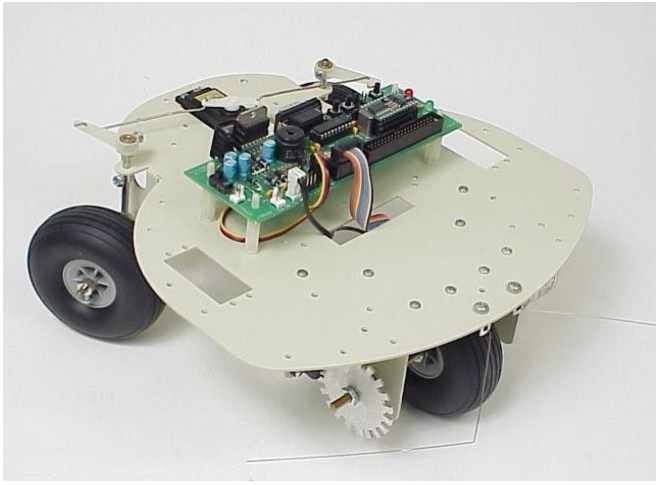
Université François-Rabelais de Tours
Institut Universitaire de Technologie de Tours
Département Génie Électrique et Informatique Industrielle

UNIVERSITE FRANCOIS-RABELAIS
TOURS
Institut Universitaire de Technologie
Département
GENIE ELECTRIQUE ET
INFORMATIQUE INDUSTRIELLE

ROBOT EXPLORATEUR



Source : http://www.robotshop.ca/robot-mobile-arobot-arrick-bs2-2.html

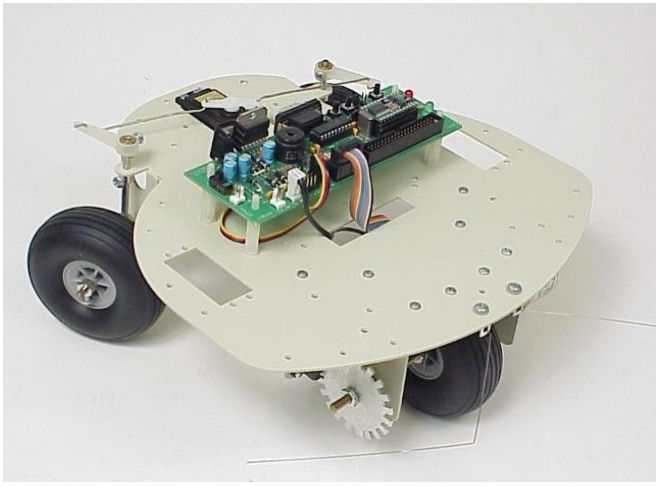
Hugo KARPINSKI, Sébastien SIROT
Année 2010 – Groupe P2
Promotion 2008-2010

Enseignants
Mme Véronique AUGER
M. Thierry LEQUEU

Université François-Rabelais de Tours
Institut Universitaire de Technologie de Tours
Département Génie Électrique et Informatique Industrielle

UNIVERSITE FRANCOIS-RABELAIS
TOURS
Institut Universitaire de Technologie
Département
GENIE ELECTRIQUE ET
INFORMATIQUE INDUSTRIELLE

ROBOT EXPLORATEUR



Source : http://www.robotshop.ca/robot-mobile-arobot-arrick-bs2-2.html

Hugo KARPINSKI, Sébastien SIROT
Année 2010 – Groupe P2
Promotion 2008-2010

Enseignants
Mme Véronique AUGER
M. Thierry LEQUEU

Sommaire

Introduction

Dans le cadre du quatrième semestre de notre formation, nous avons entrepris la réalisation d'un projet alliant les domaines de l'électronique et de l'informatique.

Nous avons choisi un nouveau projet : la réalisation d'un petit robot autonome.

Lors de la rencontre d'un obstacle, il devra se repositionner pour trouver une autre direction.

Nous utiliserons un micro-contrôleur PIC de Microchip. qui nous permettra de nous familiariser avec cette technologie, un montage en pont pour la partie puissance électrique et plusieurs types de capteurs pour éviter les obstacles ou suivre une ligne. Nos typons seront effectués avec l'aide de WinCircuit2008.

Ce projet nous permettra d'améliorer nos connaissances en électronique et d'apprendre les techniques de programmation des micro-contrôleurs qui pourront nous être utiles dans un avenir proche : lors de nos poursuites d'études ou pendant notre stage.

1. Cahier des charges

Matériel à notre disposition :

- 2 moteurs Mabuchi ;
- 2 hacheurs en pont 4 cadrans pour la commande du moteur ;
- Alimentation par batterie plomb ou par coupleur de piles ;
- boutons poussoir pour détecter les chocs ;
- 1 bouton marche arrêt ;
- 1 micro-contrôleur PIC 18f46k20 ;
- 1 programmateur PIC KIT 3 de Microchip ;
- capteurs de luminosité ;
- 3 régulateur LM317 de Fairchild ;
- 1 transducteur ;

Le projet est venu de la volonté des étudiants de réaliser un robot. Sa fonction principale est de pouvoir repartir dans une direction après avoir percuté un obstacle.

Les contraintes sont de deux types :

- le temps : 11 semaines entre le début du projet et la rédaction du rapport final.
- Les moyens financiers : le projet doit être réalisé à un coût limité.

2. Coût du projet

Désignation	Prix u. TTC (en euros)	Quantité	Prix TTC (en euros)
LED blanches	0,2	5	1
LED rouges	0,1	2	0,1
LED vertes	0,2	1	0,2
Résistances ¼ W	0,04	25	1
Mini boutons poussoirs tête métal	0,4	6	2,4
Boutons poussoirs rouges/noirs	0,1	2	0,2
Résistances variables	0,1	5	0,5
AOP LM324	0,2	4	0,8
Régulateurs LM 317	0,4	3	1,2
LCD 2x16	4	1	4
Moteurs Mabuchi	4	2	8
Radiateurs	0,4	3	1,2
Ponts en H TLE-5206S	4,5	2	9
Supports 40 pins	0,3	1	0,3
PIC18F46K20	4	1	4
Diodes IR	0,35	2	0,7
Photo-diodes IR	0,65	2	1,3
Interrupteurs IR	0,85	1	0,85
Photo-résistances	0,9	7	6,3
		Total TTC	38,35 euros

3. Recherche documentaire

La recherche documentaire a commencé pendant les premières semaines (jusqu'aux vacances de février) par la traduction et la compréhension générale du programmateur PICKIT3. Nous avions à notre disposition un tutoriel de 12 leçons fourni avec notre programmateur. Nous avons suivi le fil de ces leçons, ce qui nous a permis de réaliser des fonctions élémentaires : allumer une diode, la faire clignoter, maîtriser les interruptions... Cette période nous a permis également de comprendre le fonctionnement interne du micro-contrôleur : allocation de la mémoire, jeu d'instructions. Pour réaliser la partie électronique nous avons consulté de nombreux documents techniques ainsi que certains rapports des années précédentes, notamment sur un robot suiveur de ligne.

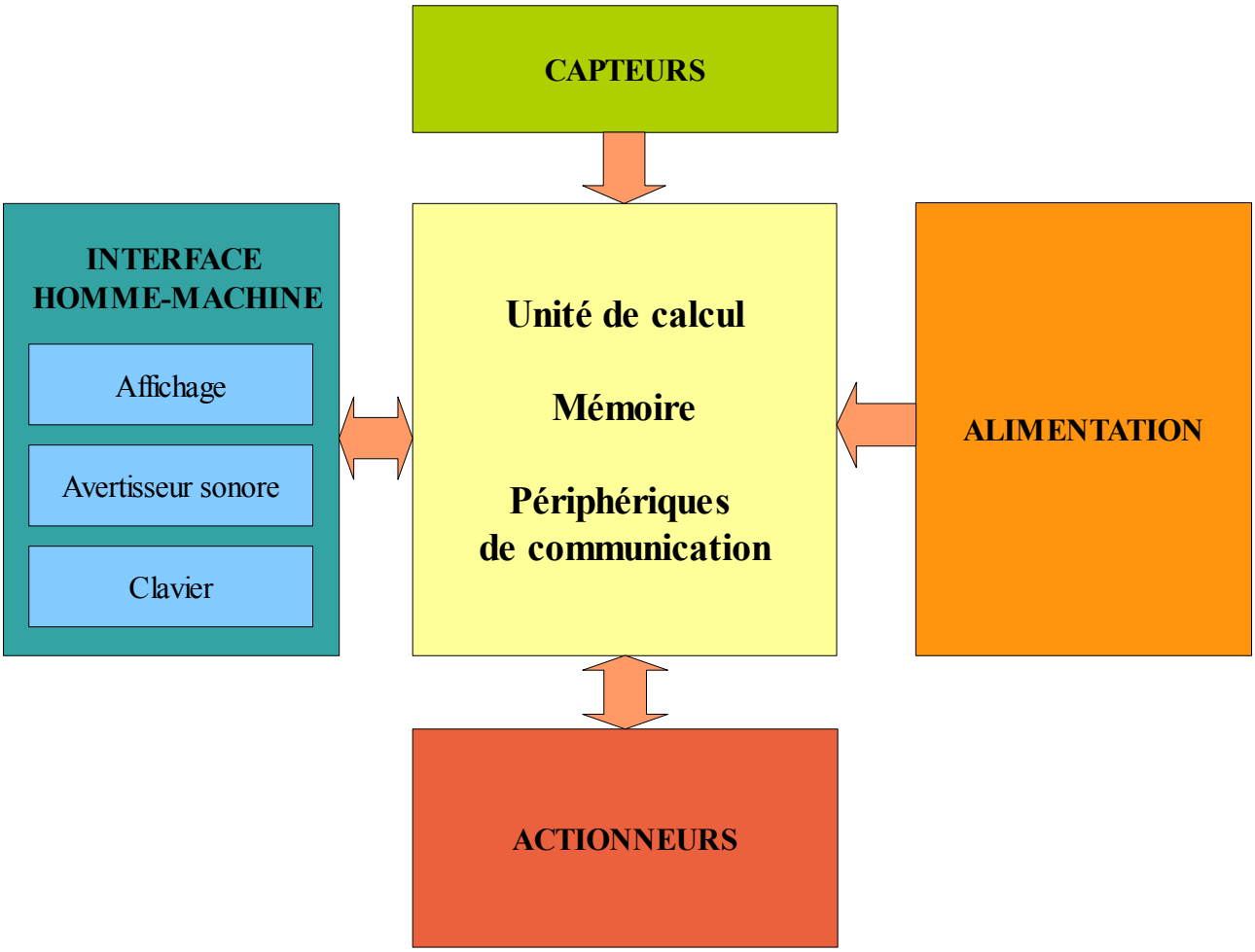
Pendant la deuxième phase de réalisation du projet, nous nous sommes intéressés à la partie électronique du robot. Nous avons décider de décomposer notre système en plusieurs modules afin de pouvoir réutiliser ces cartes ultérieurement. Il nous faudra concevoir 7 cartes :

- étage de puissance ;
- carte micro-contrôleur ;
- interface ;
- avertisseur sonore ;
- contrôle moteur.(2 cartes) ;
- carte capteur ;
- étude du LM317 de FAIRCHILD qui permet de réguler les tensions(3,3 V continue) auxquelles sont soumises les composants électroniques : AOP, micro-contrôleur.

4. Analyse technique du projet

Nous avons choisi un micro-contrôleur PIC[1], d'une part, parce que nous possédions déjà un programmateur PICKIT 3 et, d'autre part, parce que la documentation le concernant est complète. De plus, nous avons un tutoriel de 12 leçons, très complet, qui nous est proposé par son constructeur.

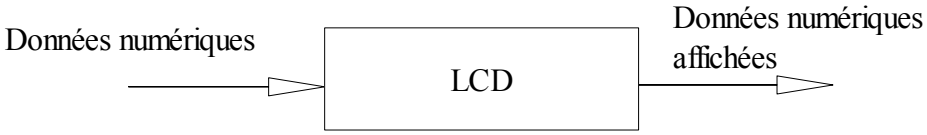
Chaque carte a été réalisé en tant que « module » afin de pouvoir être réutilisées plus tard pour d'autres projets. Le coupleur de pile a été choisi pour sa simplicité et son encombrement réduit. Les régulateurs permettent d'obtenir des tensions parfaitement lisses et au voltage désiré.



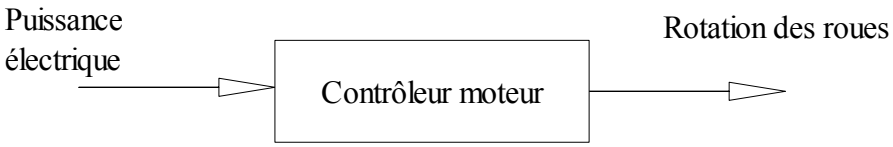
Dessin 1: schéma général de notre système

1 Les micro-contrôleurs nommés « PIC » sont fabriqués par la société Microchip

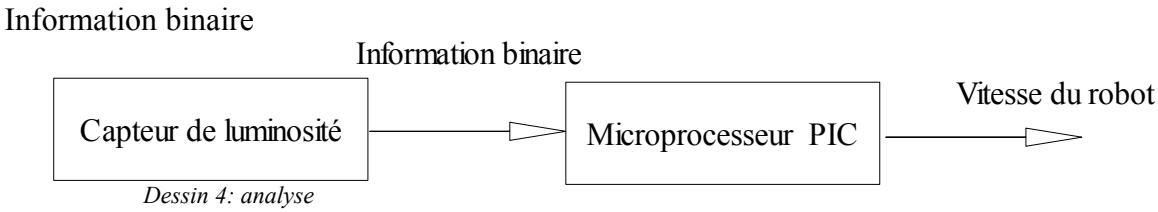
5. Les différentes fonctions du robot se décomposent ainsi (analyse fonctionnelle de niveau 2) :



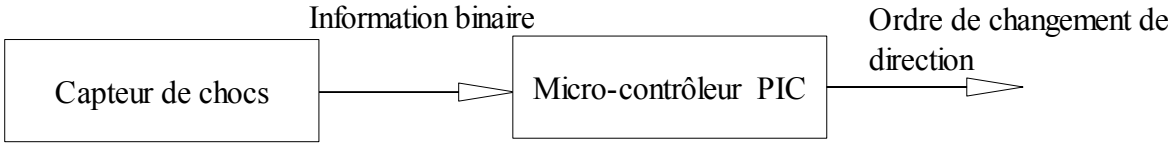
Dessin 2: analyse fonctionnelle de niveau 2 - écran LCD



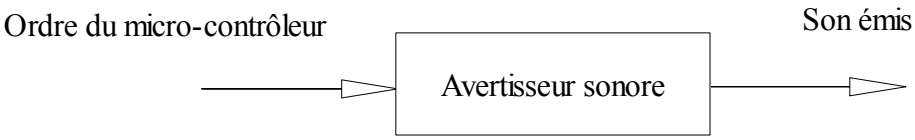
Dessin 3: analyse fonctionnelle de niveau 2 - contrôleur moteur



Dessin 4: analyse fonctionnelle de niveau 2 - capteur de luminosité



Dessin 5: analyse fonctionnelle de niveau 2 - capteur de chocs



Dessin 6: analyse fonctionnelle de niveau 2 - avertisseur sonore

6. Prise en main du 18F45K20 et du pickit 3

Il a fallu dans un premier temps comprendre comment est programmé un micro-contrôleur et comment il fonctionne (structure, architecture de la mémoire). Cette partie nous a pris beaucoup de temps, car il a fallu lire la documentation en anglais, fournie par Microchip. Elle explique des éléments assez complexes comme l'architecture Harvard. Nous avons effectué la programmation en langage C, un langage que nous avons appris lors de nos deux premiers semestres à l'IUT.

Nous allons maintenant vous présenter un résumé du contenu des 12 leçons du tutoriel de Microchip.

6.1. Organisation de la mémoire

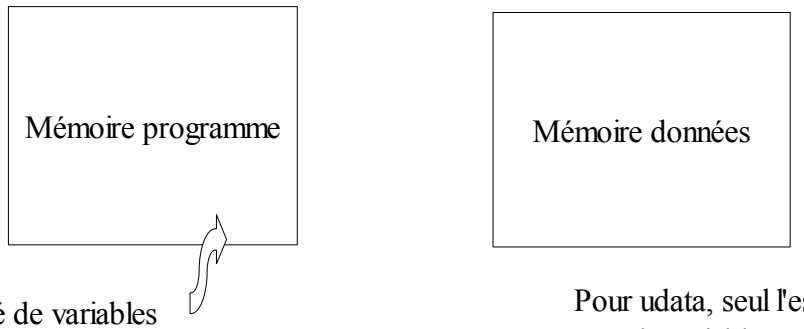
Les PIC 18FXXXX ont une architecture Harvard qui alloue trois espaces distincts : un espace pour le programme, un espace pour manipuler rapidement les données (RAM) et un espace pour sauvegarder des données (EPROM). Ces espaces sont séparés aussi physiquement et chaque espace à son propre bus.

6.1.1. La mémoire du programme (« memory program)

L'espace de la mémoire dédié au programme est adressé par un programme compteur PC de 21 bits permettant jusqu'à 2 MB de mémoire. Au reset, PC= 0, les vecteurs sont déclarés aux adresses 0x000008 à 0x000016. Habituellement une instruction Goto est mise à l'adresse 0 pour éviter les vecteurs d'interruption. Il existe également dans certains PIC une option qui protège le code.

6.1.2. La mémoire des données (« data memory »)

Nous avons à notre disposition 4 MB de RAM partagés par séquence de 8 bits. Au démarrage, les variables ont des valeurs aléatoires, les données sont organisées en banques de 256 bits ce qui nécessite que la banque (les 4 bits de poids forts de l'adresse du registre) soit sélectionnée à l'aide du Bank Select Register(BSR). Les espaces spéciaux (access RAM en anglais) peuvent être adressés sans s'occuper du BSR.



Possibilité de variables à valeurs définies (idata) conservées dans la mémoire programme puis transférées dans la mémoire des données au démarrage.

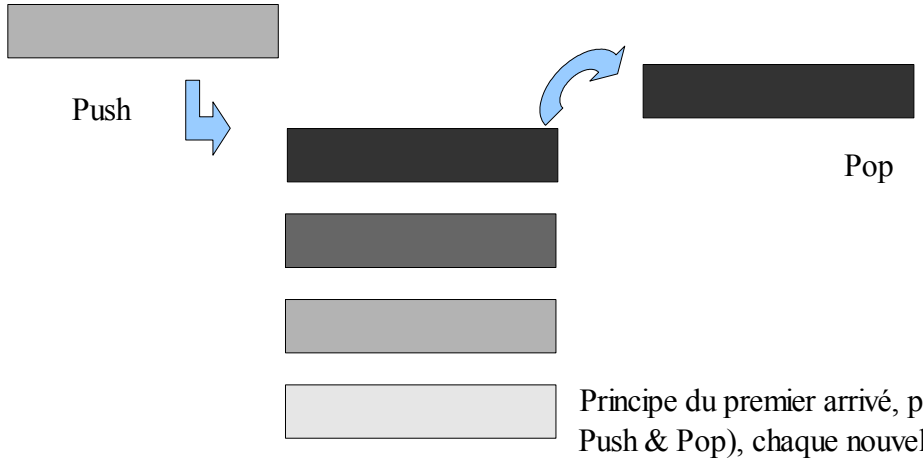
Pour udata, seul l'espace utilisé pour la variable est réservé à l'adresse choisie. udata est crée dans la mémoire programme puis transféré dans la RAM (Random Access Memory).

