Projet : Robot suiveur de ligne

Module EEA 0601

TP Transdisciplinaire Année 2023/2024

Sujet : Interface Homme Machine d'un Robot Suiveur de Ligne

Réalisé par ERRARD Mathéo et DACCACHE Christopher

Enseignants responsables : COLAS Maxime et CARTON David

SOMMAIR

INTRODUCTION

Dans le cadre de la matière EEA0601, notre responsabilité consistait à concevoir un robot suiveur de ligne. Ce projet représente une opportunité précieuse d'appliquer les connaissances acquises tout au long de notre formation en électronique et en automatique. Pour mener à bien cette mission, le projet a été subdivisé en trois composantes distinctes :

- Motorisation
- Interface Homme-Machine (IHM)
- Capteurs

Chaque groupe a bénéficié de treize séances, dont dix encadrées et trois en autonomie, en plus des séances personnelles. Un planning détaillé des séances est présenté ultérieurement dans ce rapport.

L'objectif global du projet est de développer une solution technique complète pour un robot capable de suivre une trajectoire prédéfinie grâce à des capteurs à photodiodes. Muni de deux moteurs indépendants, il peut se déplacer de manière linéaire et angulaire en modifiant la vitesse de rotation des moteurs. Ainsi, par le biais de ces moteurs, il peut corriger sa position selon son décalage par rapport à la trajectoire voulue.

Notre binôme s'est focalisé sur la conception de l'interface Homme-Machine (IHM), un élément crucial facilitant la communication entre l'utilisateur et le robot. Notre objectif principal était de développer une interface intuitive et fonctionnelle, intégrant des éléments de contrôle permettant à l'utilisateur de communiquer efficacement avec le robot. Pour ce faire, nous avons travaillé avec trois boutons et un écran d'affichage, des composants clés utilisés pour configurer les paramètres du robot et définir ses actions.

Le succès de notre projet dépendra de la collaboration étroite entre les différentes équipes travaillant sur les différentes parties du robot. Nous devons assurer une intégration fluide de notre système IHM avec les systèmes de motorisation et de capteurs, garantissant ainsi le bon fonctionnement global du robot suiveur de ligne.

La présentation de notre travail sera articulée en trois parties principales. La première consistera à présenter le cahier des charges, le matériel utilisé ainsi qu'un planning d'avancement du projet. La deuxième partie décrira la partie électronique réalisée, tandis que la dernière partie expliquera la programmation et l'asservissement des moteurs.

I. Section Préliminaire

a. Cahier des charges

i) Concevoir et implémenter une interface homme-machine pour un robot suiveur de ligne afin de pouvoir configurer différents paramètres.

ii) L'IHM doit être équipée de trois boutons, chaque bouton étant assigné à une fonction spécifique. Un écran d'affichage doit y être intégré, permettant à l'utilisateur de visualiser les paramètres du robot et de sélectionner différentes actions.

iii) Afficher différentes informations telles que la vitesse du robot, les données des capteurs comme la distance par rapport à un obstacle ou encore sa déviation par rapport à la ligne. Mais également la sélection des actions à effectuer par le robot, comme une marche lente ou rapide ou encore la correction de trajectoire.

iv) L'IHM doit être capable de recevoir et de traiter les données des capteurs du robot et de pouvoir envoyer des instructions aux moteurs du robot en fonction des informations reçues des capteurs et de la configuration que souhaite l'utilisateur.

b. Description du matériel

Pour mener à bien à bien ce projet une liste du matériel nécessaire pour chaque partie a été établit :

Capteurs	Moteurs	IHM
Carte DsPic	Carte DsPic	Carte DsPic
Capteurs de position (photos diodes) IR TCRT 5000	Moteur	3 boutons de type DT6
Capteurs d'obstacles (à ultrasons) : SRF 05	BTN 8982	Ecran LCD : EA DOGM163W-A

Partie Interface Homme-Machine	
Composants	**Quantité**
Boutons DT6	3
LCD (EA DOGM163W-A)	1
Nape électrique (26 pates)	1
Résistance (33ohm)	1
Condensateur électrochimique (100µF)	1
Condensateur électrique (100nF)	1
CON-1X2	2

c. Planning

Voici un tableau indiquant les dates et descriptions de toutes les séances réalisées (Encadrées et Autonomie).

Date	Descriptions	Membre
24/01	Explication du projet par les professeurs.	C /M[1]
31/01	Câblage de l'écran LCD et allumage du rétroéclairage.	C /M
07/02	Programmation : Affichage d'un message sur l'écran LCD. Branchement des boutons et leurs résistances.	C/M
14/02	Programmation : Affichage de l'état des boutons (appuyé ou relâché) sur l'écran.	C/M
21/02	Eagle : Ajout des librairies et réalisation du schéma de la carte électronique. Réalisation de la carte électronique.	C/M
24/02 au 02/03	Programmation : « Menu » permet de choisir une action à l'aide des boutons. (Vacances).	C
06/03	Eagle : Correction du schéma et de la carte électronique. Programmation : Correction du code « Menu ».	C/M
13/03	Eagle : Routage des pistes. Modification du choix des pattes des boutons : utilisation des CN (port B) pour les interruptions.	C/M
20/03	Eagle : Finalisation de la carte électronique.	C/M
27/03	Perçage de la carte, test de continuité à l'ohmmètre et soudure.	C/M
03/04	Programmations : Modification du programme en utilisant les interruptions sur les boutons. Ajout des structures (Keyboard et Menu)	C/M
10/04	Changement de carte µC. Ajout de la fonction menu_principale() et affichage du menu demandé (Actions/ Paramètres/Affiche Etat)	C/M
17/04	Programmation : on a géré la durée d'appui des boutons. Finalisation des sous menus et insertion des structures MenuItem et MenuValue	C/M
22/04 au 29/04	Programmation : Fonction menu_principale() quasiment fini. Affichage d'un int qu'on peut modifier à l'aide des boutons (Vacances).	C/M
07/05	Finalisation du menu avec paramétrages	C/M

Un total de 9 séances ont été réalisé en dehors des horaires de TP afin d'avancer le plus possible sur le projet.

Séances	Nombre de Séances	Travail Effectué
Encadré	10	Développement
Autonomie	5 (2 vacances)	Tests et Développement
Supplémentaires	9	Tests

[1] C : Christopher / M : Mathéo

II. Conception électronique

a. Câblage de l'afficheur

L'afficheur utilisé est écran LCD de référence EA DOGM163W-A de dimention 16x3. Il possède 24 pates de connection dont treize sont relié au +5V, cinq a la masse, quatre au microcontrôleur et deux non utilisé. Le branchement entre le micro-contrôleur et l'afficheur se fait par une nappe de 26 pates.

Selon les spécifications fournies dans la notice (document 10 en référence), pour activer l'écran (rétro-éclairage), il est nécessaire de connecter une résistance de 30Ω en parallèle avec les broches 1 et 20 vers le +5V, tel que représenté sur la figure ci-dessous. Dans notre cas, nous avons utilisé une résistance de 33Ω a cause de l'indisponibilité d'une resistance de 30Ω. Les pins 1 et 20 correspondent à des anodes et les pins 2 et 19 à des cathodes.

