception continue (p205)
}

void init_I2C(void)
{
    SSPSTAT = 0x80;     // Slew rate 100KHz
    SSPCON1 = 0x28;     // Master Mode Enable, Sclock = FOSC/(4 * (SSPADD
    + 1)) I2C bus
    SSPCON2 = 0x00;
    SSPADD  = 0x27;     // 50kHz - Prediviseur pour SCL = 100KHz a 8MHz
}

void init_ADC(void)
{
    // p227
    ADCON1bits.VCFG0 = 0;
    ADCON1bits.VCFG1 = 0;

    ADCON1bits.PCFG0 = 0; // entrees analogiques AN0, AN1 et AN2
    ADCON1bits.PCFG1 = 0;
    ADCON1bits.PCFG2 = 1;
    ADCON1bits.PCFG3 = 1;

    ADCON2bits.ADCS0 = 0; // $T_{AD}$ = 8 * Tosc = 1 µs
    ADCON2bits.ADCS1 = 0;
    ADCON2bits.ADCS2 = 1;
```

```
    ADCON2bits.ACQT0 = 0;
    ADCON2bits.ACQT1 = 0;
    ADCON2bits.ACQT2 = 1;

    PIE1bits.ADIE = 1;

    ADCON2bits.ADFM = 0; // on justifie à droite
    ADCON0bits.ADON = 1; // activation du CAN
    ADCON0bits.GO = 1;   // début de la conversion

    ADCON0bits.CHS0 = 0;
    ADCON0bits.CHS1 = 1;
    ADCON0bits.CHS2 = 0;
    ADCON0bits.CHS3 = 0;
    PIE1bits.ADIE = 1;
    PIE1bits.TXIE = 1;
    PIR1bits.TXIF = 0;
}

void init_timer(void) // Timer0
{
    // p125
    T0CONbits.T08BIT = 0; // timer en 16 bits
    T0CONbits.T0CS = 0;   // horloge interne
    T0CONbits.T0PS = 1;   // PRE = 1:4

    T0CONbits.PSA = 0;     // L'horloge du Timer0 vient de la sortie du
    préscalaire
    INTCONbits.TMR0IE = 1; // on active l'IT overflow du Timer0

    TMR0H = 0xD8; // B1 bits de poids fort
    TMR0L = 0xEF; // DF bits de poids faible

    T0CONbits.TMR0ON = 1; // on active le Timer0
}

void valider_IT(void)
{

    INTCONbits.INT0IE = 1;
    INTCON2bits.INTEDG0 = 0; // front descendant
    INTCONbits.GIE = 1;      // on active les IT globales

}
```

7.4 Interruptions : header

interruptions.h

```
#ifndef INTERRUPTIONS_H
#define INTERRUPTIONS_H

#include "initialisations.h"

extern volatile char flag_telecommande;
extern volatile char flag_capteurs;
extern volatile char flag_batterie;
extern volatile char flag_moteur;
extern volatile char robot; // variable d'état (1 = marche, 0 = arret)

void HighISR(void); // interruption haute

#endif INTERRUPTIONS_H
```

7.5 Interruptions : source

interruptions.c

```
#include "interruptions.h"

volatile unsigned char compteur = 0;
volatile char flag_telecommande = 0;
volatile char flag_capteurs_IR = 0;
volatile char flag_batterie = 0;
volatile char robot = 0;

#pragma code HighVector = 0x08

void IntHighVector(void)
{
    _asm goto HighISR _endasm
}

#pragma code
#pragma interrupt HighISR

void HighISR(void)
{
    if (INTCONbits.INT0IF) // IT telecommande (Int0)
    {
        INTCONbits.INT0IF = 0; // IT acquittée
        flag_telecommande = 1;
    }

    if (INTCONbits.TMR0IF) // IT Timer0
    {
        INTCONbits.TMR0IF = 0; // IT acquittée
        flag_capteurs_IR = 1;
        compteur++;

        // p.17 pour les valeurs
        TMR0H = 0xB1;
        TMR0L = 0xDF;

        if (compteur == 10) // le compteur se déclenche toutes les 200 ms
        {
            flag_batterie = 1;
            compteur = 0;
        }
    }
}
```

7.6 Fonctions : header

fonctions.h