Dessin 7: accès mémoire

6.1.3. Registres de fonctions spéciales (« special function registers SFRs »)

Les SFRs sont des registres du cœur du processeur (tels que Stack Pointer, STATUS Register et Program Counter). Les SFRs permettent de régler les périphériques du micro-contrôleur (à la banque[2] 15). Grâce au compilateur MPLAB C, nous pouvons accéder à ces registres par un nom.

6.1.4. Principe de sauvegarde des adresses (« Return adress Stack »)



Principe du premier arrivé, premier parti(en anglais Push & Pop), chaque nouvelle adresse est stockée puis immédiatement supprimée dès qu'une autre adresse arrive, il n'est alors plus nécessaire de conserver son adresse.

Dessin 8: la pile d'adresse

6.2. Actions sur une diode électroluminescente

Avant de commencer, nous créons un nouveau projet dans MPLAB C et intégrons un fichier linker (.lkr) qui permet au compilateur de relier les différents fichiers intégrés au programme et des fichiers .h qui contiennent des fichiers déjà existants. Pour coder, il est souvent nécessaire de reprendre des fonctions déjà créées par d'autres afin de gagner du temps au début du programme. Nous utilisons l'instruction #include suivi du nom du fichiers entre guillemets. Quant à la commande # pragma config, elle nous permet de configurer comme nous le souhaitons le micro-contrôleur (règle les deux bits de configuration). Pour configurer l'horloge qui détermine la fréquence à laquelle va clignoter la led, nous rentrerons l'instruction # pragma config FOSC = INTIO67, ce qui veut dire que la fréquence utilisée sera celle de l'horloge interne du micro-contrôleur. Si nous voulons allumer une led par exemple il faut configurer les entrées/sorties à l'aide de la fonction Trisd, nous affectons à TRISD la valeur en binaire qui correspond aux entrées et sorties que nous désirons utiliser sur la carte et mettre à 0 un bit afin d'obtenir une sortie (ex : TRISD = 0b1111110 met les broches 1 à 6 en entrées et la broche 7 en sortie) .

Dans un second temps, on utilise une autre fonction qui s'utilise d'après la commande suivante LATDbits.LATD7 =1 qui permet d'appliquer une tension constante sur la sortie 7 et donc de la mettre à 1, ce qui fournira du courant à la led et l'allumera. Si nous désirons voir la led s'allumer, il faut insérer l'ensemble des instructions dans une boucle infinie réalisée à l'aide de la commande while(1), 1 représentant la condition toujours vraie quelque soit l'entrée.

2 Banque 15 : la mémoire du micro-contrôleur est séparée en banque.

Pour faire clignoter la led, il faut seulement ajouter la commande Delay1KTCYx(50). La fonction delay se trouve dans la bibliothèque delays.h inclut au début du fichier (les autres bibliothèques utilisées en C sont aussi accessibles telles que STDIO, STDLIB, STRINGS). Delay1KTCYx(50) permet de laisser allumer la led durant 200 ms : les 1K signifient que le micro-contrôleur effectue 1000 cycles internes(un cycle est défini comme 4 temps de quartz) et le 50 multiplie le nombre de cycles donc il s'écoulera successivement 200 ms pendant chaque état de la led : éteinte ou allumée car la fréquence est celle qui est interne au micro-contrôleur soit 1MHz.

Si nous voulons réaliser un chenillard avec plusieurs led, nous sommes contraint à cause de l'architecture Harvard du PIC de déclarer deux nouveaux types de variables : udata et idata (idata est initialisé et udata permet de réserver un espace mémoire), qui sont accessibles grâce aux directives préprocesseurs #pragma udata et #pragma romdata. Il faut réserver un espace mémoire dans la ROM(read only memory : mémoire morte), qui est réalisé en une ligne par adresse(#pragma romdata table = 0x180. Pour utiliser les différentes led, nous les rangeons dans un tableau, et nous déclarons l'adresse de chacune des led en hexadécimal (par exemple const rom unsigned char Led_adress[8] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08}. Il faut modifier la fonction TRISD afin de posséder plusieurs sorties.

6.3. Gestion des interrupteurs(« Switch input »)

Pour pouvoir les utiliser, il faut ajouter le fichier 04SwitchInput.h et définir la broche utilisée grâce à #define Switch_Pin PORTBits.RB0 et la manière de détection à l'aide de la directive # include DetectsInArrow 5.

Une variable locale est placée sur la pile logicielle (crée lorsque nous rentrons dans la fonction) puis détruite lorsque l'on en sort (principe du Push & Pop, vu dans actions sur les diodes électroluminescentes). Le bouton de la carte de démonstration est alors connecté à la broche RB0 et mise au potentiel VDD. Quand le bouton est pressé : VDD= 0. PORTBits.RB0 donne la valeur sur la broche RB0. Les broches entrées/sorties qui partagent un canal d'entrée analogique doivent être configurées comme des broches digitales si elles sont utilisées ainsi, sinon leurs valeurs restent toujours nulles.(Il faut utiliser SFR ANSELH).

Voici des instructions issues du tutoriel :

INTCON2bits.RBPU=0 ; //[3] active les pull-up[4] internes sur le port B.

Wpubits.WPUB0=1 ; // active le pull-up en RB0.

ANSELH= 0x00 ; // les entrées analogiques 8 à 12 sont numériques.

TRISBbits.TRISB0=1 ; fixe l'état du port B0 à 1.

Grâce à ses quatre lignes, le bouton est opérationnel pour le micro-contrôleur.

Utilisation de Timer0

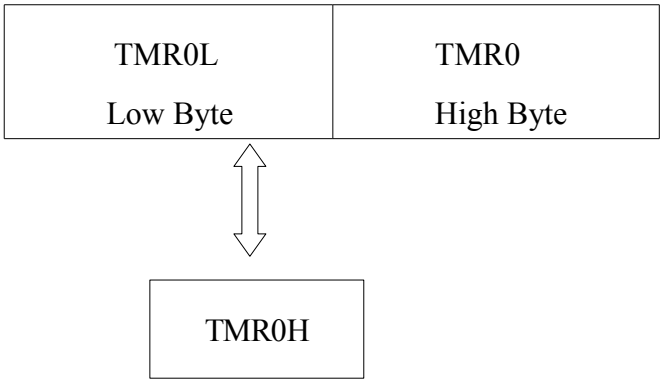
Timer0 est un compteur physique interne au micro-contrôleur qui compte les cycles ou les évènements externes. Il permet de libérer le micro-contrôleur pour d'autres tâches que celles de compter des cycles. Un « prescaler » permet de définir le nombre de cycles ou d'évènements

3 Désigne les commentaires, ce qui suit // n'est pas interprété par le compilateur.
4 Résistances de pull-up permet d'éviter les états indéterminés.

externes requis pour incrémenter le timer de 1. Timer0 compte les évènements externes sur T0CKI, sur 8 bits (O à 255) ou 16 bits (0 à 65535). Le drapeau(« flag ») est actif quand la valeur repasse à 0. Le « prescaler » permet de définir le nombre de cycles nécessaires pour exécuter une instruction(temps de cycle = temps d'horloge/4). Le « prescaler » peut être régler à 2,4,8,16,32,...,256.

INTCON TMR0IF BIT // le flag du timer0 doit être remis à 0 par la programmation.

La valeur du timer lorsqu'elle est sur 16 bits est contenue dans le SFR suivant :



Le fonctionnement du timer est contrôlé par le SFR T0CON qui est un registre de contrôle et qui sert à configurer le timer.

Il est possible aussi de créer de nouveaux types de variables si nous avons besoin de propriétés spécifiques. Voici deux exemples :

typedef enum {LEFT2RIGHT, RIGHT2LEFT} LEDDirections ;

typedef enum {FALSE, TRUE} BOOL ; // ceci crée un type de variable s'appelant bool et pouvant prendre 2 valeurs.

Pour utiliser ces types crées, nous faisons comme ceci : LEDDirections Direction = LEFT2RIGHT ;

Il faut cependant être prudent lorsque nous manipulons les SF. Utiliser certains SFR codés sur plusieurs octets(pour accéder à un bit) peut affecter les autres bits. Modifier un bit peut modifier la valeur d'autres bits. Le bit est lu, une valeur est obtenue, le bit lu a été malheureusement modifié, le micro-contrôleur réécrit la valeur initiale (la valeur lue) ce qui conserve l'intégrité des données (fonctionnement interne du micro-contrôleur sur lequel nous ne pouvons interagir.

6.4. Utiliser le déboguage de PicKit 3 express

Nous allons voir dans cette partie comment se servir du programmateur pickit en tant « in-circuit-debugger » (ICD).

Les points clés de cette leçon sont les suivants :

- un Icd se sert de ressources internes au composant qui sont spécifiques au débogage. Ces espaces réservés dans la mémoire programme et les « reserved file registers « sont marqués d'un 'R' dans MPLAB IDE mais reste inaccessibles.
- Le débogueur utilise : un niveau de « hardware return adress stack » et deux broches d'entrées/sorties du composant.
- Le nombre de points d'arrêt (ou *breakpoint*[5]) disponibles dépend du modèle du PIC utilisé.Ressources réservées pour le débogage

Les broches MCLR complémenté, PGD et PGC sont utilisées par l'ICD (leurs autres fonctions ne sont pas accessibles lors du débogage). Un niveau de pile est utilisé par l'ICD et donc réservé.

Un script 18f45k20_i.lkr doit être utilisé pour le débogage.

Opérations de base

Les opérations de bases servent à commander et tester le fonctionnement du PIC18F45K20, elles sont exécutées dans MPLAB.

F5 HALT : pour arrêter à tout moment l'exécution du programme.

F7 STEP.INTO : active l'avance pas à pas, lorsque nous arrivons à un appel de fonction, au prochain appui sur STEP.INTO, on rentre dans la dite fonction.

F8 STEP.OVER : active un avance pas à pas aussi mais nous exécutons les appels de fonction sans rentrer à l'intérieur.

F8 STEP.OUT : termine l'exécution de la fonction dans laquelle nous sommes et nous renvoie sur la première instruction qui se trouve après la fonction que nous quittons.

F9 RUN : active l'exécution du code jusqu'au prochain HALT de l'utilisateur ou jusqu'à la rencontre d'un breakpoint.

F6 RESET : permet le redémarrage du micro-contrôleur à partir du début du code programme. Disponible uniquement si HALT auparavant. RESET ouvre le fichier c018i.c qui est le code de démarrage appartenant à la librairie MPLAB.C. Cette librairie initialise les piles logicielles C (« the C software stack »), assigne les données valeurs aux variables initialisées, et saute au début du programme principal (main).

6.5. Utilisation des « breakpoint »

Après avoir sélectionné une ligne, il suffit de sélectionner dans le menu contextuel Set Breakpoint (un 'B' apparaît en marge de la ligne).

F8 (ou F7) peut être utilisé pour passer le breakpoint et reprendre l'exécution du programme. Le nombre de breakpoint dépend du micro-contrôleur.

Pour accéder à la boîte de dialogue des breakpoints :

->menu debugger → Breakpoints

« The Silicon debug Ressource toolbar informe du nombre total de breakpoint disponibles pour le µC[6](« HW BP ») et du nombre de breakpoints utilisés(« used »).

5 Un break point est un repère qui permet de signaler un endroit important du programme.
6. « µC » signifie « micro-contrôleur »

Le PIC 18f45k20 a jusqu'à 3 breakpoint disponibles.

Attention. Le nombre de breakpoints actifs peut affecter l'utilisation de « STEP Into » et « Step Over ». En effet, quand ces fonctions sont utilisées, un breakpoint est mis à l'étape suivante, celle que nous voulons « sauter ». Si tous les breakpoints sont utilisés, ces fonctions ne pourront pas mettre un breakpoint sur l'instruction suivante et donc ne fonctionnent pas comme espéré. Pour s'en servir, il faudra donc libérer des breakpoints au préalable.

6.6. Regarder des variables et des SFR (Special Function Register)

Toutes les valeurs des « File Registers » et des SFR peuvent être visualisées en ouvrant View-> File Registers et View → Special Function Register.

Toutefois, garder ces 2 fenêtres ouvertes en permanence n'est pas recommandé, car elles imposent la lecture de toutes ces données. Lire toutes ces données via le bus de l'ICD prend un temps significatif(dépend de la vitesse de l'oscillateur).

On préfèrera utiliser View->Watch à la place.

Utiliser le convertisseur analogique-numérique (Can ou ADC en anglais pour analog to digital)

Afin de contrôler et configurer le convertisseur analogique numérique, il faut écrire dans les registres suivants : ANSEL0, ANSELH, ADCON0, ADCON1, ADCON2.

Configuration et exécution du CAN

Voici les différentes étapes à respecter pour se servir de l'ADC :

1. Configurer la broche RA0/AN0 comme une entrée analogique(registre ANSEL).
2. Indiquer les tensions de référence pour la conversion (registreADCON1).
3. L'horloge doit être réglée pour devenir la plus petite possible mais doit rester toutefois supérieure à la période du TAD minimal. Le temps minimal du « TAD » pour le PIC 18f45k20 est d'après la datasheet de 1,4µs(on règle ADCSX = '000').
4. Sélectionner le canal/voie et activer l'ADC dans ADCON2.
5. Commence la conversion dans ADCON0.

Précisions sur l'étape 3 : Les bits de l'ACTQx déterminent le temps d'acquisition, et devraient prendre en compte Tacq, le temps interne d'acquisition du CAN (paramètre 132, plus de précision dans la documentation du PIC), et le temps de stabilisation du circuit connecté sur la broche du CAN(composé d'un dipôle RC formé par le potentiomètre et C3). Ce dipôle a un temps de stabilisation très long, nous règlerons pour cet exemple ACTQx à la valeur maximale '111' ce qui correspond à 20 TAD soit 20 x 2µs = 40 µs.

Pour la justification et le formatage, nous choisirons ADFM=0 ce qui dispose un résultat justifié à gauche. Ceci rend plus facile la récupération des 8 bits les plus significatifs du résultat de la conversion via ADRESH.(dans notre exemple ADCON 2 = 0b00111000).

Précisions sur l'étape 4 : Le potentiomètre présent sur la carte du PICKIT3 est relié à l'AN0, donc le canal 0 est sélectionné dans ADCON0. Le bit ADON est mis à 1 pour activer le

périphérique CAN. Le bit GO/DONE (DONE est complémenté) est laissé à 0 car la conversion ne sera pas démarrée tout de suite.

Précision sur l'étape 5 : Pour commencer la conversion : GO/DONE (DONE est complémenté) est mis à 1.

Quand la conversion est terminée GO/DONE (DONE est complémenté) repasse automatiquement à 0 et le résultat de la conversion est lisible dans ADRESH et ADRESL.

Il y a trois fonctions à déclarer dans le registre :

→ void Timer0_init(void)

→ void ADC_init(void)

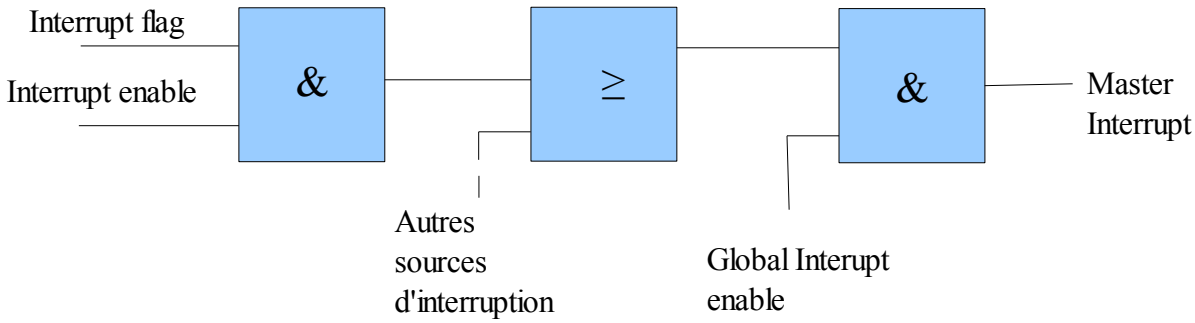
Ces deux fonctions initialisent les périphériques Timer0 et le CAN.

6.7. Gérer les interruptions

- Une interruption est un événement physique qui interrompt le code programme pour exécuter une fonction spéciale. Quand cette fonction s'achève, l'exécution du programme reprend là où elle s'était arrêtée.
- Les PIC18FXXXX supportent un ou deux niveaux de priorité d'interruption.
- Une interruption de haute priorité peut interrompre une interruption de priorité basse aussi bien que le programme principal.
- Pour définir les fonctions d'interruption, nous utilisons les directives :

  # pragma interrupt low et #pragma interrupt.

Architecture des interruptions des PIC18FXXXX



Quand une interruption survient, le PIC finit d'exécuter l'instruction en cours, stocke la prochaine adresse du programme principal dans « return adress stack » et saute jusqu'au vecteur d'interruption. A ce vecteur d'interruption, il commence l'exécution de la fonction désignée comme étant la routine d'interruption. Quand nous sortons de la fonction (à la fin de celle-ci), l'exécution du programme principal reprend à partir de l'adresse sauvegardée dans « return adress stack »(pile d'adresse).

Les interruptions permettent aux évènements hardware[7] d'agir très rapidement , ils causent l'exécution immédiate du code dédié. Le vecteur de l'interruption à haute priorité est à l'adresse de

7 Hardware : matériel physique informatique : circuit imprimé ou périphériques externes.

la mémoire programme 0x0008. Quant à l'interruption de basse priorité, elle est à l'adresse de la mémoire programme 0x0018. Si les priorités d'interruption ne sont pas utilisées, toute interruption « saute » au vecteur à 0x0008.

A présent, la variable Direction est globale pour pouvoir être accessible à la routine d'interruption.

Quand on utilise les interruptions, les vecteur d'interruption doivent être définis et placés aux adresses appropriées en utilisant les directives # pragma adéquates. Une instruction assembleur GOTO redirige s'exécution du programme vers les fonctions d'interruption dont les noms servent d'argument à l'instruction asm GOTO.