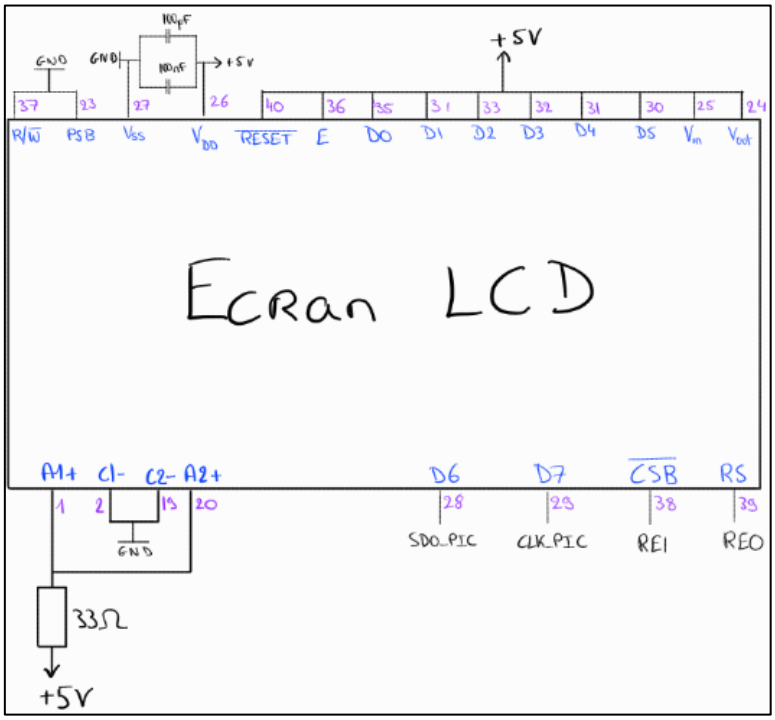


FIGURE 1 : Brochage de l'écran LCD

Pour les broches de communication RS, CSB, D6 et D7 celles-ci sont respectivement connectées aux broches RE0, RE1, RE3 et RE4. Initialement, nous avions sélectionné les broches RB pour cette fonctionnalité. Cependant, ce choix a été revu afin de réserver les broches RB pour le raccordement des boutons, ces broches représentant également les broches CN utilisées pour les interruptions. La broche 40, correspondant à la fonction RESET, a été relié au +5V plutôt qu'au port RE2, afin d'éviter toute interférence potentielle avec la fonction de réinitialisation.

Afin de maintenir une alimentation stable et minimiser les distorsions de tension d'entrée, deux condensateurs de découplage ont été installés en parallèle, à proximité des broches d'alimentation de l'écran. Le premier est un condensateur électrochimique de 100 µF, utilisé pour atténuer les variations lentes de tension sur l'alimentation. Le second est un condensateur en plastique de 100nF, destiné à lisser les fluctuations rapides de tension sur l'alimentation, assurant ainsi une tension stable sur une plage de fréquences élevées.

b. Câblage des boutons

Pour le câblage de nos trois boutons DT6, nous avons inclus des résistances (10KΩ) en série avec ces derniers. Dans un circuit, ces boutons fonctionnent comme des interrupteurs, pouvant être configurés en tant qu'interrupteurs normalement ouverts ou normalement fermés, selon la configuration requise. La figure ci-dessous illustre les deux configurations possibles pour le câblage des boutons.

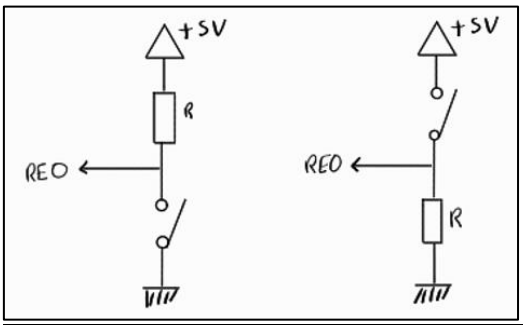


FIGURE 2 : CONFIGURATION BOUTON

Dans notre cas, nous avons opté pour la configuration de gauche. Dans cette configuration, les boutons sont normalement ouverts, ce qui signifie qu'ils renvoient un signal de 0V sur les broches auxquelles ils sont connectés lorsqu'ils ne sont pas enfoncés. Lorsqu'un bouton est enfoncé, il envoie un signal de 5V sur sa broche de connexion. La figure suivante nous résume ceci :

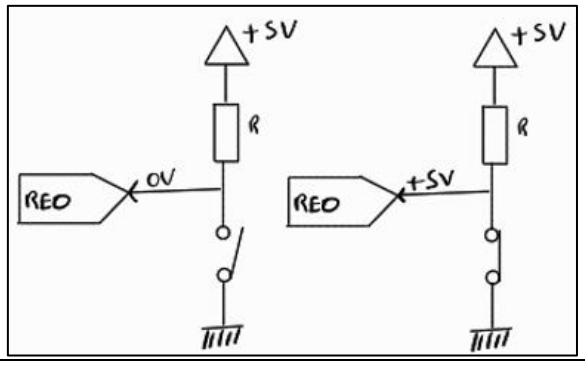


FIGURE 3 : LECTURE D'APPUI SUR UN BOUTON

Pour déterminer si un bouton est enfoncé ou non, il suffit de lire la valeur reçue sur la broche correspondante du microcontrôleur. Nos trois boutons sont de couleurs différentes : un rouge, un bleu et un noir, et leurs fonctions sont définies comme suit :

- Rouge (+) : Incrémenter des valeurs numériques ou de monter dans un menu ou sous-menu.
- Bleu (-) : Décrementer des valeurs numériques ou de descendre dans un menu ou sous-menu.
- Noir (entrée) : Valider une valeur ou sélectionner une entrée dans un menu ou sous-menu .

Ces boutons nous permettent de naviguer dans les menus et les sous-menus, de sélectionner des options et de valider des actions.

Ce cablage de boutons a été modifié, puisque le microcontroleur a des résistances intégrer aux pates RB appeler pull-up resistors. Ces resistances vont guarentire un état logique (0) lorsque le bouton n'est pas enfoncé.

c. Schéma électronique

Après le câblage de l'écran ainsi que des boutons sur la plaque à essai, et après verification du bon fonctionnement de notre circuit, nous avons réalisé sur le logiciel Eagle le schéma de la carte électronique.