```
#ifndef FONCTIONS_H
#define FONCTIONS_H

#include "initialisations.h"

#define ADRESSE_U8 0x40     // adresse des 8 LED
#define ADRESSE_TELEC 0xA2  // adresse de la telecommande
#define ADRESSE_BOUTON 0x33 // adresse de la touche du milieu
#define EXIT_SUCCESS 0
#define EXIT_FAILURE -1
#define PWM_G 62
#define PWM_D 62
#define STOP_MOTEURS 0
#define NB_MESURES_BATT 4           // nombre de mesures pour la batterie
#define NB_MESURES_IR 8             // nombre de mesures pour les capteurs
#define VALEUR_TENSION_CRITIQUE 159 // 595
#define VALEUR_DISTANCE_IR_MIN 58   // 71 // 0.4 m
#define VALEUR_DISTANCE_IR_MAX 20   // 269 // 1.5 m

extern volatile unsigned int UBAT;
extern volatile unsigned int IR_D;        // distance capteur IR droit
extern volatile unsigned int IR_G;        // distance cpateur IR gauche
extern volatile unsigned int distance_IR; // distance moyenne capteurs IR

void surveillance_batt(void);
void surveillance_tele(void);
void surveillance_capt(void);
void vitesse_moteur(int PWM_droite, int PWM_gauche);
void affichage_LED_I2C(unsigned char adresse, unsigned char donnee);
// allumage des LED via le bus I2C
void affichage_batterie(int valeur_batterie);
// affichage de la batterie sur les 8 LED
int moyenne(unsigned int liste[], int taille_liste);

#endif FONCTIONS_H
```

7.7 Fonctions : source

fonctions.c

```c
#include "fonctions.h"

volatile unsigned int dist = -1;   // Distance moyenne lue
volatile unsigned int dist_D = -1; // Distance lue par le capteur droit
volatile unsigned int dist_G = -1; // Distance lue par le capteur gauche
volatile unsigned int UBAT = 255;

int liste_UBAT[NB_MESURES_BATT] = {VALEUR_TENSION_CRITIQUE,
    VALEUR_TENSION_CRITIQUE, VALEUR_TENSION_CRITIQUE,
    VALEUR_TENSION_CRITIQUE};
int liste_dist[NB_MESURES_IR] = {50, 50, 50, 50, 50, 50, 50, 50};

void affichage_LED_I2C(unsigned char adresse, unsigned char donnee)
{
    SSPCON2bits.SEN = 1; // on envoie Start sur le bus I2C
    while (!PIR1bits.SSPIF)
        ; // on boucle jusqu'à ce que le signal soit bien envoyé
    PIR1bits.SSPIF = 0;

    SSPBUF = adresse; // on envoie l'adresse de l'appareil à atteindre
    while (!PIR1bits.SSPIF)
        ;
    PIR1bits.SSPIF = 0;

    while (SSPCON2bits.ACKSTAT)
        ; // on attend que la cible réponde avec un signal d'acquittement
    (ACK) pour confirmer la réception de l'adresse

    SSPBUF = donnee; // on envoie la donnee a la cible
    while (!PIR1bits.SSPIF)
        ;
    PIR1bits.SSPIF = 0;

    while (SSPCON2bits.ACKSTAT)
        ; // on attend que la cible réponde en envoyant un signal d'
    acquittement (ACK) pour confirmer la réception des donnees

    SSPCON2bits.PEN = 1; // on envoie un signal de fin (Stop) sur le bus
    I2C pour indiquer la fin de la transmission
    while (!PIR1bits.SSPIF)
        ; // on attend que le signal de fin soit envoyé avec succès
    PIR1bits.SSPIF = 0;
}
```

```
void vitesse_moteur(int PWM_droite, int PWM_gauche)
{
    CCPR1L = PWM_droite; // PWM Droite
    CCPR2L = PWM_gauche; // PWM Gauche
    PORTAbits.RA6 = 0; // DIRD
    PORTAbits.RA7 = 0; // DIRG
}

int moyenne(unsigned int liste[], int taille_liste)
{
    int valeur_moyennee = 0;
    int i;
    for (i = 0; i < taille_liste; i++)
    {
        valeur_moyennee = valeur_moyennee + liste[i];
    }
    return valeur_moyennee / taille_liste;
}

void surveillance_capt(void)
{
    int val_moy, i; // utile pour stocker la moyenne des distances