Code exemple :

```
#pragma code InterruptVectorHigh = 0x08
void InterruptVectorHigh(void)
{_asm                          // début du code assembleur.
   goto InterruptServiceHigh
   endasm}                     // fin du code assembleur.
#pragma code InterruptVectorLow = 0x18
void Interrupt […]
```

Ensuite, les fonctions d'interruption sont déclarées avec les directives #pragma interrupt et #pragma interruptlow.

```
#pragma Interrupt InterruptServiceHigh
void InterruptServiceHigh(void)
{
…}
#pragma Interrupt InterruptServiceLow
void InterruptServiceLow(void)
{
…}
```

Attention : comme toutes les interruptions de même priorité appellent la même fonction d'interruption, il est nécessaire de regarder dans cette fonction quel périphérique a déclenché cette interruption. Une fois le drapeau d'interruption trouvé (« flag interrupt » en anglais), le logiciel(software) doit remettre à 0 le flag pour ré-enclencher l'interruption.

6.8. Utiliser l'EEPROM[8] (Erasable Programmable Read Only Memory) interne

Le PIC18F45K20 contient 256 bytes d'EEPROM pour le stockage de données.

8 EEPROM : mémoire effaçable et programmable.

Voici les concepts-clés :

- 4 SFR(Special Function Register) contrôlent les opérations sur l'EEPROM : EECON1, EECON2, EEDATA, EEADR.
- L'EEPROM interne est lue/écrite 1 byte(octet) à la fois.
- Pour écrire dedans, une courte séquence d'instructions doit être écrite EECON2 tout de suite avant de commencer l'écriture, ainsi nous prévenons les écritures faîtes par sources d'erreur.
- Écrire un octet dans l'EEPROM prend un certain temps pour que le cycle d'écriture soit fait. Le micro-contrôleur continue d'exécuter du code durant le cycle d'écriture.

6.9. Les opération sur la mémoire programme (de type flash)

Le compilateur MPLAB C18 permet d'utiliser les pointeurs[9] (16 bits ou 24 bits). Pour choisir leur longueur : Project->Build Options-> Project.

16 bits est recommandé pour plus de 64 KB de mémoire flash(car le pointeur est plus rapide) . Il est aussi possible de déclarer individuellement un pointeur comme ceci :

near rom char *rom_pointer ; // 16 bits

ou (far) rom char *rom_pointer ; //24 bits

Puis de déclarer et d'utiliser ainsi

#pragma romdata maystrings = 0x100

rom char hello_str[] = « Hello » ;

rom pointer = hello_str ; // = &hello_str[0]

char letter = *rom_pointer ;

Ainsi, la première lettre ('H') de hello_str[] est pointée par rom_pointer et la valeur de letter est 'H'.

rom_pointer = (near rom char*)0x320 ;

A présent, rom_pointer pointe sur l'octet situé à l'adresse 0x320 de la mémoire programme.

Lire la mémoire programme FLASH nécessite donc la déclaration d'un pointeur ROM et d'une instruction pour lire la valeur pointée par ce pointeur.

Effacer et écrire cette mémoire

Contrairement à l'EEPROM, il faut effacer cette mémoire afin de pouvoir écrire dedans.

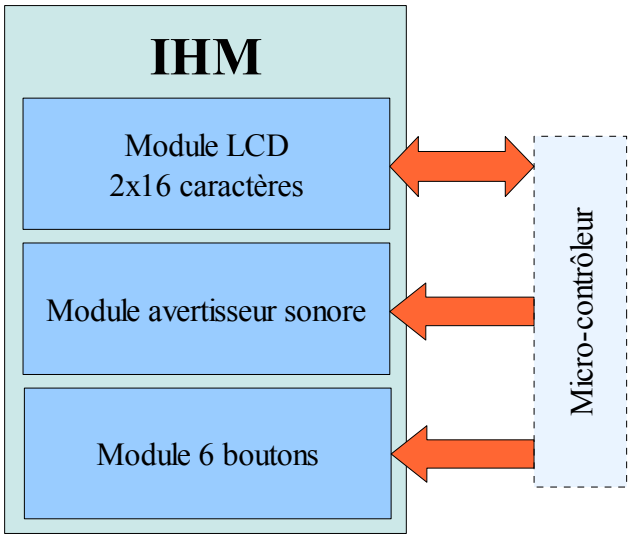
Effacer revient à écrire 0.

Pour les PIC18F45K20, nous ne pouvons effacer que par blocs de 64 octets(à partir d'une adresse définie), de 128 à 191 par exemple mais pas de 100 à 163 (pour en savoir plus, il faut regarder la procédure d'effacement dans la documentation).

9 Un pointeur est une variable contenant l'adresse d'une autre variable d'un type donné .Il s'utilise de la manière suivante : *nom_de_la_variable.

7. La partie électronique du robot

Voyons tout d'abord comment est constituée l'Interface Homme-Machine de notre robot.



Dessin 9: l'Interface Homme-Machine de notre robot

L'IHM de notre robot se décompose donc en 3 parties : un écran, un avertisseur sonore et un clavier de 6 touches.

7.1. Module n°1 - LCD 2 lignes x 16 caractères

7.1.1. Description

Le premier module de notre IHM permet la visualisation d'informations. Il s'agit d'un module LCD ayant la capacité d'afficher jusqu'à 32 caractères répartis sur deux lignes. Comme tout module de ce type, il intègre en son sein tous les éléments utiles : un écran LCD, une horloge interne, et des mémoires ROM et RAM.

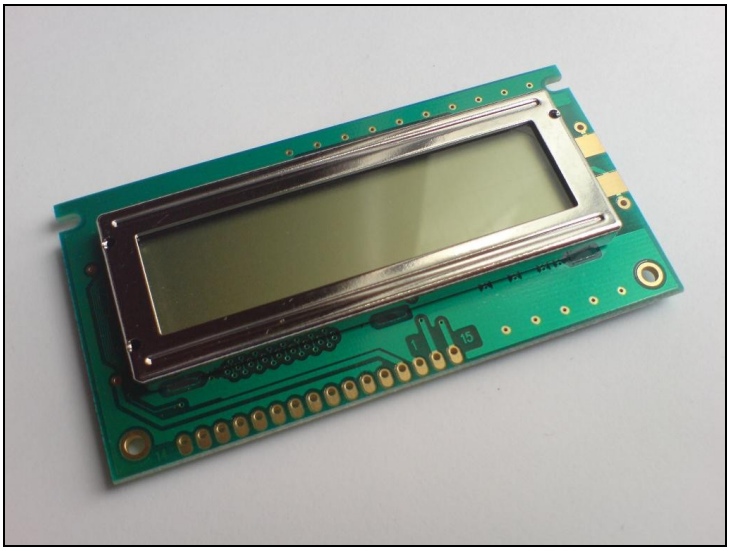


Illustration 1: photographie de l'afficheur LCD 2 lignes x 16 caractères

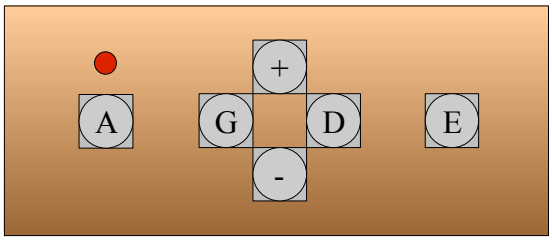
7.1.2. Fonctionnement

Nous aborderons le fonctionnement de ce module, en détail, un peu plus loin dans notre notre rapport, dans la partie consacrée à la programmation de notre projet. En effet, son utilisation requiert l'écriture d'un code respectant les instructions et les procédures décrites dans son *datasheet*.

7.2. Module n°2 - Carte 6 boutons et témoin lumineux

7.2.1. Description

Tout aussi indispensable pour notre IHM que le module d'affichage, voici comment se présente notre carte 6 boutons :



Dessin 10: sisposition des 6 boutons

Cette disposition nous permettra de naviguer aisément dans les sous-menus et de modifier les valeurs affichées à l'écran. Ainsi, nous pourrons modifier le comportement du robot ou encore régler ce qui nécessite un étalonnage.

Voici à présent les fonctions de chaque touche :

E	Validation du choix ou de la valeur
A	Annulation ou Réinitialisation à la valeur par défaut
G	Retour en arrière ou Déplacement à gauche
D	Entrer dans le sous-menu ou Déplacement à droite
+	Incrémenter la valeur affichée
-	Décrémenter la valeur affichée

Tableau 1: fonctions des boutons

Pour cette carte, nous avons établi le schéma électrique suivant :

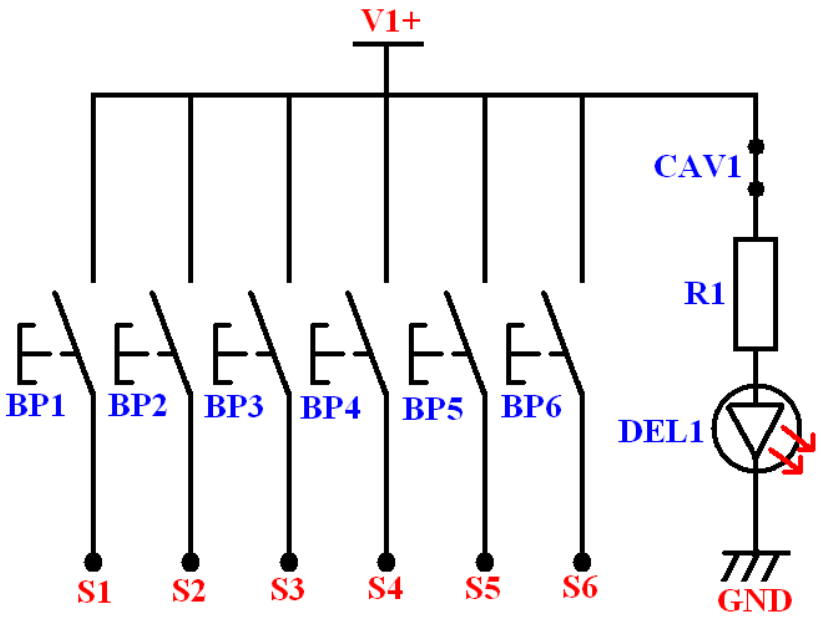


Illustration 2: schéma électrique de la carte n°2

7.2.2. Nomenclature

V1+	Tension de 3V
S1, S2, S3, S4, S5, S6	Sorties associées aux boutons
GND	Masse

Tableau 2: entrées/sorties de la carte n°2

DEL1	Diode électro-luminescente basse-consommation d'énergie
R1	Résistance de 400 ohms ¼ W
BP1, BP2, BP3, BP4, BP5, BP6	Boutons poussoirs
CAV1	Cavalier

Tableau 3: composants de la carte n°2

7.2.3. Typon et photographie de la carte réalisée

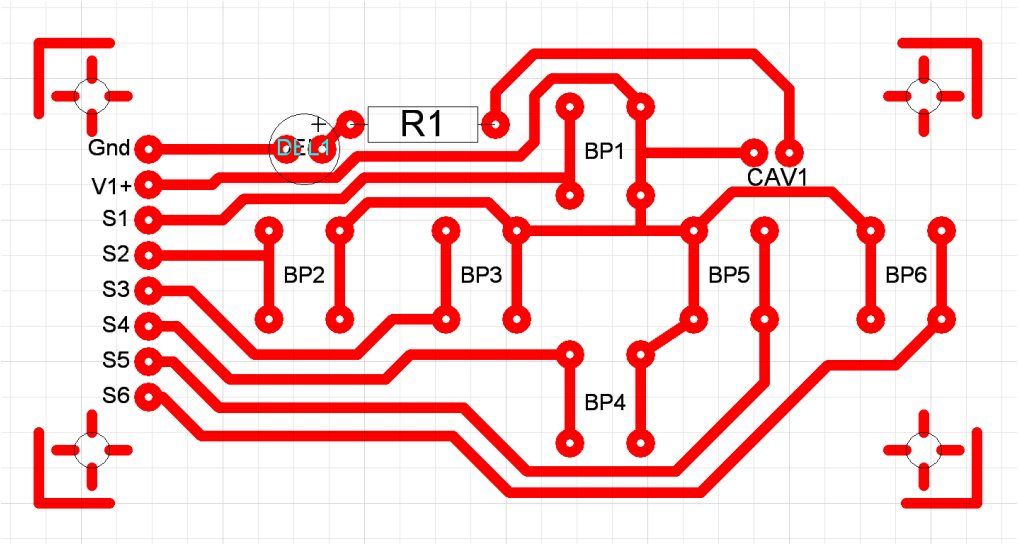


Illustration 3: typon de la carte n°2

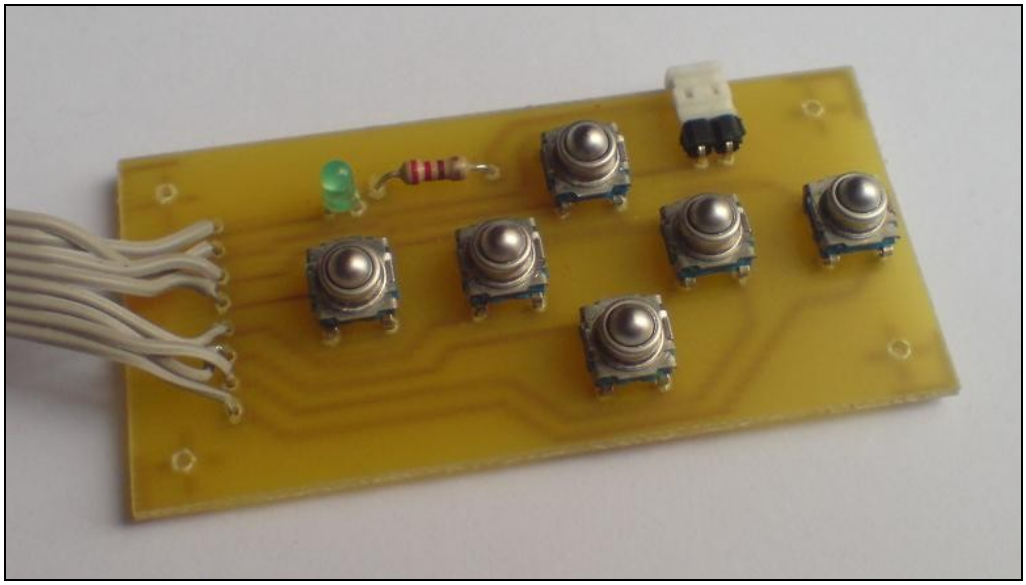


Illustration 4: photographie de la carte à 6 boutons

7.3. Module n°3 - Avertisseur sonore et témoin lumineux

7.3.1. Description

Cette carte a deux fonctions : celle d'"avertisseur sonore" et celle de "témoin lumineux".

La fonction "avertisseur sonore" permet d'enrichir l'Interface Homme-Machine du robot. Grâce à elle, il pourra communiquer des informations en utilisant des sons. Lors de la rencontre d'un obstacle, il en émettra un. Lors de la capture des informations (température et intensité lumineuse), il en émettra un autre. Ainsi, à chaque action ou problème rencontré par le robot, nous pourrons associer un son bien particulier.

Le témoin lumineux constitue la seconde fonction de cette carte. Ce voyant servira à informer l'utilisateur de ce module de la présence ou non d'une tension dans son système. Toutefois, puisqu'une autre carte de notre système dispose déjà de cette fonctionnalité, nous désactiverons le témoin lumineux de cette carte.

7.3.2. Tests préliminaires

Nous avons récupéré un *buzzer* sur une ancienne carte-mère d'ordinateur. Néanmoins, ne sachant pas s'il s'agissait réellement d'un *buzzer,* fonctionnant en continu, ou d'un transducteur, fonctionnant en alternatif, nous avons dû tester ce composant en appliquant une tension continue de 5V à ses bornes.

Nous n'avons pas entendu un son durable. Seul un bruit faible et bref se faisait entendre à chaque fois que nous connections l'alimentation 5V. Nous en avons conclu qu'il s'agissait d'un transducteur.

Nous avons alors fait un petit montage sur plaque d'essai. Nous avons utilisé le transducteur, un transistor NPN, une résistance et un générateur de basse fréquence pour le signal carré.

Le test s'est révélé concluant ; de celui-ci découle le schéma électrique de notre troisième module.

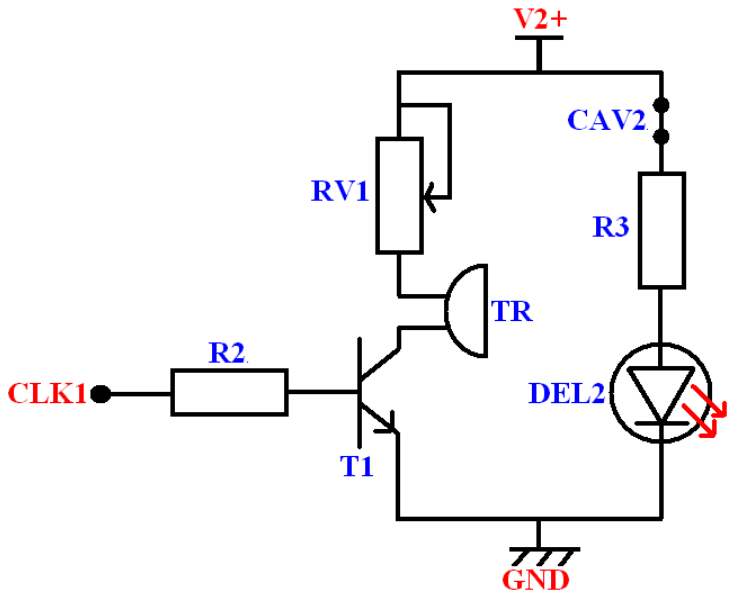


Illustration 5: schéma électrique de la carte n°3

7.3.3. Fonctionnement

Nous appliquons un signal carré sur l'entrée CLK1. Lorsque le signal est à l'état haut (3V), le transistor T1 est passant ; lorsqu'il est à l'état bas (0V), le transistor est bloqué. Ainsi, le transducteur TR est parcouru par un courant alternatif. Le son émis par TR dépend de la fréquence du signal carré, et son intensité change en fonction de la valeur de la résistance variable RV1.

La présence (ou l'absence) du cavalier CAV1 permet, si besoin, d'activer (ou de désactiver) la diode lumineuse présente sur la carte.