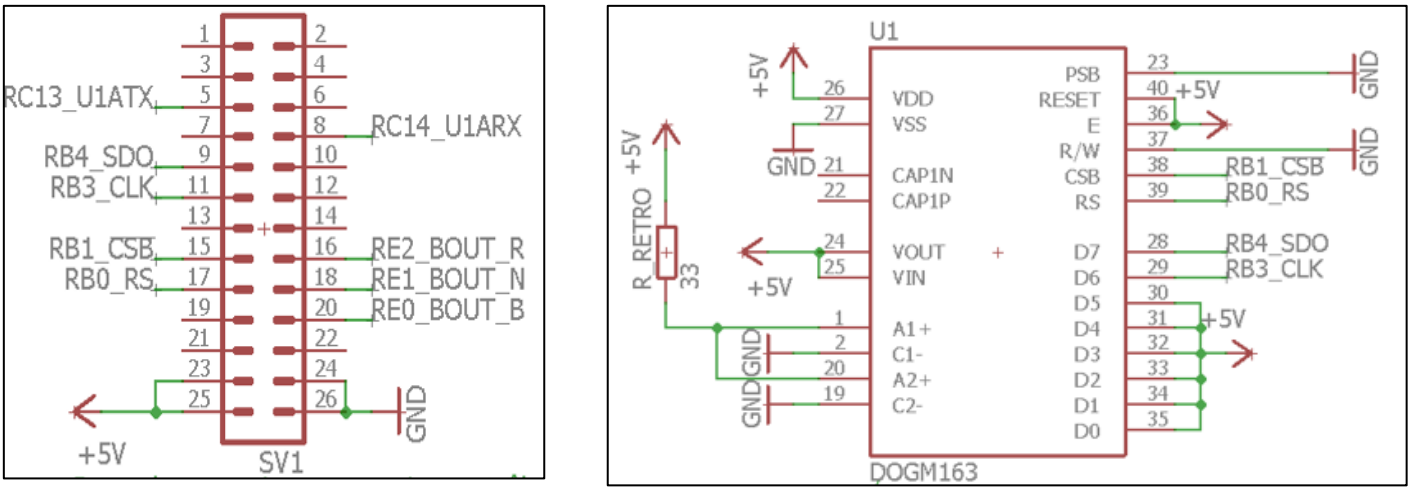
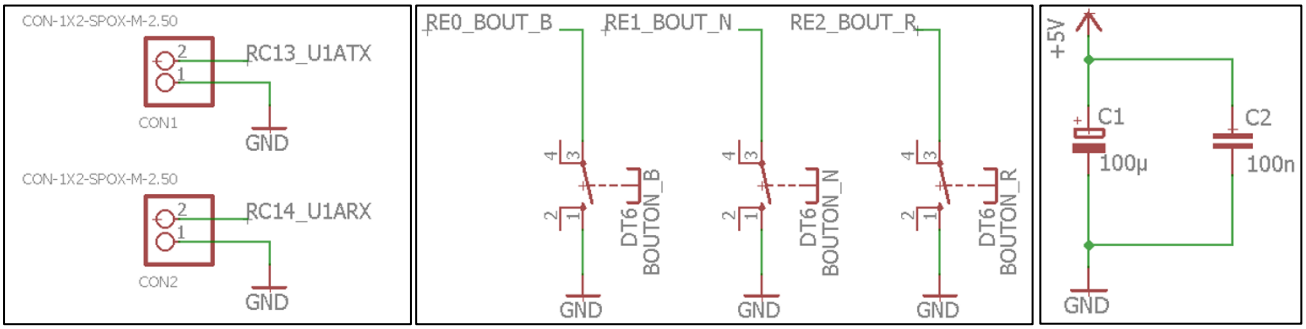


FIGURE 4 : SCHÉMA EAGLE



Pour pouvoir communiqué avec les autres microcontroleurs (capteurs et motorisaton) à l'aide d'une liaison RS232, nous avons donc placé deux connectiques Molex 22035025. L'un servira à recevoir des informations des capteurs et l'autre à envoyer des informations à la motorisation. L'image suivante nous montre comment se fait le branchement:

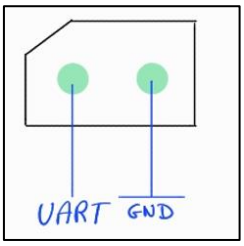


FIGURE 5 : CONNECTIQUES

La connexion se fait par les pattes UART du µC. La patte U1ATX sert à la réception de données et la patte U1ARX sert à la transmission de données.

d. Carte électronique

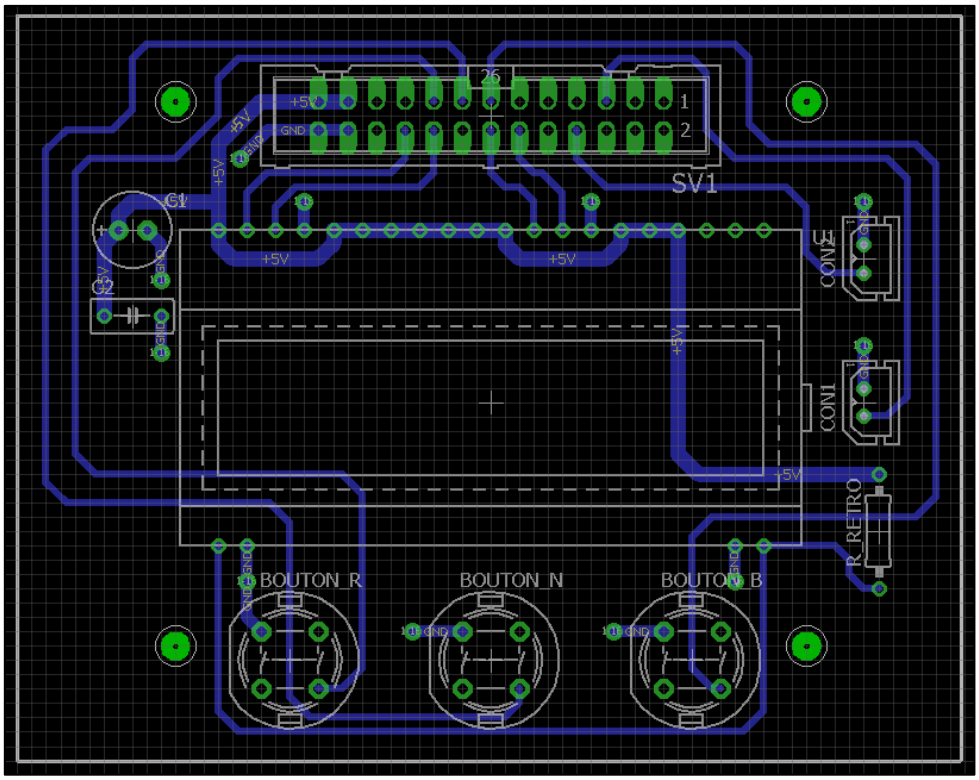


Figure 6 : carte électronique finale

Après la conception du schéma électrique, le logiciel Eagle a permis la génération des composants à placer pour la fabrication de la carte. La réalisation de la carte s'est déroulée en trois étapes principales :

- Placement des composants :

  Les composants ont été positionnés initialement de manière à faciliter l'utilisation des boutons, à optimiser la visibilité de l'écran et à simplifier le raccordement de la nappe et des connecteurs au microcontrôleur.

- Routage des pistes :

  Lors de la conception du routage des pistes, une réorganisation de l'emplacement des condensateurs a été réalisée afin de les placer aussi près que possible de l'entrée d'alimentation de l'écran. Ceci visait à garantir une alimentation la plus stable possible en entrée, limitant ainsi les interférences et les distorsions du signal.

- Optimisation :

  Des mesures d'optimisation ont été prises, notamment la création d'un plan de masse, la réduction de la longueur des pistes et la minimisation des angles droits. Ces actions visaient à atténuer les perturbations électromagnétiques potentielles et à améliorer la qualité globale du signal.

Enfin, des trous ont été prévus et dimensionnés pour permettre une fixation appropriée de la carte sur le support du robot, assurant ainsi une intégration harmonieuse dans le système global.

III. Conceptions informatiques

a. Premier programme

Initialement, nous avons entamé la programmation et l'affichage d'un message simple afin de nous familiariser avec le logiciel. Nous avons choisi d'afficher le message "Bonjour le monde". La figure suivante correspond à l'organigramme de ce message.