    // décalage des valeurs vers la droite
    for (i = NB_MESURES_IR-1; i>0; i--) liste_dist[i] = liste_dist[i-1];

    ADCON0bits.CHS = 0; // channel 0
    ADCON0bits.GO = 1;
    while (ADCON0bits.DONE);
    dist_D = (int)ADRESH;

    ADCON0bits.CHS = 1; // channel 1
    ADCON0bits.GO = 1;
    while (ADCON0bits.DONE);
    dist_G = (int)ADRESH;

    dist = (dist_G + dist_D) / 2; // moyenne des deux capteurs
    liste_dist[0] = dist;
    val_moy = moyenne(liste_dist, NB_MESURES_IR); // moyenne des 8 mesures

    printf("dist_D : %d \r\ndist_G : %d\r\n\r\n", dist_D, dist_G);

    if (robot == 1 && val_moy > VALEUR_DISTANCE_IR_MAX && val_moy <
    VALEUR_DISTANCE_IR_MIN) vitesse_moteur(PWM_D, PWM_G);
    else vitesse_moteur(STOP_MOTEURS, STOP_MOTEURS);
}
```

```
void surveillance_tele(void)
{
    char buffer_telec[3]; // tableau utilisé pour stocker les donnees lues
    Lire_i2c_Telecom(ADRESSE_TELEC, buffer_telec);
    if (buffer_telec[1] == ADRESSE_BOUTON) // verification touche centrale
    {
        robot = (robot == 0);
        PORTBbits.RB5 = (robot == 1);
    }
}

void surveillance_batt(void)
{
    int val_moy, i; // Décalage des indices

    for (i = NB_MESURES_BATT-1; i>0; i--) liste_UBAT[i] = liste_UBAT[i-1];

    // Selection du channel 2
    ADCON0bits.CHS0 = 0;
    ADCON0bits.CHS1 = 1;
    ADCON0bits.CHS2 = 0;
    ADCON0bits.CHS3 = 0;

    // Demarrage de la conversion
    ADCON0bits.GO = 1;

    while (ADCON0bits.DONE);

    UBAT = (unsigned int)ADRESH;
    liste_UBAT[0] = UBAT;

    val_moy = moyenne(liste_UBAT, NB_MESURES_BATT); // 4 mesures/seconde ?

    if (val_moy < VALEUR_TENSION_CRITIQUE)
    {
        PORTBbits.RB5 = 1;  // on allume la LED Test
        printf("Valeur batterie : %d (batterie faible)\r\n", val_moy);
        affichage_batterie(val_moy);
    }
    else
    {
        PORTBbits.RB5 = 0;  // on allume la LED Test
        printf("\r\nValeur batterie : %d \r\n", val_moy);
        affichage_batterie(val_moy);
    }
}
```

```c
void affichage_batterie(int valeur_batterie)
{
    int valeur_led = 0b11111111;

    if (1)
    {
        int pourcentage_batterie = 100 * valeur_batterie / 255;
        printf("pourcentage = %d \n\r", pourcentage_batterie);

        if (pourcentage_batterie > 88) valeur_led = 0b00000000;
        else if (pourcentage_batterie > 75 && pourcentage_batterie <= 88)
    valeur_led = 0b10000000;
        else if (pourcentage_batterie > 63 && pourcentage_batterie <= 75)
    valeur_led = 0b11000000;
        else if (pourcentage_batterie > 50 && pourcentage_batterie <= 63)
    valeur_led = 0b11100000;
        else if (pourcentage_batterie > 38 && pourcentage_batterie <= 50)
    valeur_led = 0b11110000;
        else if (pourcentage_batterie > 25 && pourcentage_batterie <= 38)
    valeur_led = 0b11111000;
        else if (pourcentage_batterie > 13 && pourcentage_batterie <= 25)
    valeur_led = 0b11111100;
        else if (pourcentage_batterie > 0 && pourcentage_batterie <= 13)
    valeur_led = 0b11111110;
        affichage_LED_I2C(ADRESSE_U8, valeur_led);
    }
}
```