7.3.4. Nomenclature

V2+	Tension de 5V
CLK1	Signal carré de fréquence variable
GND	Masse

Tableau 4: entrées/sorties de la carte n°3

R2	Résistance de 2,1kohms ¼ W
R3	Résistance de 220 ohms ¼ W
RV1	Résistance variable de 220 ohms
TR	Transducteur pièzo-électrique
T1	Transistor NPN
DEL2	Diode électro-luminescente rouge 1,2V
CAV2	Cavalier

Tableau 5: composants de la carte n°3

7.3.5. Typon et photographie de la carte réalisé

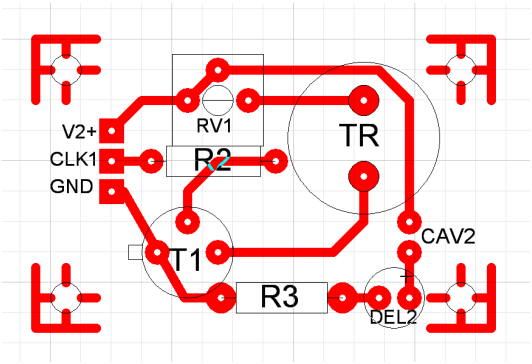


Illustration 7: typon de la carte n°3

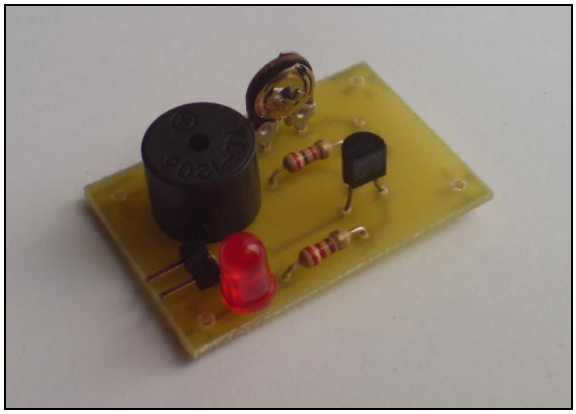


Illustration 6: photographie de la carte Avertisseur sonore

7.4. Module n°4 - Carte contrôleur (pour un moteur à courant continu)

7.4.1. Description

Pour aller chercher des informations, notre robot doit pouvoir se mouvoir librement. Il est donc nécessaire de l'équiper d'actionneurs. Comme motorisation, nous avons choisi d'utiliser deux

moteurs à courant continu.

Pour contrôler ces deux moteurs, nous avons décidé de réaliser deux cartes séparées. Chaque module comporte un circuit intégré, nous permettant de piloter un moteur dans le sens horaire et anti-horaire (circuit de type « Pont en H »), ainsi qu'un système compte-tours, nous permettant de calculer les distances parcourues et les vitesses.

7.4.2. Fonctionnement

Nous avons choisi d'utiliser une roue percée de nombreux trous pour pouvoir compter le nombre de tours effectué par la roue. Par le biais d'un montage de type trigger de Schmitt, nous obtiendrons un signal en créneau pour chaque roue. Ces créneaux nous permettront, entre autre, de calculer les distances parcourues.

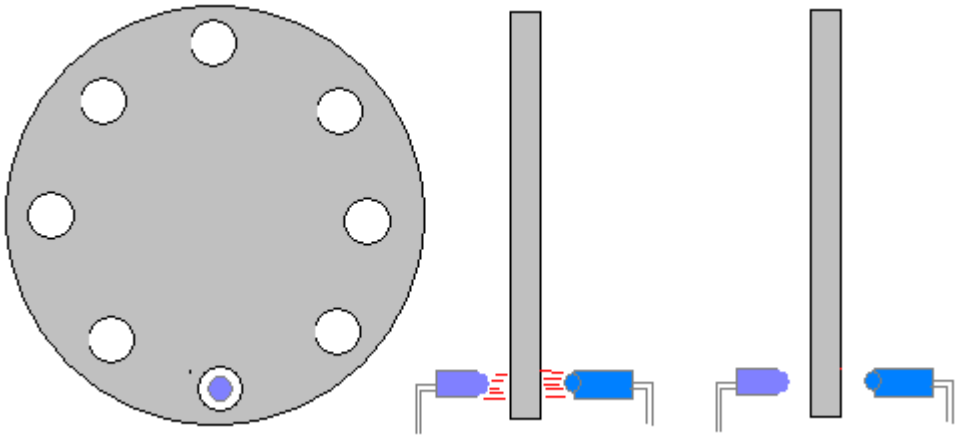


Illustration 8: la roue et le système de détection optique

Voici le premier schéma que nous avons élaboré pour cette carte :

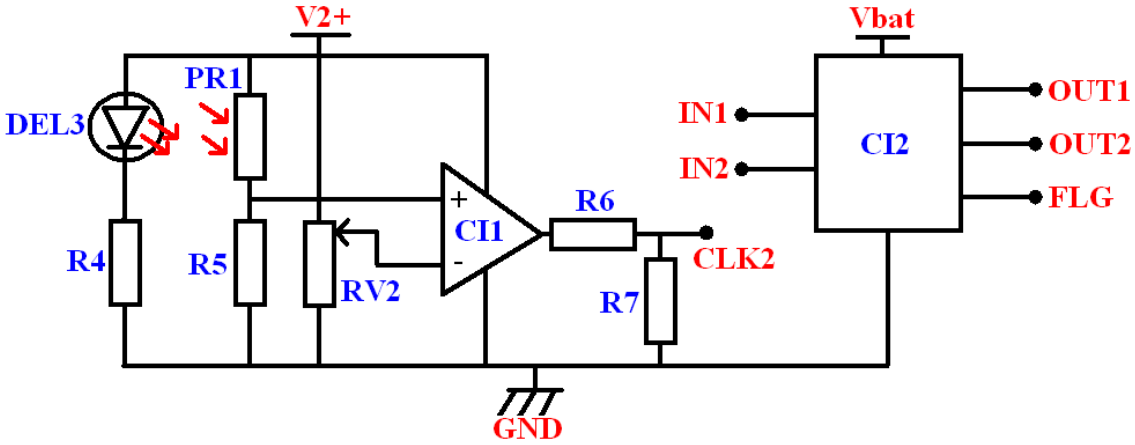


Illustration 9: premier schéma électrique de la carte n°4

Des tests nous ont prouvé la faisabilité de ce circuit, toutefois, ceux-ci ont mis en exergue les

nombreux défauts dont souffrait la partie compte-tours de ce montage.

En effet, la photo-résistance nous est apparue comme un piètre capteur pour l'usage auquel nous le destinions.

Premièrement, une photo-résistance est sensible à la lumière du jour ; son usage nous aurait contraint à fabriquer un habillage imperméable à la lumière pour prévenir tout information parasite sur les capteurs. Deuxièmement, sa grande surface nous aurait obligé à faire des trous de tailles conséquentes, nous imposant une résolution faible. Enfin, pour terminer, son temps de réaction nous a semblé faible.

En guise de capteur, nous avons donc opté pour un autre couple de composant : une diode Infra-Rouge et une photo-diode Infra-Rouge. Nous avons légèrement modifier notre schéma initial.

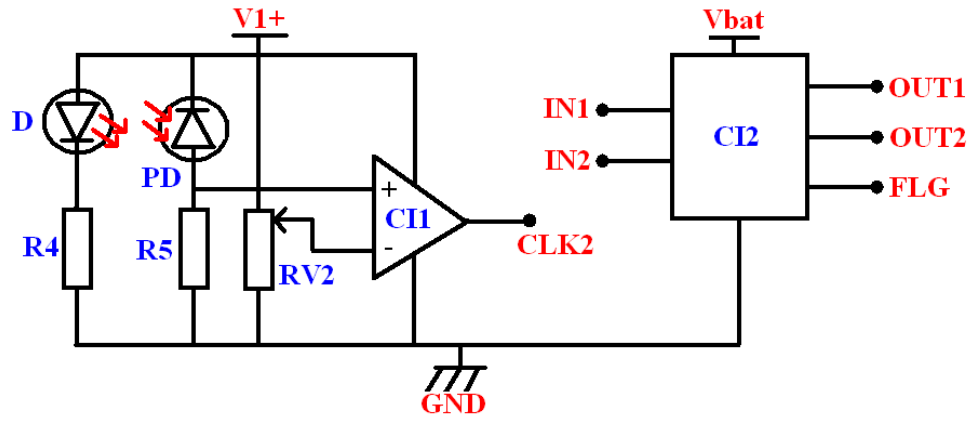


Illustration 10: schéma électrique retenu pour la carte n°4

Remarquons que l'usage de ces autres composants nous a permis d'utiliser V1+ (Alimentation 3V) à la place de V2+ (Alimentation 5V). De même, les résistance R6 et R7 sont devenues inutiles et ont donc été retirées.

7.4.3. Nomenclature

V1+	Tension de 5V
Vbat	Tension de la batterie
GND	Masse
IN1	Entrée n°1 pour le signal état Haut/état Bas ou MLI
IN2	Entrée n°2 pour le signal état Haut/état Bas ou MLI
CLK2	Signal carré
FLG	Sortie signalant une erreur
OUT1	Sortie n°1 pour alimentation du moteur
OUT2	Sortie n°2 pour alimentation du moteur

Tableau 6: entrées/sorties de la carte n°4

R4, R5	Résistance de ? ¼ W
R7	Résistance de ? ¼ W
RV2	Résistance variable de 10 kohms
PD	Photo-diode Infra-Rouge
D	Diode Infra-Rouge
CI1	Ampli-OP
CI2	Pont en H - TLE5206

Tableau 7: composants de la carte n°4

7.4.4. Typon et photographie de la carte réalisée

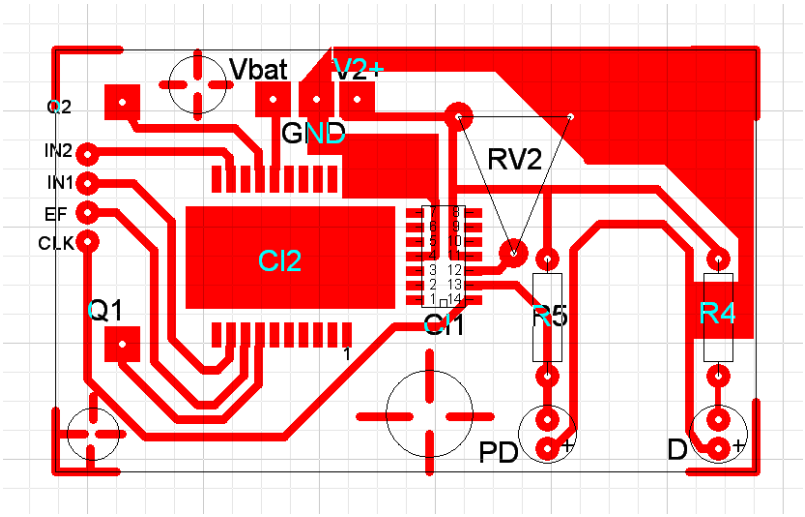


Illustration 11: typon des modules n°4

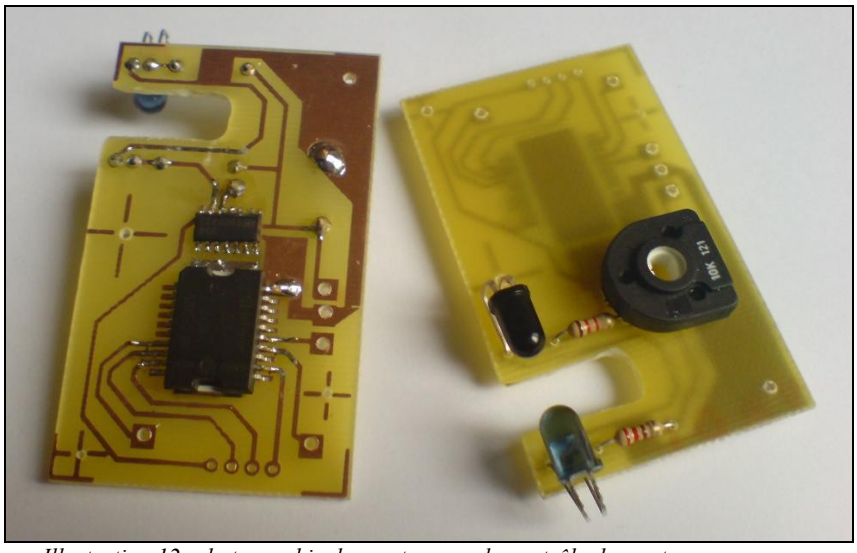


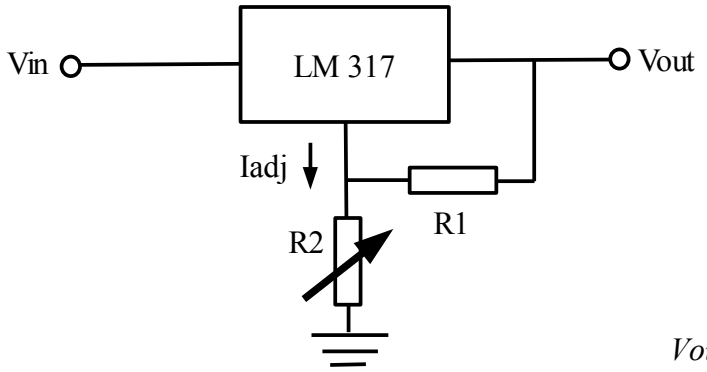
Illustration 12: photographie des cartes pour le contrôle des moteurs

7.5. Module n°5 – Carte d'alimentation

7.5.1. Description

Pour fonctionner, notre système a besoin de plusieurs sources de tension. Une source de 3 V est nécessaire pour le micro-contrôleur et les amplificateurs opérationnels et une autre, de 5 V, l'est pour le capteur de tension et les diodes blanches de fortes puissances ; c'est la raison pour laquelle nous avons fabriqué une carte d'alimentation capable de fournir à notre système jusqu'à 4 sources de tension différentes : 3 sources régulées de valeur réglable (par le biais d'une résistance variable), et la source du générateur. Nous avons choisi un régulateur très bon marché pour cette fonction de régulation : le LM317.

7.5.2. Fonctionnement



$$Vout = 1{,}25\,V \times (1 + R2/R1) + Iadj \times R2$$

Dessin 11: leLM317

Afin de calculer les valeurs désirées pour la tension et le courant de sortie de chacun des régulateurs , nous avons négligé le courant I_{adj} (inférieur à 100µA) et déterminer le rapport entre la résistance ajustable R2 et la résistance R1.

Le constructeur recommande dans certains cas l'usage de condensateurs en entrée et/ou en sortie. Dans notre cas, leur présence n'est pas indispensable. Toutefois, nous en avons mis quelques-uns afin de lisser un peu notre tension (nb : condensateurs non représentés sur le schéma électrique)

7.5.3. Nomenclature

GND	Masse
V1+	Source de tension de réglable (réglée sur 3V)
V2+	Source de tension de réglable (réglée sur 5V)
V3+	Source de tension de réglable (non utilisée)
Vbat	Source de tension délivrée en entrée de la carte par la batterie *(source disponible également en sortie de la carte pour l'alimentation des moteurs à courant continu)*

Tableau 8: entrées/sorties de la carte n°5

R9	Résistance de 680 ohms ¼ W
R6, R7, R8	Résistance de 1kohms ¼ W
RV3, RV4, RV5	Résistance variable 10 k ohms
CI3, CI4, CI5	LM 317
DEL3	DEL rouge
CAV3	Cavalier
F	Fusible
C1, C2, C3	Condensateur chimique 47uF