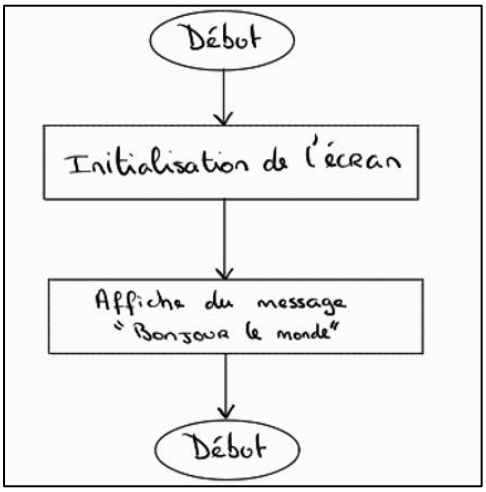


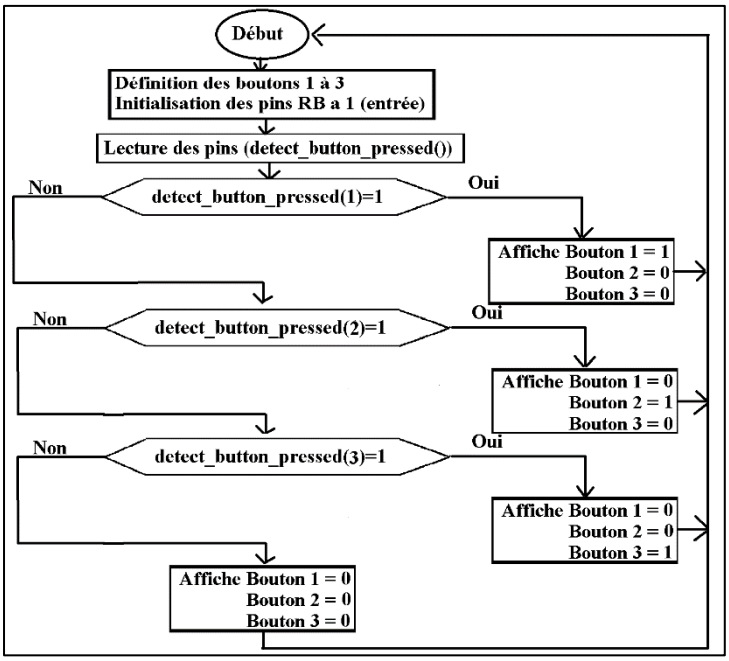
Figure 7 : Organigramme du premier code

```
InitLCD();
LCDDisplayOn();
LCDGoto(0,0);
LCDWriteStr("Bonjour le monde");
```

Figure 8 : Code du premier programme

b. Programme Boutons 1

Initialement, nous avons élaboré un code de test afin de vérifier le bon fonctionnement de la fonction detect_button_pressed(). L'organigramme de ce code est illustré dans la figure suivante. Par la suite, nous avons apporté des modifications à ce code afin de détecter et afficher l'appui sur un ou plusieurs boutons. Durant cette phase, nous avons commencé par programmer sans utiliser les interruptions.



Le programme de cet organigramme est présent dans la partie annexe (III.b.1)du compte rendu.

FIGURE 9 : Organigramme Programme Boutons 1

c. Programme Boutons 2

Dans une deuxième étape, nous avons réécrit cette même fonction en utilisant des interruptions. L'utilisation des interruptions garantit une exécution plus fluide du programme. En plus de l'implémentation des interruptions, pour rendre le code plus structuré, nous avons utilisé une structure de type "typedef" déclarée globalement, qui enregistre l'état des trois boutons lors de l'interruption. La figure suivante illustre l'organigramme de ce code.

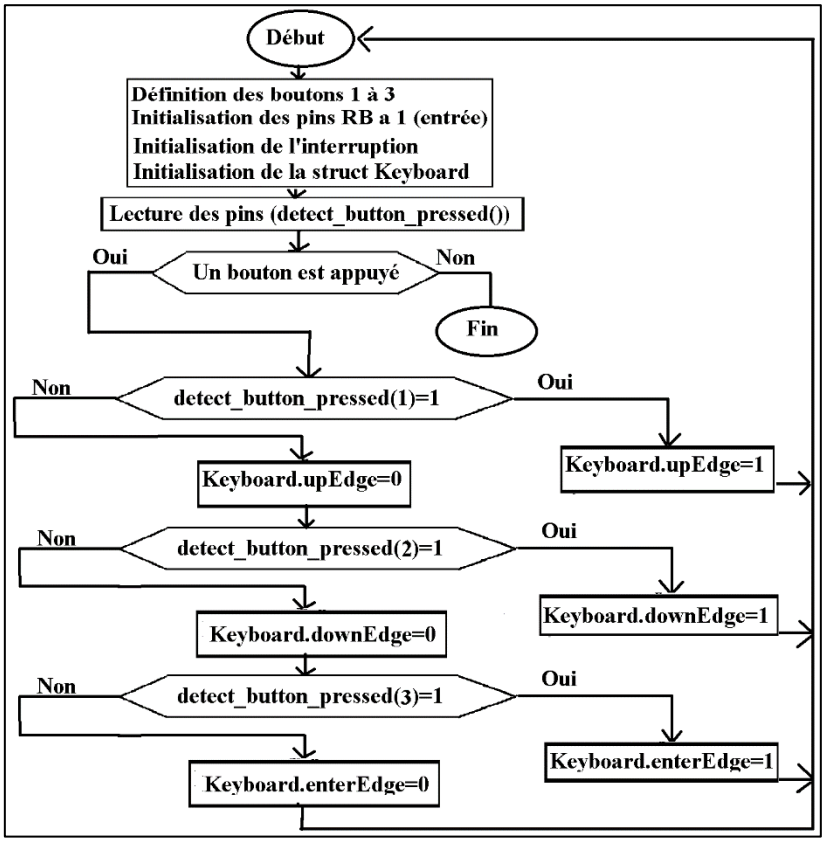


Figure 10 : Organigramme Programme Bouton 2

**Le programme de cet organigramme est présent dans la partie annexe (III.b.2).*

d. Programme Menu

Une fois la conception et les tests du programme de contrôle des boutons terminés, et leur bon fonctionnement confirmé, nous avons entrepris le développement du menu permettant l'interaction avec notre robot. Ce menu a été divisé en trois parties principales, chacune comprenant ses propres sous-parties. Dans sa première version, le menu était peu structuré et principalement implémenté à l'aide d'instructions conditionnelles "if" et "else if". Afin d'améliorer la lisibilité et la maintenabilité de notre code, nous avons décidé de restructurer notre programme en le dissociant de notre code principal. Pour ce faire, nous avons créé deux fichiers distincts : un fichier .c (fonctions_sup.c) et un fichier .h (fonctions_sup.h), où nous avons déplacé les fonctionnalités liées au menu.

fonctions_sup.c fonctions_sup.h mains.c

Ensuite, nous avons ajouté des structures "typedef" dans le fichier .h pour initialiser de manière plus structurée les différentes parties et sous-parties du menu de façon globale. La structure du menu est représentée par l'organigramme suivant (*référence : Guide de réalisation du projet EEA0601.pdf*) :

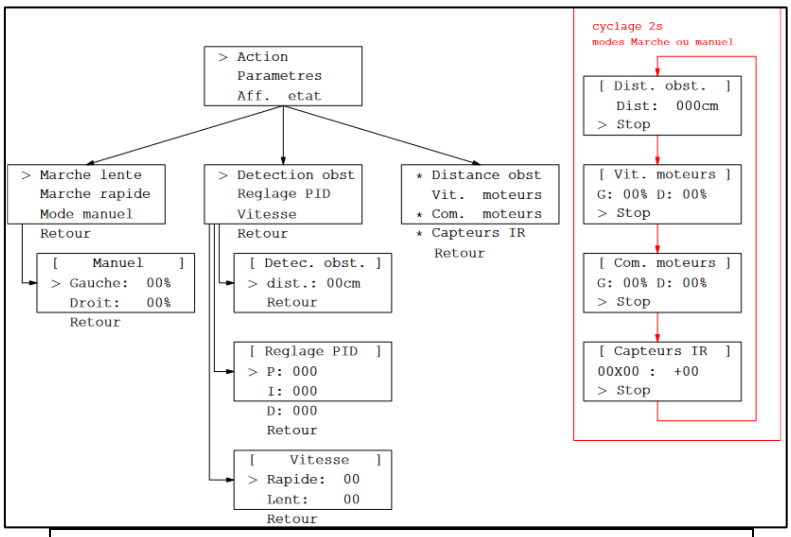


Figure 11 : Organigramme de la structure du menu

Les structures "typedef" qui renferment toutes les informations du menu ont été nommées "Menu, MenuItem, MenuValue". La structure Menu contient 5 types d'informations (titre, root, items, offset et selected), la structure MenuItem contient 4 types d'informations (label, action, type et checked) et enfin, la structure MenuValue contient 5 types d'informations (tuningValue, tunedValue, min, max et type). Le code d'initialisation des structures est présenté ci-dessous :

```
typedef struct menu Menu;
typedef struct menuitem MenuItem;
typedef struct menuvalue MenuValue;
struct menu{
    char *title;
    Menu *root;
    MenuItem *items;
    short offset:6;
    short selected:4;
};

struct menuitem{
    char *label;
    void *action;
    short type:2;
    short checked:1;
};

struct menuvalue{
    int tuningValue;
    int *tunedValue;
    int min;
    int max;
    short type;
};
```