Tableau 9: composants de la carte n°5

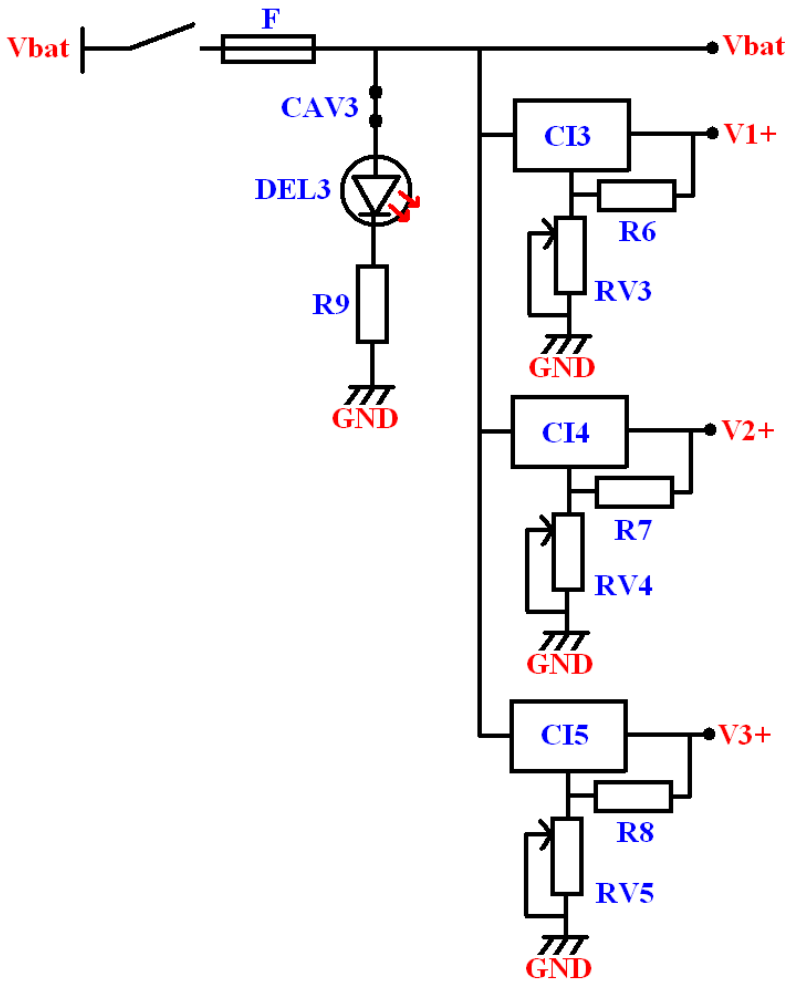


Illustration 13: schéma électrique de la carte d'alimentation

7.5.4. Typon et photographie de la carte réalisée

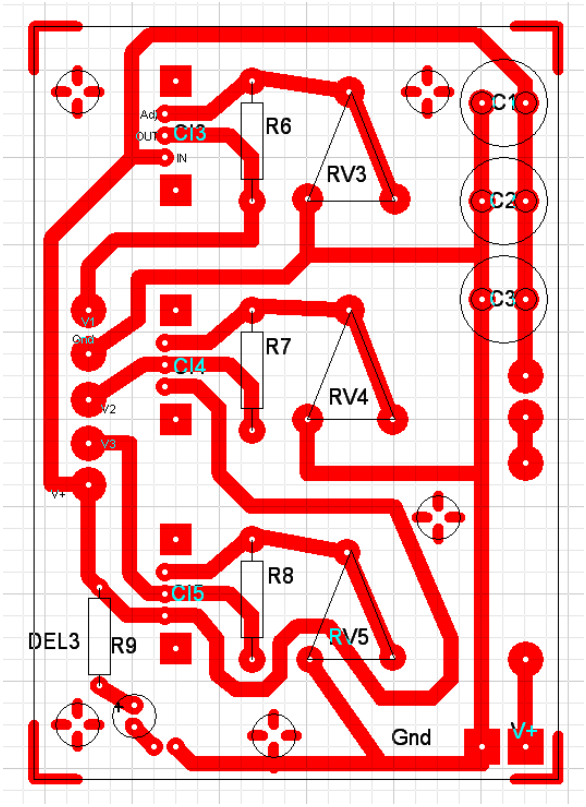


Illustration 14: typon des modules n°5

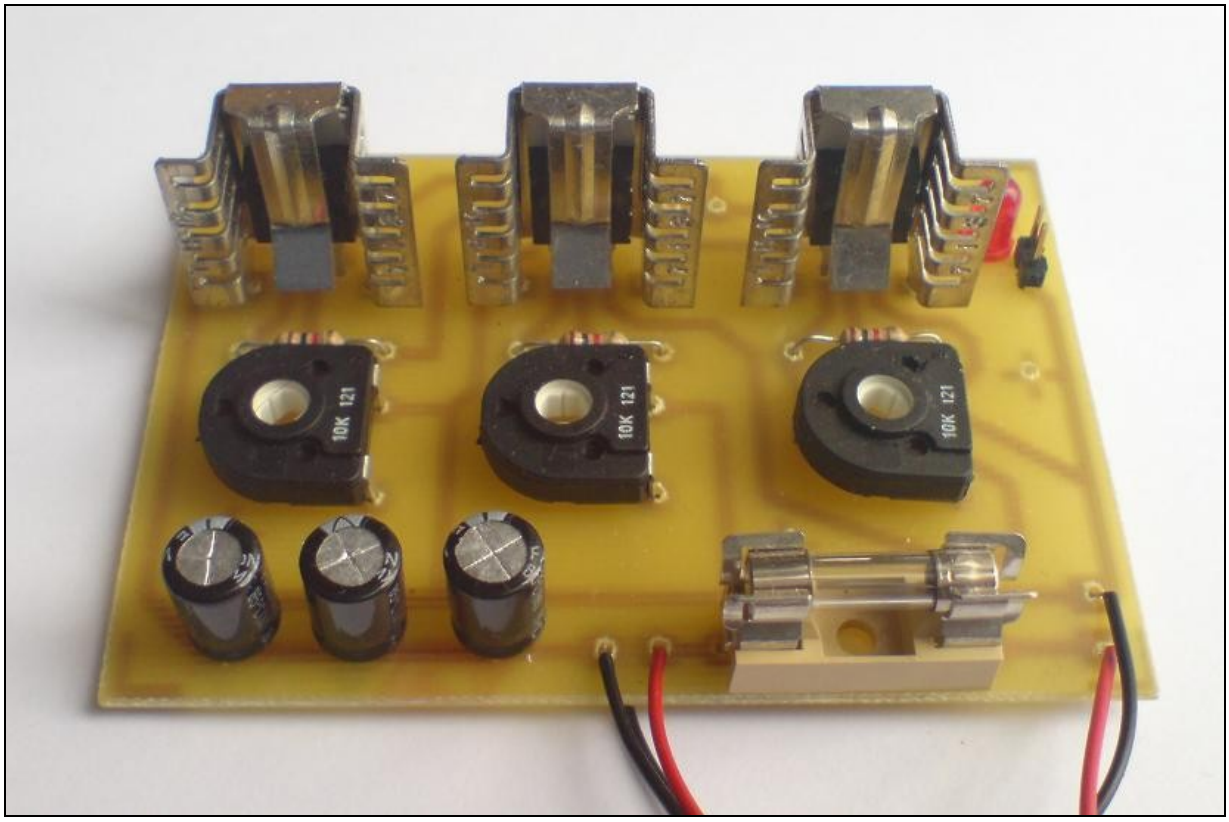


Illustration 15: photographie de la carte d'alimentation

7.6. Module n°6 – La carte des capteurs et des détecteurs

7.6.1. Description

Les capteurs et les détecteurs permettent au robot de glaner des informations sur son environnement. C'est à partir de ces informations qu'il peut « prendre des décisions ». Plus un robot a de capteurs et plus il offre de possibilités au programmeur.

A l'origine, ce robot devait seulement être en mesure de savoir s'il avait heurté quelque chose et de modifier sa course en conséquence. La création de cette carte n'était donc pas prévue dans le cahier des charges initial ; seul un bouton poussoir utilisé en guise de détecteur de chocs était prévu. Toutefois, l'un de nous souhaitant se construire une plateforme de test la plus complète possible, pour expérimenter divers algorithmes mais aussi, et surtout, pour enrichir ses connaissances sur les micro-contrôleurs, la décision de rajouter d'autres capteurs fut prise. Tout d'abord, nous décidâmes de mettre deux capteurs de lumière, puis, un capteur de température LM45 au format CMS[10]. Enfin, la lecture d'un ancien rapport portant sur un projet robotique type « suiveur de ligne » nous donna l'envie de rajouter des détecteurs capables de détecter 2 teintes l'une de l'autre. Ne disposant pas de capteur Infra-Rouge adéquats, nous avons utilisé un couple LED blanche – photo-résistance, avec l'idée de les protéger de toute lumière parasite. Seuls des tests – non réalisés – permettront de déterminer si ce choix était judicieux.

La fabrication de cette carte – et la volonté d'étendre les capacités de ce robot – est l'une des raisons pour laquelle le projet « robot explorateur », robot qui change de direction lorsqu'il touche un obstacle, n'a pas abouti dans le temps imparti.

Voici le schéma électrique partiel de cette carte (le LM45 est absent et les boutons poussoirs ont été retirés pour être simplement déportés) :

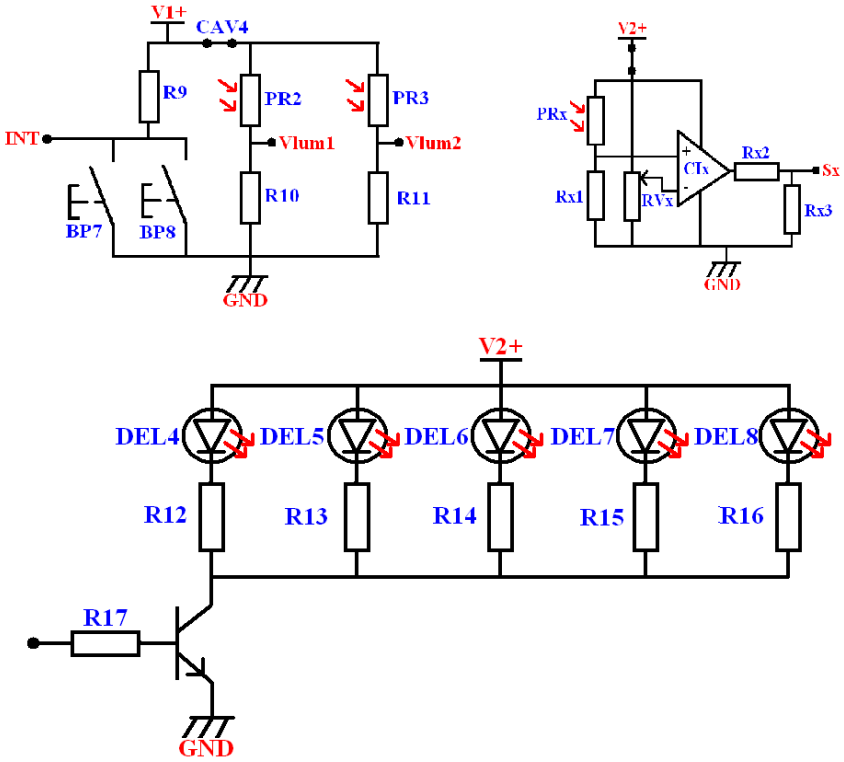


Illustration 16: schéma électrique partiel de la carte des capteurs

10 CMS : Composant Monté en Surface

7.6.2. Nomenclature

GND	Masse
V1+	Source de tension de 3V
V2+	Source de tension de 5V
INT	Sortie détecteur de choc
Vlum1, Vlum2	Tension image de la lumière
Sx	X sorties numérique du détecteur de ligne

Tableau 10: entrées/sorties de la carte n°6

R9	?
R10, R11	Résistance de 3,9 k ohms ¼ W
BP7, BP8	Retirés de la carte
R12, R13, R14, R15, R16	Résistance de 270 ohms ¼ W
R17	Résistance de 2 k ohms ¼ W
DEL4, DEL5, DEL6, DEL7, DEL8	Diode électro-luminescente blanche
PR3, PR4, PRx={PR5, PR6, PR7, PR8, PR9}	Photo-résistance 100 ohms-100kohms
CIx	LM 324
T2	Transistor NPN BC 338

Tableau 11: composants de la carte n°6

Typon et photographie de la carte réalisée

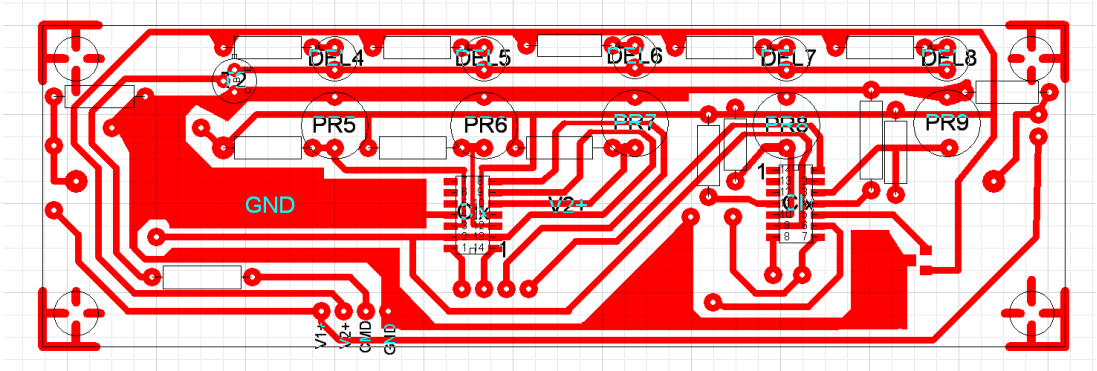


Illustration 17: typon final de la carte des capteurs

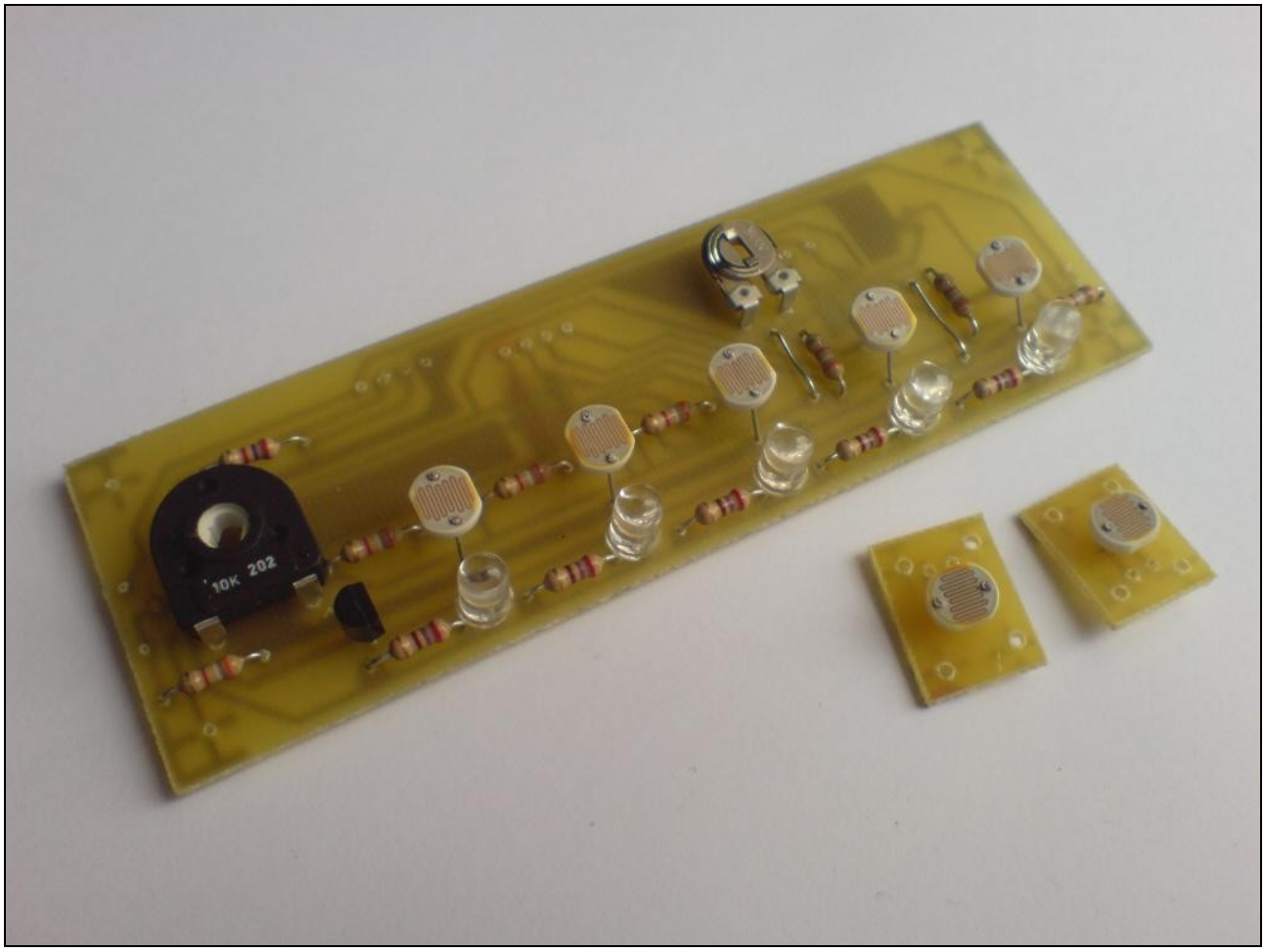


Illustration 18: les cartes de la partie "capteurs" de notre projet

Nous avons également profité d'avoir une plaque trop grande pour fabriquer, sur le surplus, 3 petites cartes : 2 pour déporter les capteurs de lumière, et une équipé d'un interrupteur infra-rouge. Cette dernière carte, positionnée sur le nez du robot permettra peut-être de détecter la présence/l'absence du sol (pour éviter la chute dans un escalier par exemple. Si ce composant ne permettait pas d'atteindre ce but, ce petit module pourra toujours servir en tant qu'interrupteur sans contact.

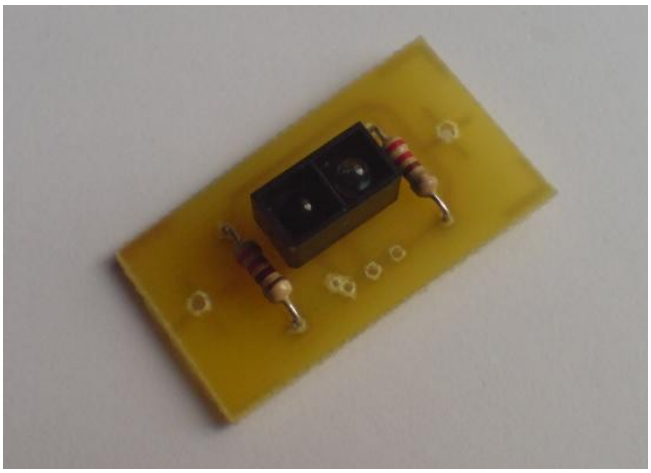


Illustration 19: mini carte "détecteur de sol"

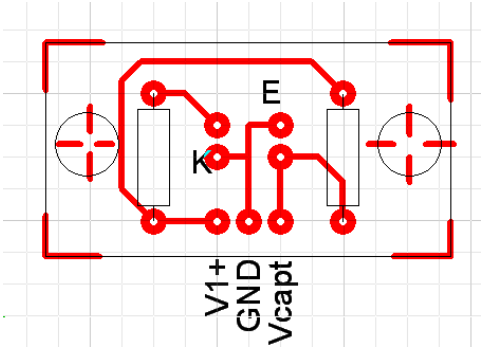


Illustration 20: typon du module interrupteur IR

7.7. Module n°7 – La carte du micro-contrôleur

7.7.1. Description

Lors de notre étude du PIC18F45K20, nous nous sommes rendus compte de l'utilité de la carte de test fourni avec le programmateur PICKIT, c'est pourquoi nous avons décidé de créer une carte similaire pour la carte qui est au cœur de notre système et qui reçoit notre PIC18F46K20 au format DIL 40 broches. Nous avons fait en sorte que la quasi-totalité des broches, bien que déjà utilisées par une connexion soient également accessibles via des connecteurs « tulipes ».

7.7.2. Typon et photographie de la carte réalisée

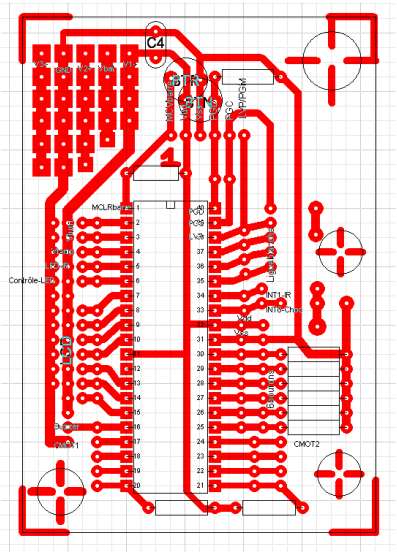


Illustration 21: typon de la carte du micro-contrôleur

Cette carte a été fabriquée en dernier. Lors de sa conception, nous avons pu prendre en compte les nombreuses cartes que nous avions déjà conçues, pour imposer un câblage logique (voir ***6.2.1. Fonctions assignées aux broches***)

Nous n'avons pu terminer nos cartes durant les séances de TP, néanmoins, nous avons pu avoir facilement accès à la salle de gravure, ce qui nous a permis de terminer nos cartes à la date du 23 mars.

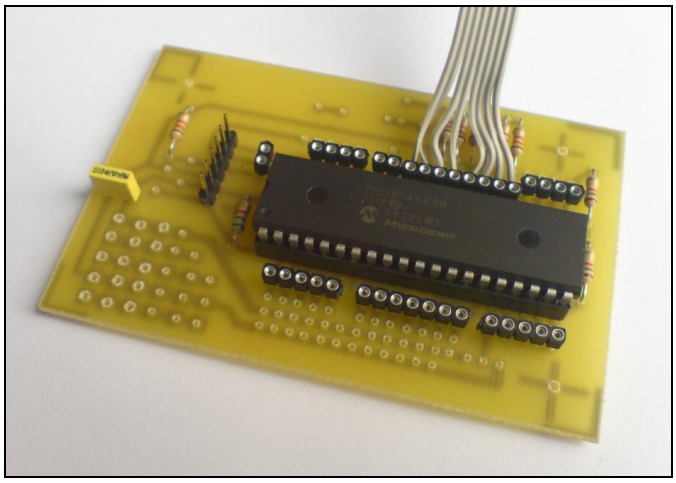
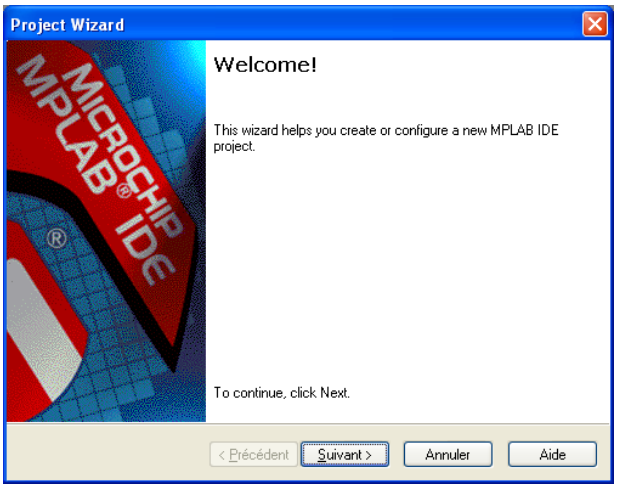


Illustration 22: la carte réalisée pour le micro-contrôleur

8. La programmation du robot

8.1. Création du projet

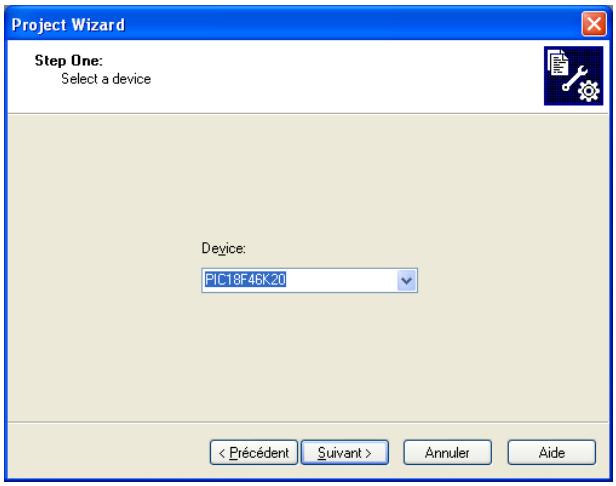
Une fois nos cartes terminées, nous avons pu entamer l'écriture du code propre à notre robot. Contrairement à CodeVisionAVR d'ATMEL, le logiciel MPLAB de MICROCHIP ne propose pas un équivalent à *CodeWizard*. La configuration des périphériques doit donc se faire en langage C, en respectant les règles et les mnémoniques connus du compilateur C18. Toutefois, MPLAB dispose d'un *Project Wizard* facilitant la création d'un nouveau projet.



ETAPE N°1

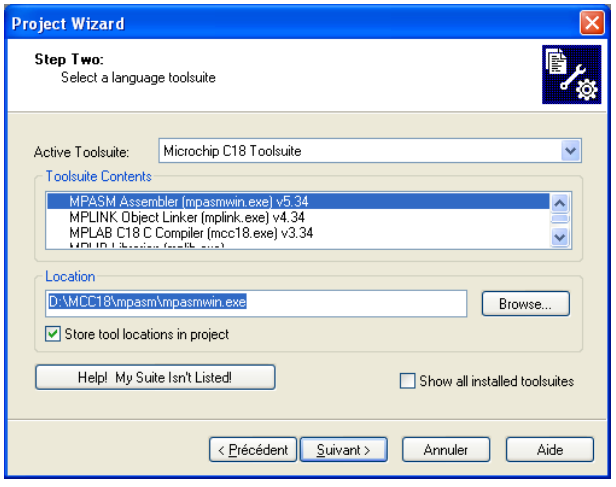
Lancer Project Wizard.

Project > Project Wizard



ETAPE N°2

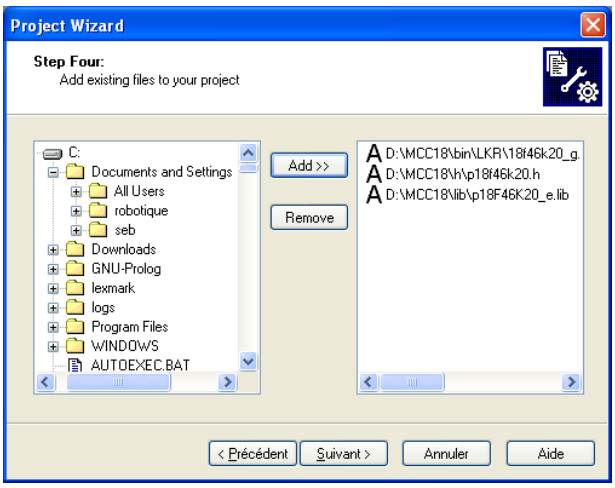
Sélectionner le bon micro-contrôle



ETAPE N°3

Choisir le compilateur C18

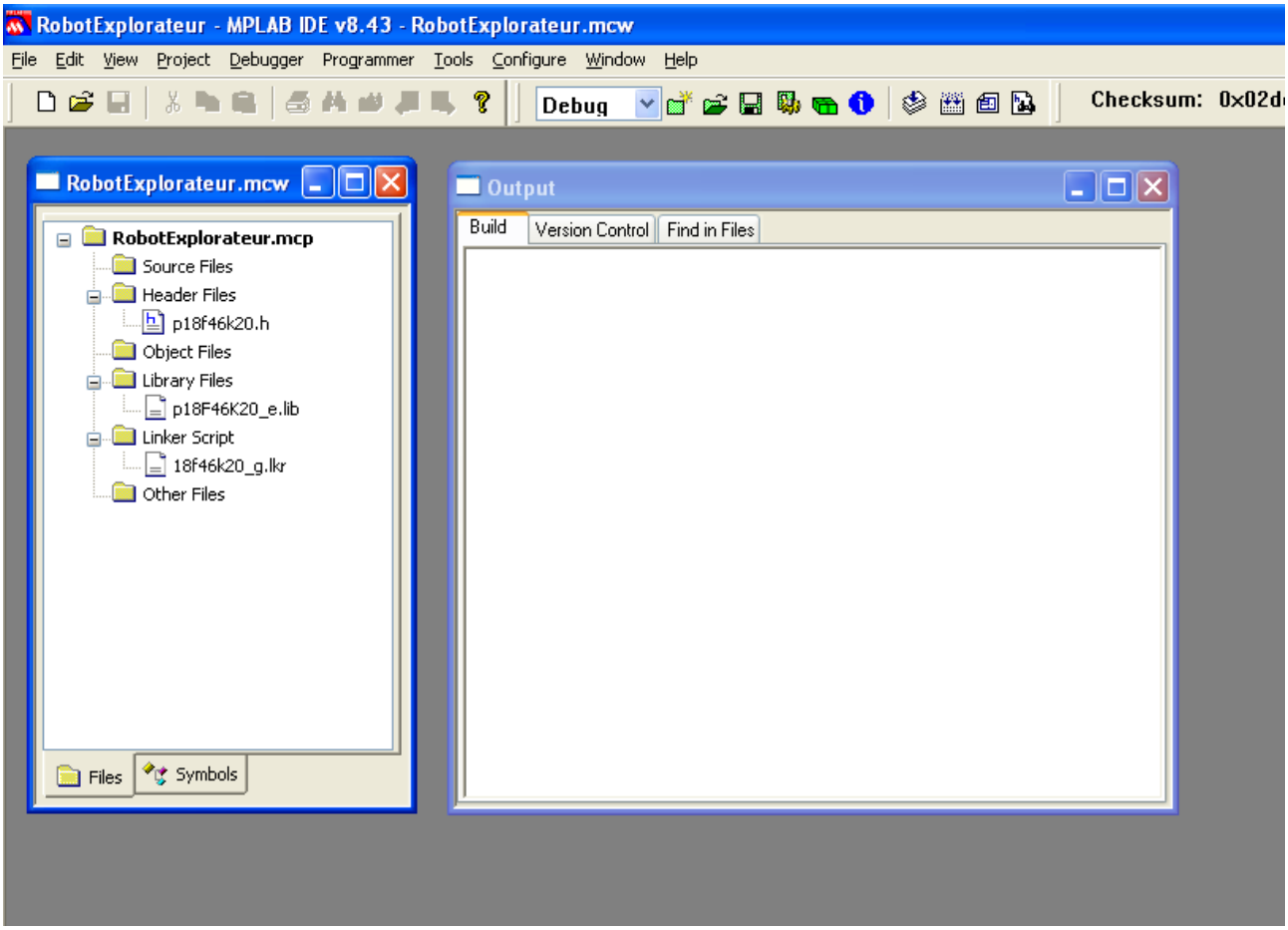
(nb : cocher la case « store tool locations in project » permet d'éviter certains désagréments)



ETAPE N°4

Sélectionner les fichiers .h, .lib, .lkr adéquats. (dans le répertoire nommé MCC18)

Après avoir terminé la procédure, voici ce que nous avons sous nos yeux :



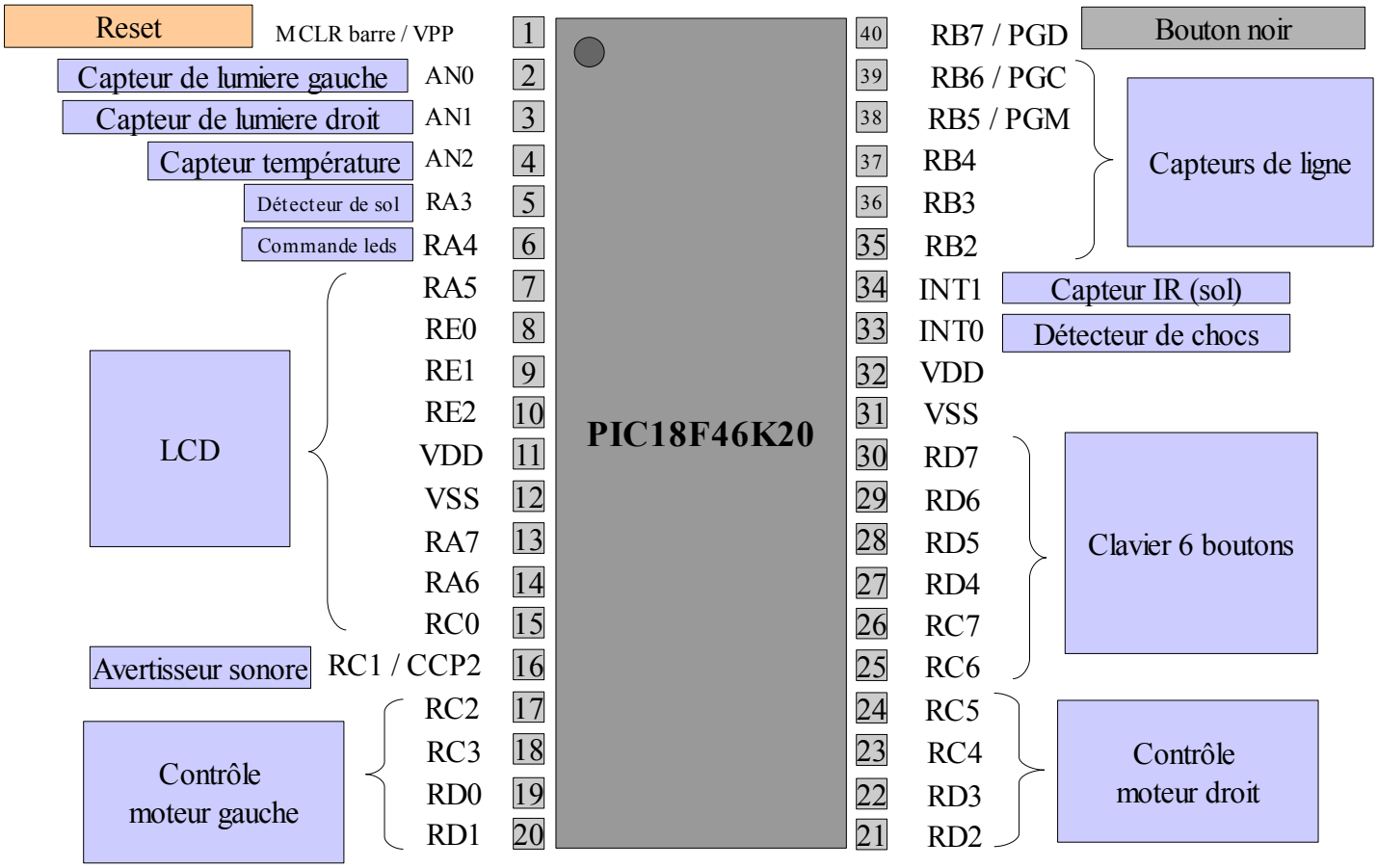
Il suffit maintenant d'aller dans ***File > New*** pour créer un nouveau fichier, puis de le sauvegarder avec l'extension .c : il s'agira de notre code source dans lequel nous écrirons notre fonction principale *main()*.

Maintenant que notre projet est créé, qu'il contient un fichier .c, nous pouvons passer à la configuration du micro-contrôleur.

8.2. Configuration du micro-contrôleur PIC18F46K20

8.2.1. Fonctions assignées aux broches

Avant de coder, il nous faut assigner à chaque broche du composant une fonctionnalité bien spécifique. Notre câblage et notre programmation dépendent de cette assignation.



Dessin 12: assignation des broches du PIC18F46K20

Maintenant que la fonction de chaque broche est définie, nous pouvons nous lancer dans la programmation de notre micro-contrôleur. Nous commençons tout d'abord par configurer ses périphériques et ses broches, conformément aux choix que nous avons fait.

8.2.2. Activation des périphériques du micro-contrôleur

Avant tout autre chose, nous devons activer les périphériques nécessaires à notre projet. Bien sûr, par la même occasion, nous désactiverons ceux qui nous sont inutiles. Nous utiliserons dans les deux cas la commande prévue à cette effet pour le compilateur C18 : **#pragma config**.

Nous cliquons donc sur ***Help > Topics...*** pour ouvrir la fenêtre ***MPLAB Help Topics***, puis nous double-cliquons sur ***PIC18 Config Settings***. Nous choisissons ***PIC18F46K20*** dans la liste. Toutes les informations relatives aux bits de configuration de notre composant s'affichent devant nous.

Nous configurons le maximum de bits de configuration du PIC18F46K20, comme nous le conseille le constructeur dans différents documents. Néanmoins, nous n'avons pas touché à certains bits. Ne sachant pas quelle pouvait être la répercussion de leur activation/désactivation, nous avons laissé au compilateur C18 le soin de donner à ceux-ci la valeur qui leur était assignée par défaut.

Voici le code que nous avons écrit :

```
// Nous choisissons l'oscillateur interne comme horloge
// et nous définissons RA6 et RA7 comme des ports I/O.
// Ceux-ci sont utilisés pour le contrôle du LCD.
#pragma config FOSC = INTIO67

// Le RESET externe est mis à disposition (RE3 inutilisable)
// Ainsi, on pourra forcer un redémarrage du programme, en appuyant
// sur un bouton rouge par exemple.
#pragma config MCLRE = ON

// Nous désactivons la fonction "entrée analogique" sur les bits PORTB<4:0>
#pragma config PBADEN = OFF

// Nous multiplexons le signal MLI de CCP2 avec la broche RC1
// (et non avec la broche RB2). La broche RC1 servira peut-être de
// commande au module "Avertisseur sonore", car il est très facile
// de faire varier la fréquence de CCP2 via un registre. Toutefois,
// peut-être utiliserons nous une autre méthode pour obtenir
// un signal de fréquence variable sur la sortie RC1...
#pragma config CCP2MX = PORTC

// Nous désactivons la plupart des périphériques avancés
#pragma config FCMEN = OFF, IESO = OFF, PWRT = OFF, BOREN = OFF
#pragma config HFOFST = OFF, LPT1OSC = OFF, WDTEN = OFF

// Nous désactivons la protection du code et permettons l'accès à l'EEPROM
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF
#pragma config CPB = OFF, CPD = OFF
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF
#pragma config WRTB = OFF, WRTC = OFF, WRTD = OFF
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF, EBTRB = OFF
```

A présent, nous pouvons paramétrer les différents périphériques en modifiant les valeurs de certains registres.

8.3. Paramétrage des périphériques

Nous configurons nos périphériques (Horloge, Entrées/Sorties, CAN, écran LCD...) au début de la fonction main(). Nous créons une fonction *InitialisationDuSysteme()*.

8.3.1. Configuration de l'horloge

Puisque nous avons décidé d'utiliser l'horloge interne du micro-contrôleur, il nous faut définir sa fréquence. A cette fin, nous écrivons ceci :

```
OSCCONbits.IRCF0=0;     // Nous réglons les bits du Registre IRCF
OSCCONbits.IRCF1=1;     // pour imposer une fréquence d'horloge
OSCCONbits.IRCF2=1;     // de 16 MHz.
OSCTUNEbits.PLLEN=0;    // Nous désactivons la PLL (qui multiplierait
                        // sinon par 4 l'horloge interne)
```

8.3.2. Déclaration des entrées/sorties

Afin de rendre notre code plus lisible et de faciliter sa maintenance, nous créons cinq fichiers .h : **controledulcd.h, controleduson.h, controledudeplacement.h, controledesboutons.h, controledescapteursetdesdetecteurs.h,**

Ces cinq fichiers contiendront respectivement les mnémoniques et les fonctions relatives au contrôle du LCD, de l'avertisseur sonore, des moteurs, des capteurs/détecteurs, et des boutons. Nous les ajoutons au projet et rajoutons les **#include** nécessaires en haut de notre fichier .c principal.

Au sein de ces fichiers nous déclarons donc grâce à des **#define**, les mnémoniques pour toutes les entrées/sorties du micro-contrôleur. Ainsi, plutôt que d'avoir à écrire RA4=1, nous pourrons écrire CMD_LEDS=1. (signifie « allumer les leds blanches »). Voici ce que nous avons déclaré...