Figure 12 : Programme initialisation Structures

<u>Lexique :</u>

La structure Menu comprendra 8 sous-parties (root, action, param, affetat, manu, detec, PID et vit). Pour accéder aux paramètres de chaque Menu, la syntaxe sera la suivante : par exemple, « rootMenu.title = NULL ».

La structure MenuItem comportera 7 sous-parties (root, action, param, manu, detec, PID et vit). Pour accéder aux paramètres de chaque MenuItem, la syntaxe sera de la forme : par exemple, « rootItems[0].label = 'Actions' ».

La structure MenuValue comprendra 4 sous-parties (manu, dist, pid, vit). Pour accéder aux paramètres de chaque MenuValue, la syntaxe sera de la forme : par exemple, « distValue[0].tuningValue = 10 ».

**La definition de chaque valeur dans les structures Menu, MenuItem et MenuValue est expliquer en annexe (III.d.1).*

Ensuite, nous avons créé une fonction void menu_principale() qui contient l'intégralité du code du menu. Après la création de cette fonction, nous avons initialisé toutes les structures du menu en leur attribuant leurs valeurs désignées. Voici ci-dessous un exemple de chaque structure:

```
#define ACTIONS "Actions        "
#define PARAMETRES "Parametres     "
#define AFFETAT "Aff. Etat      "

Menu rootMenu;

void menu_principale(){

MenuValue manuValue[]={{0,0,0,100,0},{0,0,0,100,0}};

MenuItem rootItems []={{ACTIONS,&actionMenu,2,0},{PARAMETRES,&paramMenu,2,0},{AFFETAT,&affetatMenu,2,0}};

rootMenu.title = NULL;
rootMenu.root =NULL;
rootMenu.items = rootItems;
rootMenu.selected = 0;
rootMenu.offset = 0;

}
```

Figure 13 : Exemple Initialisation des Structures

**Le code complete de l'initialisation des structures est en annexe (III.d.2).*

**N.B : les #define ACTIONS ''Actions'' sont appeler des macros*

Après la création et l'initialisation de toutes les structures du menu, nous avons commencé à programmer l'affichage du menu sur l'écran LCD. Cette partie a été réalisée à l'aide d'une boucle "do while". Cette boucle comporte plusieurs niveaux, représentés par la variable "choix_menu", qui est de type "int". Cette variable peut prendre 3 valeurs : {0 = menu principal, 1 = sous-menu 1 (Actions, Paramètres et Aff. État), et 2 = sous-menu 2 (Mode Manuel, Détection obst., Réglage PID et Vitesse)}.

À chaque pression du bouton "entrer", si une option de sous-menu est sélectionnée par le curseur, "choix_menu" s'incrémente de 1. Si le curseur est positionné sur l'option "retour", "choix_menu" subit une décrémentation de 1. Si une modification de variable des paramètres est en cours et que le bouton "entrer" est pressé, "choix_menu" subit une décrémentation de 1 pour revenir au menu ou sous-menu précédent.

La fonction menu_principale(Keyboard *keyboard) prend comme paramètre une structure de type Keyboard. Cette structure nous permet d'utiliser les valeurs qu'elle contient pour détecter l'appui des boutons et assurer le bon fonctionnement fluide du menu.

**L'organigramme du menu sera réalisé d'une facon simplifier vue la grandeur et complexiter du code du menu.*

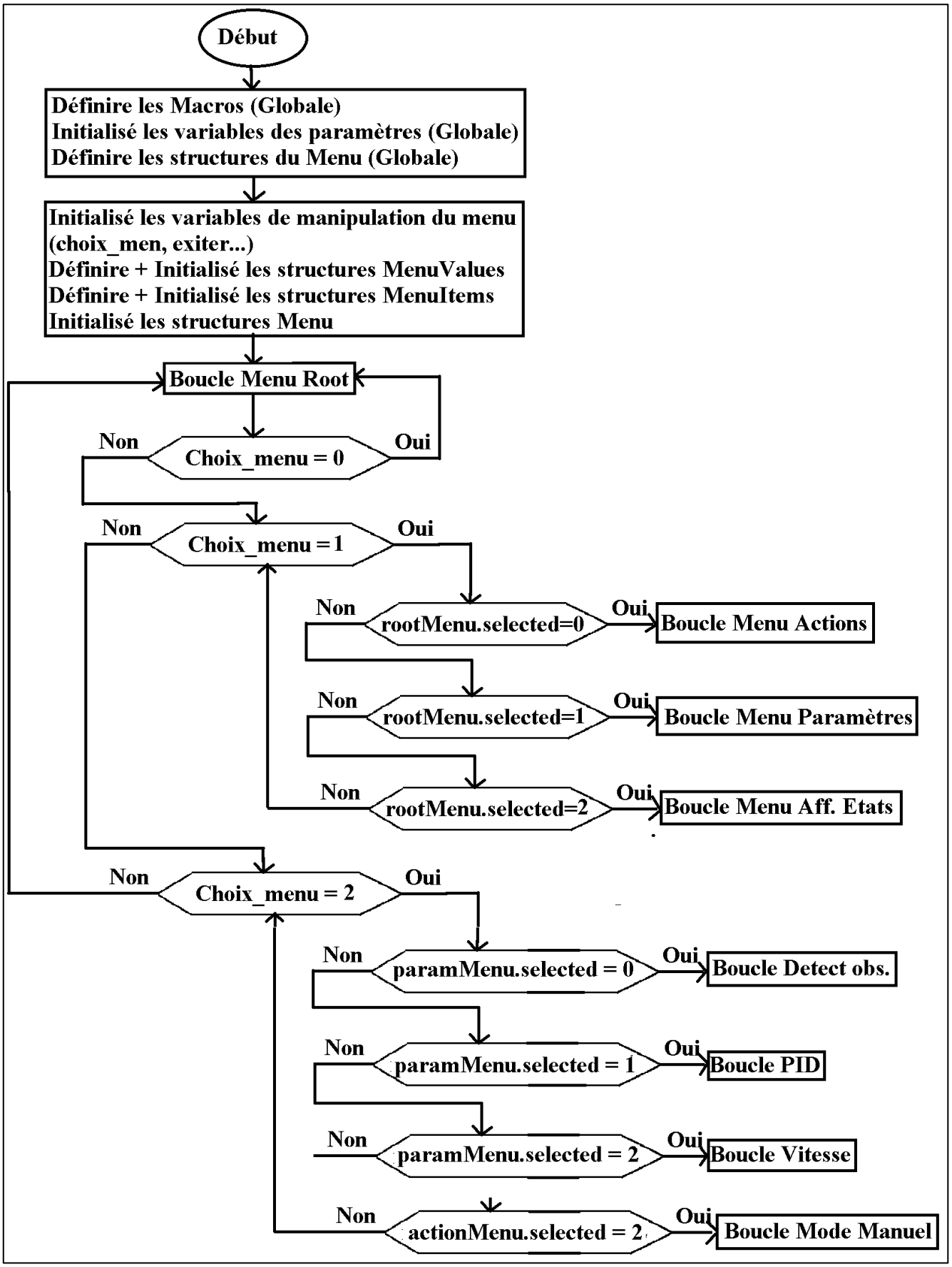


Figure 14 : Organigramme Simplifié du Menu

**Le code complet du menu est en annexe (III.d.3).*

Enfin, après la configuration et la programmation du menu, nous sommes capables de naviguer librement. La partie suivante est cruciale car c'est là que les données nécessaires au déplacement du robot entrent en jeu.