… dans **controledulcd.h** :

```
#define LCD_RS  LATAbits.LATA5
#define LCD_RW  LATEbits.LATE0
#define LCD_E   LATEbits.LATE1
#define LCD_DB4 LATEbits.LATE2
#define LCD_DB5 LATAbits.LATA7
#define LCD_DB6 LATAbits.LATA6
#define LCD_DB7 LATCbits.LATC0
```

... dans **controleduson.h :**

```
#define CMD_SON LATCbits.LATC1
```

... dans **controledudeplacement.h :**

```
// Moteur droit
#define CMD1_MOT_D  LATDbits.LATD3  // Commande n°1 moteur droit CMD1_MOT_D
#define CMD2_MOT_D  LATCbits.LATC4  // Commande n°2 moteur droit CMD2_MOT_D
#define COMPT_MOT_D LATCbits.LATC5  // Compteur moteur droit    COMPT_MOT_D
#define ERR_MOT_D   LATDbits.LATD2  // Erreur moteur droit  ERR_MOT_D

// Moteur gauche
#define CMD1_MOT_G  LATCbits.LATC3  // Commande n°1 moteur gauche CMD1_MOT_G
#define CMD2_MOT_G  LATDbits.LATD0  // Commande n°2 moteur gauche CMD2_MOT_G
#define COMPT_MOT_G LATCbits.LATC2  // Compteur moteur gauche   COMPT_MOT_G
#define ERR_MOT_G   LATDbits.LATD1  // Erreur moteur gauche ERR_MOT_G
```

... dans **controledescapteursetdesdetecteurs.h :**

```
// Capteurs de lumière et de température
#define CAPT_LUM_G  LATAbits.LATA0
#define CAPT_LUM_D  LATAbits.LATA1
#define CAPT_TEMP   LATAbits.LATA2

// Détecteur IR - Couple émetteur et récepteur infrarouge IR
#define EMET_IR     LATAbits.LATA3
#define RECEPT_IR   LATBbits.LATB1

// Commande des leds blanches du détecteur de ligne
#define CMD_LEDS    LATAbits.LATA4

// Les 5 détecteurs du detecteur de ligne
#define DETECT_LB1  LATBbits.LATB2
#define DETECT_LB2  LATBbits.LATB3
#define DETECT_LB3  LATBbits.LATB4
#define DETECT_LB4  LATBbits.LATB5
#define DETECT_LB5  LATBbits.LATB6

// Le détecteurs de chocs
#define DETECT_CHOC LATBbits.LATB0
```

Certains de ces mnémoniques déclarées ne seront jamais utilisés. Nous verrons plus tard

lesquels supprimer pour alléger notre code et garder l'indispensable. Pour le moment, tous ces mnémoniques n'ont qu'un seul but : faciliter, rendre plus agréable notre programmation.

8.3.3. Configuration des entrées/sorties

Certaines broches sont des entrées numériques, d'autres des entrées analogiques, d'autres encore des sorties numériques, c'est pourquoi nous devons dire qui fait quoi en configurant chaque broche ainsi :

```
/************************************
************************************
REGLAGE DES ENTREES NUMERIQUES
************************************
************************************/
TRISBbits.TRISB7=1; // Bouton noir
TRISCbits.TRISC6=1; // Bouton 1 BT1
TRISCbits.TRISC7=1; // Bouton 2 BT2
TRISDbits.TRISD4=1; // Bouton 3 BT3
TRISDbits.TRISD5=1; // Bouton 4 BT4
TRISDbits.TRISD6=1; // Bouton 5 BT5
TRISDbits.TRISD7=1; // Bouton 6 BT6
TRISBbits.TRISB0=1; // Détecteur de choc DETECT_CHOC
TRISBbits.TRISB1=1; // Détecteur de sol (Récepteur) RECEPT_IR
TRISBbits.TRISB2=1; // Détecteur n°1 Ligne Blanche DETECT_LB1
TRISBbits.TRISB3=1; // Détecteur n°2 Ligne Blanche DETECT_LB2
TRISBbits.TRISB4=1; // Détecteur n°3 Ligne Blanche DETECT_LB3
TRISBbits.TRISB5=1; // Détecteur n°4 Ligne Blanche DETECT_LB4
TRISBbits.TRISB6=1; // Détecteur n°5 Ligne Blanche DETECT_LB5
TRISCbits.TRISC2=1; // Compteur moteur gauche COMPT_MOT_G
TRISDbits.TRISD1=1; // Erreur moteur gauche ERR_MOT_G
TRISCbits.TRISC5=1; // Compteur moteur droit COMPT_MOT_D
TRISDbits.TRISD2=1; // Erreur moteur droit ERR_MOT_D

/************************************
************************************
REGLAGE DES SORTIES NUMERIQUES
************************************
************************************/
TRISAbits.TRISA5=0; // LCD_RS
TRISEbits.TRISE0=0; // LCD_RW
TRISEbits.TRISE1=0; // LCD_E
TRISEbits.TRISE2=0; // LCD_DB4
TRISAbits.TRISA7=0; // LCD_DB5
TRISAbits.TRISA6=0; // LCD_DB6
TRISCbits.TRISC0=0; // LCD_DB7
TRISAbits.TRISA3=0; // Détecteur de sol (Emetteur) EMET_IR
TRISAbits.TRISA4=0; // CMD_LEDS
TRISCbits.TRISC1=0; // CMD_SON
TRISCbits.TRISC3=0; // CMD1_MOT_G
TRISDbits.TRISD0=0; // CMD2_MOT_G
TRISCbits.TRISC4=0; // CMD1_MOT_D
TRISDbits.TRISD3=0; // CMD2_MOT_D

/************************************
************************************
REGLAGE DES ENTREES ANALOGIQUES
************************************
************************************/
TRISAbits.TRISA0=1; //capteur lumière gauche CAPT_LUM_G
TRISAbits.TRISA1=1; //capteur lumière droite CAPT_LUM_D
TRISAbits.TRISA2=1; //capteur température CAPT_TEMP
/* NB : au démarrage/redémarrage, les broches PORTA<3:0>
sont des entrées analogiques par défaut. Il n'est donc
pas nécessaire de les configurer.*/
```

8.4. L'interface Homme-machine

8.4.1. Programmation relative au module LCD

La programmation du module LCD est la première chose que nous avons abordée, car il nous a semblé indispensable de pouvoir visualiser facilement la valeur de certaines variables internes lorsque le programme était en cours d'exécution, ou encore de savoir à tout instant à quelle étape se trouvait notre programme. Nous avons donc créé une bibliothèque contenant toutes les fonctions qui nous étaient indispensables pour gérer cet afficheur.

Le module LCD utilisé est un afficheur 2 lignes x 16 caractères. C'est grâce au très bon tutoriel en anglais fourni par notre vendeur que nous avons pu comprendre le fonctionnement d'un tel afficheur.

Il existe deux manières d'interfacer ce type d'afficheur à cristaux liquides avec un micro-contrôleur. Ceux-ci peuvent soit fonctionner en mode 8 bits, soit en mode 4 bits. Bien que le mode 8 bits soit plus rapide, nous avons préféré limiter le nombre de fils de connexion, nous avons donc opter pour une utilisation en mode 4 bits.

Mais avant de pouvoir utiliser l'afficheur, il est nécessaire de l'initialiser. Pour faire cela, il faut lui envoyer diverses commandes en respectant des temps d'attente.

Pour envoyer ces commandes, nous avons crée une fonction nommée *EnvoyerCommandeLCD :*

```
void EnvoyerCommandeLCD(char b7,char b6,char b5 ,char b4,char b3, char b2, char b1, char b0)
{
    LCD_E=0;
    LCD_RS  = 0;
    LCD_RW  = 0;
    LCD_DB7 = b7;
    LCD_DB6 = b6;
    LCD_DB5 = b5;
    LCD_DB4 = b4;
    Delay10KTCYx(1);
    LCD_E=1;
    Delay10KTCYx(20);
    LCD_E=0;
    LCD_DB7 = b3;
    LCD_DB6 = b2;
    LCD_DB5 = b1;
    LCD_DB4 = b0;
    Delay10KTCYx(1);
    LCD_E=1;
    Delay10KTCYx(20);
    LCD_E=0;
}
```

Grâce à cette fonction nous avons pu créer une fonction *InitialiserLCD,* fonction que nous appellerons dorénavant dans la fonction *InitialisationDuSysteme() :*

```
void InitialiserLCD(void)

{
    Delay10KTCYx(50);
    LCD_E=0;
    LCD_RS  = 0;
    LCD_RW  = 0;
    LCD_DB7 = 0;
    LCD_DB6 = 0;
    LCD_DB5 = 1;
    LCD_DB4 = 0;
    Delay10KTCYx(50);
    LCD_E=1;
    Delay10KTCYx(50);
    EnvoyerCommandeLCD(0,0,1,0,1,0,0,0);
    Delay10KTCYx(50);
    EnvoyerCommandeLCD(0,0,0,0,1,1,0,0); // curseur / blink
    Delay10KTCYx(50);
    EnvoyerCommandeLCD(0,0,0,0,0,0,0,1);
    Delay10KTCYx(50);
    EnvoyerCommandeLCD(0,0,0,0,0,0,1,1);
    Delay10KTCYx(50);
}
```

L'intégralité des fonctions que nous avons écrites pour gérer le module LCD est disponible en annexe de ce rapport. Nous signalerons pour terminer un dernier point important, concernant l'affichage des chaînes de caractères.

Il faut tout d'abord créer une variable globale de type ram char :

ram char MonBuffer[40];

… avant d'appeler la fonction d'écriture ainsi :

strcpypgm2ram (MonBuffer, "Moteur : Avance !");
EcrirePhraseLCD(MonBuffer);

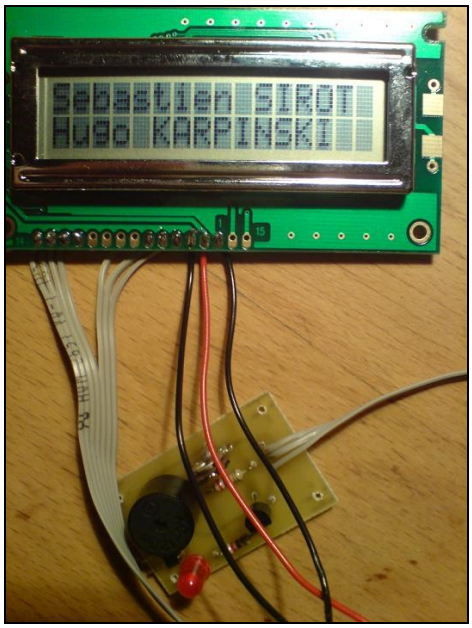


Illustration 23: affichage d'information sur un module LCD

8.4.2. Programmation relative aux modules 6 boutons

Pour détecter si un bouton est pressé, nous avons créé la fonction suivante dans **controledesboutons.h** :

```
unsigned char BoutonPresse(void)
{
    char bt;
    bt=0;
    if ((PORTD&0x80)==0x80) bt=1;
    if ((PORTD&0x40)==0x40) bt=2;
    if ((PORTD&0x20)==0x20) bt=3;
    if ((PORTD&0x10)==0x10) bt=4;
    if ((PORTC&0x80)==0x80) bt=5;
    if ((PORTC&0x40)==0x40) bt=6;
    return bt;
}
```

Cette fonction renvoie le numéro du bouton pressé. Attention, celle-ci ne gère par la pression de deux touches en même temps ou encore les éventuels rebonds des boutons pressé.

8.4.3. Programmation relative au module avertisseur sonore

Faute de temps, cette partie n'a pu être abordée avant la date limite.

8.4.4. Programmation relative au bouton rouge

Aucune programmation n'est nécessaire. Lorsque ce bouton est enfoncé, la broche MCLR est mise à l'état bas (0V) : le micro-contrôleur subit un redémarrage *hardware*.

8.4.5. Programmation relative au bouton noir

Faute de temps, cette partie n'a pu être abordée avant la date limite.

8.5. Les capteurs et détecteurs

8.5.1. Le Convertisseur Analogique-Numérique ou CAN

Afin d'utiliser les divers capteurs de notre robot, l'usage du CAN interne du micro-contrôleur est nécessaire. Nous devons tout d'abord crée une fonction d'initialisation :

```
void InitialiserCAN(void)
{
    ANSEL=0b00000111;
    ANSELH=0b00000000;  // sur les ports 2,3,4 uniquement
    ADCON1bits.VCFG1=0; // Voltage de ref : VSS
    ADCON1bits.VCFG0=0; // Voltage de ref : VDD
    ADCON2=0b10011101;  // Nous règlons le TAD, l'horloge de
                        // conversion à Fosc/16 et le résultat
                        // de la conv est justifié à droite
                        // (2 bits présent dans ADRESH et
                        // 8 bits présents dans ADRESL)
    ADCON0bits.ADON=1;  // Nous activons le CAN
}
```

Tout comme la fonction d'initialisation du LCD, cette fonction d'initialisation du CAN sera appelée en début de programme, dans la fonction *InitialisationDuSysteme()*

Maintenant, il suffit de créer une fonction pour récupérer la valeur convertie par le convertisseur. Cette valeur est codée sur 10 bits et se trouve partagée dans deux registres. Pour récupérer la valeur, il suffit de faire un masquage sur la première partie, puis de faire un décalage avant de l'additionner avec la seconde partie.

Voici notre fonction de récupération. Nous lui passons en paramètre la voie sur laquelle la mesure doit être faite, et il nous retourne le résultat (valeur comprise en 0 et 1023)

```
unsigned long LireCAN (unsigned char voie)
{
    unsigned int res;
    switch (voie)
    {
        case (1):   ADCON0=0b00000000;
        break;

        case (2):   ADCON0=0b00000100;
        break;

        case (3):   ADCON0=0b00001000;
        break;
        default :
        return 0;
    }
    ADCON0bits.ADON=1;  // Nous activons le CAN
    ADCON0bits.GO_DONE=1;
    while (ADCON0bits.GO_DONE!=0);
    ADCON0bits.ADON=0;  // Nous désactivons le CAN
    res=ADRESH&0x03; // recupère poids fort de res et mets sur poids faible de res
    res=res<<8;  // poids faible devient poids fort
    res=res+ADRESL; // complte avec le pids faible
    return res;

}
```

8.5.2. Programmation relative au capteur de température et de luminosité

Pour récupérer une valeur (comprise en 0 et 1023) image de la tension (et donc de la luminosité ou de la température), il suffit d'appeler la fonction précédente.

```
int ValeurTemperature(void)
{
    return LireCAN(3);
}

int ValeurLuminositeCapteurG(void)
{
    return LireCAN(1);
}

int ValeurLuminositeCapteurD(void)
{
    return LireCAN(2);
}
```

8.5.3. Programmation relative à la détection des chocs

Faute de temps, cette partie n'a pu être abordée avant la date limite.

8.5.4. Programmation relative au détecteur de la ligne blanche

Partie hors cahier des charges, non abordée dans notre rapport.

8.5.5. Programmation relative au détecteur de sol

Partie hors cahier des charges, non abordée dans notre rapport.

8.6. Le déplacement

8.6.1. Programmation relative à la commande des moteurs

De même, pour contrôler les moteurs de notre robot, ces fonctions simples suffisent.

```
void Avancer(void)
{
    CMD1_MOT_G=1;
    CMD2_MOT_G=0;
    CMD1_MOT_D=1;
    CMD2_MOT_D=0;
}

void Reculer(void)
{
    CMD1_MOT_G=0;
    CMD2_MOT_G=1;
    CMD1_MOT_D=0;
    CMD2_MOT_D=1;
}

void Stopper(void)
{
    CMD1_MOT_G=0;
    CMD2_MOT_G=0;
    CMD1_MOT_D=0;
    CMD2_MOT_D=0;
}

void TournerG(int angle)
{
    CMD1_MOT_G=0;
    CMD2_MOT_G=1;
    CMD1_MOT_D=1;
    CMD2_MOT_D=0;
    // Stopper();
}

void TournerD(int angle)
{
    CMD1_MOT_G=1;
    CMD2_MOT_G=0;
    CMD1_MOT_D=0;
    CMD2_MOT_D=1;
    // Stopper();
}
```

8.6.2. Programmation relative à la mesure de distance

Faute de temps, cette partie n'a pu être abordée avant la date limite.

9. Compte rendu des tests

	Commentaires
Module LCD	Longuement testé. Évidemment, parfaitement fonctionnel.
Carte 6 boutons	RAS, carte pleinement fonctionnelle testée avec le micro-contrôleur.
Carte avertisseur sonore	Testée avec GBF et micro-contrôleur : fonctionnement OK. Sirène modulée très facilement obtenue en faisant varier la fréquence du signal MLI
Carte d'alimentation	Testée. Tout fonctionne : possibilité d'avoir de 1,5V à Vbat environ sur chaque sortie
Carte capteur et détecteur	Capteurs de luminosité OK. Signal du capteur de température correctement amplifié par l'AOP. Allumage/extinction des leds blanches fonctionnelles. Reste de la carte non testé .
Cartes contrôleur (moteur DC)	Commande des moteurs fonctionnant parfaitement mais trigger de Shmitt (compteur) apparemment non fonctionnel. Pas encore cherché la raison de la panne..
Carte détecteur de sol	Non testée.
Carte micro-contrôleur	Testée, fonctionnement OK.

Toutes les cartes ont pu être réunies et soudées ensemble le 25 mars. Durant quelques heures, nous avons pu effectuer quelques tests. A l'exception de la fonction compteur des cartes de contrôle moteur, tout ce que nous avons testé a fonctionné pour le mieux (affichage de la température, émission de « bips » lorsque la luminosité était trop faible sur l'un des capteurs, marche des moteurs dans le sens horaire et anti-horaire, arrêt des moteurs, sirène, contrôle des leds blanches, boutons,etc...).

10. Planning prévisionnel et réel

N° de semaine	3	4	5	6	7	8	9	10	11	12	13
Définition du projet : cahiers des charges et planning											
Conception de la partie électronique											
Étude du micro-contrôleur PIC et du PICKIT 3											
Programmation du micro-contrôleur											
Assemblage											
Rédaction du rapport											
Remise du rapport											

Prévisionnel

Réel

L'apprentissage de la programmation du PIC à l'aide de MPLAB nous a pris plus de temps que prévu. Ceci est en grande partie du à la documentation en anglais. Il nous a fallu comprendre certaines choses nouvelles par nous-mêmes. Le retard pris lors de cette durée d'apprentissage de la programmation s'est répercuté et n'a jamais réellement pu être rattrapé.

De même, la réalisation de la partie électronique a été plus longue que nous ne l'aurions pensé. Ceci s'explique en partie parce que des fonctions supplémentaires ont été rajoutées au robot en cours de route, mais aussi, et surtout, parce que nous n'avons pas repris des cartes existantes, nous avons dû réaliser nos circuits et nos cartes de A à Z avec WinCircuit2008. Cette étape de notre

projet nous a donc également pris beaucoup de temps.

11. Fiche de suivi de projet

Intitulé du projet	Étudiants
Robot explorateur	**Hugo KARPINSKI, Sébastien SIROT**

Date	Commentaires
19/01/10	Choix du sujet et étude afin de choisir les solutions développées plus tard. Définition du cahier des charges et du planning. Compréhension de la structure générale de mémoire du PIC.
26/01/10	Début des leçons sur Pic kit : allumage d'une LED, clignotement.
02/02/10	Réalisation d'un chenillard (LED qui clignotent successivement).Apprentissage de l'utilisation des « Timer »
26/02/10	Apprentissage de la gestion des interruptions.
02/03/10	Mise en place des interruptions qui permettent de calculer la distance parcourue et de gérer les collisions.
09/03/10	Typons des cartes Avertisseur sonore et 6 boutons.
18/03/10	Typon des cartes capteurs&détecteur, micro-contrôleur et contrôle moteur. Réalisation de cartes. Début des tests et début de la programmation.
26/03/10	Remise du rapport.