e. Programme Paramètres Menu

Cette section de la programmation est intégrée dans le menu mais elle fait également partie du programme de correction de trajectoire. Les paramètres que nous devons spécifier avant de lancer le code de démarrage du robot sont les suivants :

- Distance d'Obstacle : Cette configuration nous permet de définir une distance maximale [0, distmax] à laquelle le robot doit corriger sa trajectoire avant de heurter l'obstacle.
- PID : Il comprend trois valeurs (facteur proportionnel, facteur intégrateur et facteur dérivatif). Ces trois valeurs sont utiles pour obtenir une réponse plus précise lors de la correction de la trajectoire du robot avec le moins de fluctuations possible.
- Vitesse : Cette valeur nous permet de définir la vitesse maximale pour la "Marche Rapide" et la "Marche Lente" dans le menu "Actions".

L'organigramme du programme des paramètres du menu est ci-dessous :

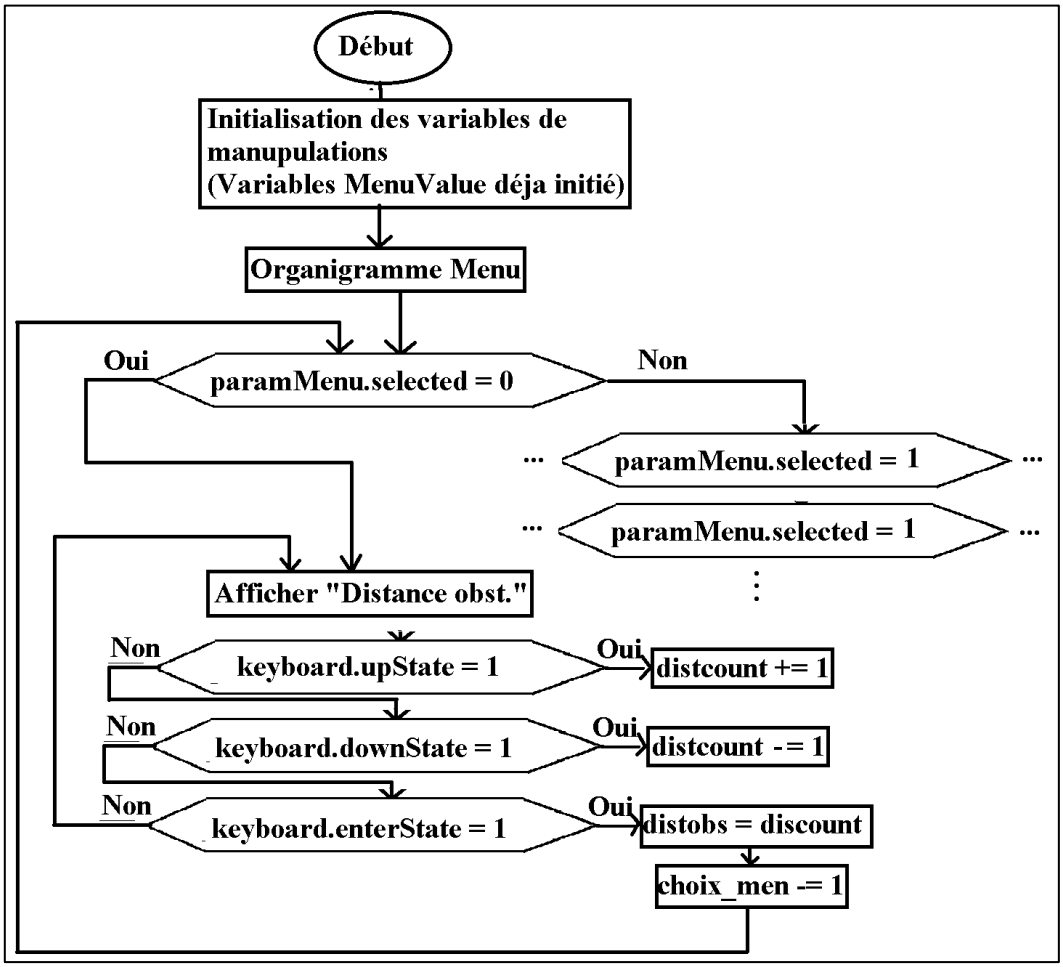


Figure 15 : Organigramme code paramètres du menu

** Le code complet du paramètrage du menu est en annexe (III.d.4).*

Nous avons eu un problème concernant l'affichage de variable sur le menu, car la fonction d'affichage de l'écran ne prend qu'un paramètre de type « char ». Pour résoudre ce problème, nous avons créé une chaîne de caractères, « distance[]={'0','1','2','3','4','5','6','7','8','9'}, contenant les chiffres de 0 à 9, et nous avons configuré l'appui du bouton "+" pour incrémenter et l'appui du bouton "-" pour décrémenter l'indice de la chaîne de caractères "distance[]".

f. Programme Acquisition Données Capteurs

Pour faciliter la communication entre notre microcontrôleur et celui des capteurs, nous connectons la broche U1ATX de notre microcontrôleur à la broche U1RTX de celui des capteurs. Après avoir établi cette liaison physique, il est nécessaire de programmer une interruption des deux parties à une fréquence égale afin de recevoir toutes les données envoyées. La période des signaux envoyés et reçus est de 100 ms.

Nous recevrons deux types de données : une valeur variant entre [-64, 64], qui représente l'information envoyée par les capteurs infrarouges, et une autre valeur variant entre [0, 127], qui correspond à la distance en centimètres entre le capteur ultrasonore et l'obstacle.

Les informations seront reçues sous forme binaire de taille 8 bits. Si le bit 7 est égal à 0, il s'agit de la donnée envoyée par les capteurs infrarouges, tandis que si le bit 7 est égal à 1, il s'agit de la donnée envoyée par le capteur ultrasonore.

Pour recevoir correctement ces informations et les stocker, nous devons créer une interruption au niveau du Timer3, qui s'active toutes les 100 ms et stocke l'information reçue des capteurs dans des variables qui seront mises à jour toutes les 100 ms.

Une fois les données du capteur stockées, il est nécessaire de les analyser et de générer une correction de trajectoire afin que le robot continue à suivre la ligne et évite de heurter des obstacles.

**La partie réception des informations n'a pas pu être réalisée en raison de contraintes de temps.*

g. Programme Envoi Correction Moteur

Pour permettre la communication entre notre microcontrôleur et celui du moteur, nous relions la broche U1RTX de notre microcontrôleur à la broche U1ATX de celui du moteur. Après avoir établi cette liaison physique, il est nécessaire de programmer une interruption des deux parties à une fréquence égale pour pouvoir envoyer la correction de la trajectoire sous forme de vitesse linéaire et de vitesse de rotation. La période des signaux envoyés et reçus est de 100 ms.

Les informations à envoyer seront déterminées à l'aide de l'algorithme écrit pour corriger la trajectoire du robot en fonction des données reçues par les capteurs. Une fois cette correction générée, nous la traduirons en vitesse linéaire et vitesse de rotation pour changer la direction du robot.

**L'algorithme de correction de trajectoire et le code des interruptions et de l'envoi des données aux moteurs n'ont pas été réalisés en raison de contraintes de temps.*

IV.Conclusion

Après un investissement de 15 séances régulières et 9 supplémentaires dédiées, nous avons réussi à mettre en place le programme complet du menu, incluant le système de modification des paramètres. Le menu fonctionne de manière fluide et répond parfaitement à nos attentes. Cependant, nous avons été confrontés à une contrainte majeure : le temps. Malheureusement, cette contrainte ne nous a pas permis de finaliser notre contribution au projet à 100 %. En particulier, la mise en œuvre de la communication entre les cartes et la correction de trajectoire reste inachevée. Malgré cette limitation temporelle, nous avons consacré tous nos efforts à la conception de l'interface de communication homme-machine, tant du point de vue électronique que du point de vue programmation. Si nous avions disposé de plus de temps, nous aurions pu mener à bien ces aspects manquants du projet.

Références :

[1] *Schema_de_la_carte_Dspic.pdf*
[2] *dspic30f4012-2010-02-23-maj-2018-03-21.pdf*
[3] *dsPIC30F_Family_Reference_Manual_-_70046E.pdf*
[4] *dsPIC30F4011-4012_Data_sheet_-_70135G.pdf*
[5] *16-bit_MCU_and_DSC_Programmer_s_Reference_Manual_-_70157F.pdf*
[6] *EEA0601 - Communications.pdf*
[7] *Guide de réalisation du projet EEA0601.pdf*
[8] *EEA0601 - Mode demploi synthétique du logiciel Eagle - 2022-2023.pdf*
[9] *EEA0601 - Principes de réalisation dune carte électronique - 2022-2023.pdf*
[10] *dog-me_DOG_SERIES_3.3V.pdf*

Annexe:

Github : https://github.com/christopherdacc/Projet_EEA0601

II.b.1) Programme Boutons 1

```
#define BUTTON1_PIN   PORTBbits.RB0
#define BUTTON2_PIN   PORTBbits.RB1
#define BUTTON3_PIN   PORTBbits.RB2

void init_pins() {
    TRISBbits.TRISB0 = 1;
    TRISBbits.TRISB1 = 1;
    TRISBbits.TRISB2 = 1;
}
```

```
int detect_button_press(int button) {
    switch(button) {
        case 1:
            return !BUTTON1_PIN;
        case 2:
            return !BUTTON2_PIN;
        case 3:
            return !BUTTON3_PIN;
        default:
            return 0;
    }
}
```

```
int16_t main(void)
{
    init_pins();
    int loop = 1;
    do{
        while (detect_button_pressed(1)==1)
        {
            LCDGoto(0,0);
            LCDWriteStr("Button 1 = 1");
        }
        while (detect_button_pressed(2)==1)
        {
            LCDGoto(1,0);
            LCDWriteStr("Button 2 = 1");
        }
        while (detect_button_pressed(3)==1)
        {
            LCDGoto(2,0);
            LCDWriteStr("Button 3 = 1");
        }
        while(detect_button_pressed(1)==0&&
                detect_button_pressed(1)==0&&
                detect_button_pressed(1)==0)
        {
            LCDGoto(0,0);
            LCDWriteStr("Button 1 = 0");
            LCDGoto(1,0);
            LCDWriteStr("Button 2 = 0");
            LCDGoto(2,0);
            LCDWriteStr("Button 3 = 0");
        }

      }while(loop==1);

}
```

II.b.2) Programme Boutons 2 (+Appuie de 2s detecter)

```
Keyboard *myKeyboardptr;
Keyboard myKeyboard;
void initCN(void)
{
   CNEN1bits.CN2IE=1;
   CNEN1bits.CN3IE=1;
   CNEN1bits.CN4IE=1;
}
void CNInterruptEnable(void)
{
   IFS0bits.CNIF=0;
   IEC0bits.CNIE=1;
}
```

```
void __attribute__(( __interrupt__ ,__auto_psv__ )) _CNInterrupt(void) {
   if(detect_button_press(1))
   {
      myKeyboardptr->upEdge=1;
      myKeyboardptr->upState=1;
   }
   else if(!detect_button_press(1))
   {
      myKeyboardptr->upEdge=0;
      myKeyboardptr->upState=0;
   }
   if(detect_button_press(2))
   {
      myKeyboardptr->downEdge=1;
      myKeyboardptr->downState=1;
   }
   else if(!detect_button_press(2))
   {
      myKeyboardptr->downEdge=0;
      myKeyboardptr->downState=0;
   }
   if(detect_button_press(3))
   {
      myKeyboardptr->enterEdge=1;
      myKeyboardptr->enterState=1;
   }
   else if(!detect_button_press(3))
   {
      myKeyboardptr->enterEdge=0;
      myKeyboardptr->enterState=0;
   }
   IFS0bits.CNIF=0;
}
#define BUTTON1_PIN   PORTBbits.RB0
#define BUTTON2_PIN   PORTBbits.RB1
#define BUTTON3_PIN   PORTBbits.RB2
void init_pins() {
   TRISBbits.TRISB0 = 1;
   TRISBbits.TRISB1 = 1;
   TRISBbits.TRISB2 = 1;
   CNPU1bits.CN2PUE = 1;
   CNPU1bits.CN3PUE = 1;
   CNPU1bits.CN4PUE = 1;
}
int16_t main(void)
{
   init_pins();
   initTime();
   initCN();
   IFS0bits.CNIF=0;
```

```
   CNInterruptEnable();
   myKeyboardptr=&myKeyboard;
do{
      Time tm7 = getTime();
      while (myKeyboardptr->enterEdge==1)
      {
         LCDGoto(0,0);
         LCDWriteStr("Button 1 = 1");
      }
      while (myKeyboardptr->downEdge==1)
      {
         LCDGoto(1,0);
         LCDWriteStr("Button 2 = 1");
      }
      while (myKeyboardptr->upEdge==1)
      {
         LCDGoto(2,0);
         LCDWriteStr("Button 3 = 1");
         while(isTimeOver(tm7,2000)&&detect_button_press(1)){
            LCDGoto(2,0);
            LCDWriteStr("Button 3 = 2");
         }
      }
      while(myKeyboardptr->downEdge==0&&myKeyboardptr->upEdge==0&&myKeyboardptr-
>enterEdge==0)
      {
         LCDGoto(0,0);
         LCDWriteStr("Button 1 = 0");
         LCDGoto(1,0);
         LCDWriteStr("Button 2 = 0");
         LCDGoto(2,0);
LCDWriteStr("Button 3 = 0");
}
    }while(xxx==1);

   while(1)
   {
   }
}
```

III.d.1) Explication des valeurs dans les structures Menu, MenuItem et MenuValue :

Structure Menu : (référence : Guide de réalisation du projet EEA0601.pdf)

– title représente le titre du menu. Il est insrit sur la première ligne entre crochets et ne peut se déplacer. Si le pointeur associé est nul, il n'y a pas de titre au menu et les trois lignes de l'afficheur sont alors utilisées pour présenter les items de menu

.

– root est un pointeur vers le menu parent ou celui vers lequel se brancher après appui sur la touche retour.

– items pointe sur le tableau des items du menu. Ils sont affichés à partir de la seconde ligne de l'écran (si un titre est présent) et dans leur ordre d'apparition. Les items non visibles peuvent être affichés à l'aide des touches + et - par scrolling. Par exemple, l'item d'indice 2 du menu [Reglage PID] n'est initialement pas visible. Un premier appui sur sur - déplace le curseur de sélection > devant l'item 1 (I: 000) puis un second appui remplace l'item d'indice 0 par celui d'indice 1 et l'item d'indice 1 (P: 000) par celui d'indice 2 (D: 000).

– offset mémorise le décalage vertical compté algébriquement (positif vers le haut) de l'item d'indice 0 par rapport à l'affichage initial du menu. Ainsi, suite aux opérations décrites ci- dessus, l'offset du menu vaut 1 .

– selected indique la ligne de l'élément sélectionné : dans l'exemple précédent, selected vaut 2.

Structure MenuItems :

– label est la chaine de caractère affichée à l'écran pour cette entrée du menu ;

– action est un pointeur dont la signification dépend de la valeur du champ type ;

– type peut prendre les valeurs suivantes : 0 : selectable, 1 : function, 2 : submenu ou 3:value. selectable désigne une entrée commutable par appui sur la touche entrée. Lors- qu'une telle entrée est active, elle est précédée d'une étoile (*) et d'une espace dans le cas contraire. function indique qu'action est un pointeur vers une fonction à exécuter sur appui de la touche entrée. submenu indique qu'action est un sous-menu à afficher sur appui de la touche entrée, enfin value indique une valeur numérique modifiable et dans ce cas, action pointe vers une structure de type MenuValue décrite ci-dessous.

– checked indique soit un item sélectionné (type selectable) ou une action engagée (type action).

Structure MenuValues:

• 0:direct : ce paramétrage indique une variable à évolution immédiate , i.e. dont la grandeur pilotée change instantanément lors du réglage ;

• 1:batch : ce paramétrage indique une variable dont l'évolution ne sera prise en compte qu'après appui sur la touche entrée.

tuningValue correspond à la valeur temporaire de la grandeur pilotée (mode batch). tunedValue est la variable abritant la grandeur pilotée. min et max indiquent respectivement les valeurs minimales et maximales admissibles sur la grandeur pilotée.

III.d.2) Code complete de l'initialisation des structures : Fichier fonctions_sup.h

```
#ifndef FONCTIONS_SUP_H
#define FONCTIONS_SUP_H
// forward declaration
typedef struct menu Menu;
typedef struct menuitem MenuItem;
typedef struct menuvalue MenuValue;
typedef struct{
   int upState:2;
   int downState:2;
   int enterState:2;
   int upEdge:2;
   int downEdge:2;
   int enterEdge:2;
}Keyboard;
struct menu{
   char *title;            //titre du menu
   Menu *root;       //menu precedent
   MenuItem *items;   //option du menu specifique
   short offset:6;          //decalage verticale de la selection
   short selected:4;        //indique l'option selectionner
};
struct menuitem{
   char *label;       //chaine de char pour l'option selectionner
   void *action;      //depend de type (si type=1 on execute une fonction/si 2 affiche submenu/si 3 on pointe vers MenuValue)
   short type:2;      //0=selectable, 1=action, 2=submenu, 3=value
   short checked:1;   //0=selectable, 1=action->(action, submenu, value)
};
struct menuvalue{
   int tuningValue;
   int *tunedValue;
   int min;
   int max;
   short type;       //0 la valeu change lors du changement, 1 la valeur change apres appuie sur entree
};
void menu_principale(Keyboard *keyboard);
int detect_button_press(int button);
void delay_en_s(float time);
void _wait10mus(unsigned int tenmus);
#endif
```

Fichier fonctions_sup.c:

```
int gauche=0,droite=0,regP=0,regI=0,regD=0,distObs=0,maxG=0,maxD=0;
int *distObsptr=&distObs,*regPptr=&regP,*regIptr=&regP,*regDptr=&regD,*maxGptr=&maxG,*maxDptr=&maxD;
Menu rootMenu,actionMenu,paramMenu,affetatMenu,manuMenu,detecMenu,PIDMenu,vitMenu;
void menu_principale(Keyboard *keyboard)
{
MenuValue manuValue[]={{0,0,0,100,0},{0,0,0,100,0}};
   MenuValue distValue[]={{0,0,0,100,1}};
   MenuValue pidValue[]={{0,0,0,100,1},{0,0,0,100,1},{0,0,0,100,1}};
   MenuValue viteValue[]={{0,0,0,100,1},{0,0,0,100,1}};
   // déclarer les items
   MenuItem rootItems []={{ACTIONS,&actionMenu,2,0},{PARAMETRES,&paramMenu,2,0},{AFFETAT,&affetatMenu,2,0}};
   MenuItem                                                                                    actionItems
[]={{MALENTE,NULL,1,1},{MARAPIDE,NULL,1,1},{MOMANUEL,&manuMenu,2,0},{RETOUR,&rootMenu,2,0}};
   MenuItem                                                                                    paramItems
[]={{DEOBST,&detecMenu,2,0},{REGPID,&PIDMenu,2,0},{VITESSE,&vitMenu,2,0},{RETOUR,&rootMenu,2,0}};
   MenuItem manuItems []={{GAUCHE,manuValue,3,0},{DROITE,manuValue,3,0},{RETOUR,&actionMenu,2,0}};
   MenuItem detObsItems[]={{"Dist: ",distValue,3,0},{RETOUR,&paramMenu,2,0}};
   MenuItem PIDItems[]={{"P: ",pidValue,3,0},{"I: ",pidValue,3,0},{"D: ",pidValue,3,0},{RETOUR,&paramMenu,2,0}};
   MenuItem vitItems[]={{"Rapide: ",viteValue,3,0},{"Lent: ",viteValue,3,0},{RETOUR,&paramMenu,2,0}};
   // déclarer les menus
   //Menu Principale
   rootMenu.title = NULL;
   rootMenu.root =NULL;
   rootMenu.items = rootItems;
   rootMenu.selected = 0;
   rootMenu.offset = 0;
   //Menu Action
   actionMenu.title = NULL;
   actionMenu.root = &rootMenu;
   actionMenu.items = actionItems;
   actionMenu.selected = 0;
   actionMenu.offset = 0;
   //Menu Parametres
   paramMenu.title = NULL;
   paramMenu.root = &rootMenu;
   paramMenu.items = paramItems;
   paramMenu.selected = 0;
   paramMenu.offset = 0;
   //Menu Affiche Etat
   affetatMenu.title = NULL;
   affetatMenu.root = &rootMenu;
   //affetatMenu.items = ;
   affetatMenu.selected = 0;
   affetatMenu.offset = 0;
   //Menu Mode Manuel
   manuMenu.title = "[Manuel]";
   manuMenu.root = &actionMenu;
   manuMenu.items = manuItems;
   manuMenu.selected = 0;
   manuMenu.offset = 0;
   //Menu Detection Obstacles
   detecMenu.title = "[Detec. Obs.]";
   detecMenu.root = &paramMenu;
   detecMenu.items = detObsItems;
   detecMenu.selected = 0;
   detecMenu.offset = 0;
   //Menu PID
   PIDMenu.title = "[Reglage PID]";
   PIDMenu.root = &paramMenu;
   PIDMenu.items = PIDItems;
```

```
    PIDMenu.selected = 0;
    PIDMenu.offset = 0;
    //Menu Vitesse
    vitMenu.title = "[Vitesse]";
    vitMenu.root = &paramMenu;
    vitMenu.items = vitItems;
    vitMenu.selected = 0;
    vitMenu.offset = 0;
}
```

III.d.3) Fichier fonction_sup.c

III.d.4) Fichier fonction_sup.c :

https://github.com/christopherdacc/Projet_EEA0601/blob/main/Programmation/modele.X/lib/fonctions_sup.c