12. Le système terminé

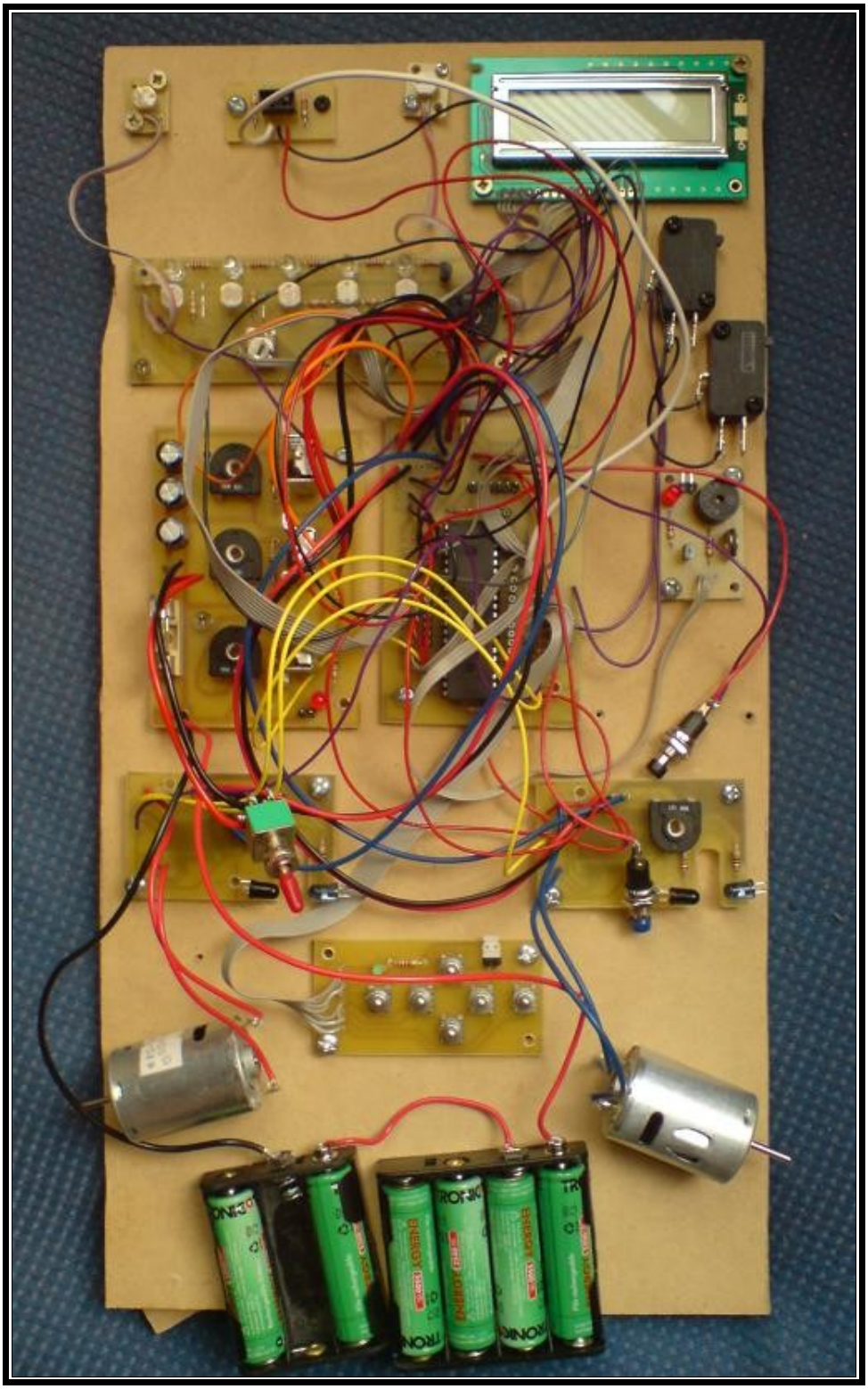


Illustration 24: le système lors de nos tests finaux

Conclusion

Au cours de nos séances de TP, nous avons découvert la programmation des micro-contrôleurs PIC. Nous avons utilisé un programmateur PICKIT3, accompagné de sa carte d'essai d'origine, pour suivre le tutoriel proposé par le constructeur Microchip.

Après avoir acquis des connaissances suffisantes pour la partie programmation de notre projet, nous avons abordé la conception électronique, en tenant compte des spécificités des PIC18F4XK20. Nous nous sommes servis du logiciel WinCircuit2008 pour concevoir les typons de nos cartes. Malheureusement, nous avons passé trop de temps sur la compréhension du PIC et de son compilateur, ce qui nous a empêché de bien réaliser les parties électroniques et mécaniques du robot.

Toutefois, nous sommes parvenu à réaliser un système presque complet, en poursuivant notre travail en dehors des heures de cours.

Ce projet nous apporté de nouvelles compétences dans le domaine de l'informatique industrielle (paramétrage des périphériques internes, interfaçage avec des modules externes,...) ainsi que dans le domaine de l'électronique des capteurs (température, luminosité, Infra-Rouge,...).

Bibliographie

[1] Documentation technique, Microchip Technology Inc., *PICkit 3 Debug Express Lessons User's Guide, PIC18F45K20 – MPLAB® C Lessons,*2009

[2] Documentation technique, Microchip Technology Inc., *PIC18F23K20/24K20/25K20/26K20/43K20/44K20/45K20/46K20 Data Sheet 28/40/44-Pin Flash Microcontrollers with nanoWatt XLP Technology*, 2010

[3] Julyan Ilett, *How to use intelligent LCDs – part one*, Everyday practical Electronics magazine, février 1997.

[4] Julyan Ilett, *How to use intelligent LCDs – part two*, Everyday practical Electronics magazine, mars 1997.

Tous les documents techniques, autres que ceux cités précédemment, relatifs aux composants, ont été trouvés sur le site internet http://*www.alldatasheet.com*

Index des illustrations

Index des tables

ANNEXES

Avertissement

Le code suivant a été écrit quelques jours avant la présentation orale de notre projet. Il avait pour seul but de tester le bon fonctionnement de certaines parties de notre système. Nous le présentons en annexe uniquement pour montrer le travail que nous avons pu effectuer sur la partie programmation durant les derniers jours, lorsque nous avions enfin terminé la partie électronique de notre projet.

NB : bien que présenté ici, le dernier fichier, relatif à l'avertisseur sonore, n'a pas été testé.

Annexe n° 1 : copie du fichier du programme principal (fichier .c)

```
/*************************************************
*************************************************
*************************************************
*************************************************
*************************************************

                 -------------------------
                | PROJET ROBOT EXPLORATEUR |
                 -------------------------

                      Projet de 2ème année
                      IUT de Tours
                      Promotion 2008-2010

    Etudiants : Hugo KARPINSKI
                      Sébastien SIROT

*************************************************
*************************************************
*************************************************
*************************************************
*************************************************/

/*
    PROGRAMME PRINCIPAL
*/


/***********************************
***********************************
Les fichiers inclus
***********************************
***********************************/

#include "p18f46k20.h"     // Nous incluons le fichier d'entête du micro-contrôleur
#include "controledulcd.h"
#include "controleduson.h"
#include "controledudeplacement.h"
#include "controledescapteursetdesdetecteurs.h"
#include "controledesboutons.h."
```

```
#include "string.h"


/**************************************
**************************************
Configuration des bits de configuration
**************************************
**************************************/

// Nous choisissons l'oscillateur interne comme horloge
// et nous définissons RA6 et RA7 comme des ports I/O.
// Ceux-ci sont utilisés pour le contrôle du LCD.
#pragma config FOSC = INTIO67

// Le RESET externe est mis à disposition (RE3 inutilisable)
// Ainsi, on pourra forcer un redémarrage du programme, en appuyant
// sur un bouton rouge par exemple.
#pragma config MCLRE = ON

// Nous désactivons la fonction "entrée analogique" sur les bits PORTB<4:0>
#pragma config PBADEN = OFF

// Nous multiplexons le signal MLI de CCP2 avec la broche RC1
// (et non avec la broche RB2). La broche RC1 servira peut-être de
// commande au module "Avertisseur sonore", car il est très facile
// de faire varier la fréquence de CCP2 via un registre. Toutefois,
// peut-être utiliserons nous une autre méthode pour obtenir
// un signal de fréquence variable sur la sortie RC1...
#pragma config CCP2MX = PORTC

// Nous désactivons la plupart des périphériques avancés
#pragma config FCMEN = ON, IESO = OFF, PWRT = OFF, BOREN = OFF
#pragma config HFOFST = OFF, LPT1OSC = OFF, WDTEN = OFF

// Nous désactivons la protection du code et permettons l'accès à l'EEPROM

#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF
#pragma config CPB = OFF, CPD = OFF
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF
#pragma config WRTB = OFF, WRTC = OFF, WRTD = OFF
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF, EBTRB =
OFF

void InitialisationDuSysteme(void);

ram char MonBuffer[40];


void main (void)

{
 unsigned int compt;
    char a=0;
    int i,o;
    long g;
    g=0;
    o=1;
    compt=0;

    InitialisationDuSysteme();
```

```
    while (1)
    {


if ((BoutonPresse()==1)&(o!=1))
{
    o=1;
    EffacerLCD();
    JouerUnSon(4,5);
}
if ((BoutonPresse()==2)&(o!=2))
{
    o=2;
    EffacerLCD();
    JouerUnSon(3,5);
}


if ((BoutonPresse()==3)&(o!=3))
{
    o=3;
    EffacerLCD();
    strcpypgm2ram (MonBuffer, "LUMIERE");
    EcrirePhraseLCD(MonBuffer);
    JouerUnSon(5,5);
}

if ((BoutonPresse()==4)&(o!=4))
{
    o=4;
    EffacerLCD();
    strcpypgm2ram (MonBuffer, "MOTEUR GAUCHE");
    EcrirePhraseLCD(MonBuffer);
    JouerUnSon(5,5);
}
if ((BoutonPresse()==5)&(o!=5))
{
    o=5;
    EffacerLCD();
    strcpypgm2ram (MonBuffer, "MOTEUR DROIT");
    EcrirePhraseLCD(MonBuffer);
    JouerUnSon(5,5);
}


    if (o==1)
    EcrireLuminosite(ValeurLuminositeCapteurG(),ValeurLuminositeCapteurD());
    else
    if (o==2)
    {
    PremiereLigne();
    strcpypgm2ram (MonBuffer1,"Temp. ambiante :");
    EcrirePhraseLCD(MonBuffer1);
    AllerA(2,9);
    EcrireTemperature (ValeurTemperature());
    Delay10KTCYx(50);
    }
    else
    if (o==3)
    {
          if (CMD_LEDS) EteindreLEDs(); else AllumerLEDs ();
```

```
        Delay10KTCYx(100);
    }

    else
    if (o==4)
    {
        if (CMD2_MOT_G==0) TournerG(5); else Stopper();
        Delay10KTCYx(100);
    }
    else
    if (o==5)
    {
        if (CMD1_MOT_G==0) TournerD(5); else Stopper();
        Delay10KTCYx(100);
    }
    o=0;
}
}


void InitialisationDuSysteme(void)
{

    /************************************
    ************************************
    REGLAGE DE L'HORLOGE PRINCIPAL
    ************************************
    ************************************/
    OSCCONbits.IRCF0=0;           // Nous réglons les bits du Registre IRCF
    OSCCONbits.IRCF1=1;           // pour imposer une fréquence d'horloge
    OSCCONbits.IRCF2=1;           // de 16 MHz.
    OSCTUNEbits.PLLEN=0;    // Nous désactivons la PLL (qui multiplierait
                                  // sinon par 4 l'horloge interne)

    /************************************
    ************************************
    REGLAGE DES ENTREES NUMERIQUES
    ************************************
    ************************************/
    TRISBbits.TRISB7=1;     // Bouton noir
    TRISCbits.TRISC6=1;     // Bouton 1 BT1
    TRISCbits.TRISC7=1;     // Bouton 2 BT2
    TRISDbits.TRISD4=1;     // Bouton 3 BT3
    TRISDbits.TRISD5=1;     // Bouton 4 BT4
    TRISDbits.TRISD6=1;     // Bouton 5 BT5
    TRISDbits.TRISD7=1;     // Bouton 6 BT6
    TRISBbits.TRISB0=1; // Détecteur de choc DETECT_CHOC
    TRISBbits.TRISB1=1;     // Détecteur de sol (Récepteur) RECEPT_IR
    TRISBbits.TRISB2=1;     // Détecteur n°1 Ligne Blanche DETECT_LB1
    TRISBbits.TRISB3=1;     // Détecteur n°2 Ligne Blanche DETECT_LB2
    TRISBbits.TRISB4=1;     // Détecteur n°3 Ligne Blanche DETECT_LB3
    TRISBbits.TRISB5=1;     // Détecteur n°4 Ligne Blanche DETECT_LB4
    TRISBbits.TRISB6=1;     // Détecteur n°5 Ligne Blanche DETECT_LB5
    TRISCbits.TRISC2=1;     // Compteur moteur gauche COMPT_MOT_G
    TRISDbits.TRISD1=1;     // Erreur moteur gauche ERR_MOT_G
    TRISCbits.TRISC5=1;     // Compteur moteur droit COMPT_MOT_D
    TRISDbits.TRISD2=1;     // Erreur moteur droit ERR_MOT_D

    /************************************
    ************************************
    REGLAGE DES SORTIES NUMERIQUES
```

```
    ************************************
    ************************************/
    TRISAbits.TRISA5=0;     // LCD_RS
    TRISEbits.TRISE0=0;     // LCD_RW
    TRISEbits.TRISE1=0;     // LCD_E
    TRISEbits.TRISE2=0;     // LCD_DB4
    TRISAbits.TRISA7=0;     // LCD_DB5
    TRISAbits.TRISA6=0;     // LCD_DB6
    TRISCbits.TRISC0=0;     // LCD_DB7
    TRISAbits.TRISA3=0;     // Détecteur de sol (Emetteur) EMET_IR
    TRISAbits.TRISA4=0;     // CMD_LEDS
    TRISCbits.TRISC1=0;     // CMD_SON
    TRISCbits.TRISC3=0;     // CMD1_MOT_G
    TRISDbits.TRISD0=0;     // CMD2_MOT_G
    TRISCbits.TRISC4=0;     // CMD1_MOT_D
    TRISDbits.TRISD3=0;     // CMD2_MOT_D

    /************************************
    ************************************
    REGLAGE DES ENTREES ANALOGIQUES
    ************************************
    ************************************/
    TRISAbits.TRISA0=1;     //capteur lumière gauche CAPT_LUM_G
    TRISAbits.TRISA1=1;     //capteur lumière droite CAPT_LUM_D
    TRISAbits.TRISA2=1;     //capteur température CAPT_TEMP
    /* NB : au démarrage/redémarrage, les broches PORTA<3:0>
    sont des entrées analogiques par défaut. Il n'est donc
    pas nécessaire de les configurer.*/

    /************************************
    ************************************
    AUTRES REGLAGES
    ************************************
    ************************************/
    EteindreLEDs(); // Nous nous assurons que les LEDs blanches
                            // soient éteintes au démarrage
    EMET_IR = 0;      // De même que l'émetteur Infra-rouge
    PORTAbits.RA3=1;
    etat=0;
    compt=0;


    /************************************
    ************************************
    Initialisation du LCD et du CAN
    ************************************
    ************************************/
    InitialiserLCD();
    InitialiserCAN();

    /************************************
    ************************************
    Jouer une musique au démarrage
    ************************************
    ************************************/
    JouerMusiqueDemarrage();
    Stopper();
}
```

Annexe n° 2 : copie du fichier controledulcd.h

```
/*
    FONCTIONS RELATIVES A L'UTILISATION DU LCD
*/

// Nous déclaration de mnémoniques pour
// faciliter l'accès aux broches du LCD
#define LCD_RS  LATAbits.LATA5
#define LCD_RW  LATEbits.LATE0
#define LCD_E   LATEbits.LATE1
#define LCD_DB4 LATEbits.LATE2
#define LCD_DB5 LATAbits.LATA7
#define LCD_DB6 LATAbits.LATA6
#define LCD_DB7 LATCbits.LATC0

#include "delays.h"
#include "stdlib.h"
#include "string.h"


  void EnvoyerCommandeLCD(char b7,char b6,char b5 ,char b4,char b3, char b2,
char b1, char b0);
  void EnvoyerCaractereLCD(char b7,char b6,char b5 ,char b4,char b3, char b2,
char b1, char b0);

  void InitialiserLCD(void);
  void EffacerLCD(void);
  void EcrireCaractereLCD(char caractere);
  void EcrirePhraseLCD (char * chaine);
  void PremiereLigne(void);
  void SecondeLigne(void);
  void AllerA(char ligne, char colonne);
  void EcrireValeurLCD (long val);
  void EcrireTemperature (long val);
  void EcrireLuminosite (int valG, int valD);

  ram char MonBuffer1[40];
  int memoG=0;
  int memoD=0;



  void EnvoyerCommandeLCD(char b7,char b6,char b5 ,char b4,char b3, char b2,
char b1, char b0)
  {
      LCD_E=0;
      LCD_RS      = 0;
      LCD_RW      = 0;
      LCD_DB7     = b7;
      LCD_DB6     = b6;
      LCD_DB5     = b5;
      LCD_DB4 = b4;
      Delay100TCYx(10);
      LCD_E=1;
      Delay1KTCYx(3);
      LCD_E=0;
      LCD_DB7     = b3;
      LCD_DB6     = b2;
```

```
    LCD_DB5     = b1;
    LCD_DB4 = b0;
    Delay100TCYx(10);
    LCD_E=1;
    Delay1KTCYx(3);
    LCD_E=0;
}

  void EnvoyerCaractereLCD(char b7,char b6,char b5 ,char b4,char b3, char b2,
char b1, char b0)
  {
    LCD_E=0;
    LCD_RS      = 1;
    LCD_RW      = 0;
    LCD_DB7     = b7;
    LCD_DB6     = b6;
    LCD_DB5     = b5;
    LCD_DB4 = b4;
    Delay100TCYx(10);
    LCD_E=1;
    Delay1KTCYx(3);
    LCD_E=0;
    LCD_DB7     = b3;
    LCD_DB6     = b2;
    LCD_DB5     = b1;
    LCD_DB4 = b0;
    Delay100TCYx(10);
    LCD_E=1;
    Delay1KTCYx(3);
    LCD_E=0;
  }

  void InitialiserLCD(void)

  {
     Delay10KTCYx(500);
     LCD_E=0;
    LCD_RS      = 0;
    LCD_RW      = 0;
    LCD_DB7     = 0;
    LCD_DB6     = 0;
    LCD_DB5     = 1;
    LCD_DB4 = 0;
    Delay100TCYx(10);
    LCD_E=1;

    Delay10KTCYx(20);
    EnvoyerCommandeLCD(0,0,1,0,1,0,0,0);
    Delay10KTCYx(20);
    EnvoyerCommandeLCD(0,0,0,0,1,1,0,0); // curseur / blink
    Delay10KTCYx(20);
    EnvoyerCommandeLCD(0,0,0,0,0,0,0,1);
    Delay10KTCYx(20);
    EnvoyerCommandeLCD(0,0,0,0,0,0,1,1);
    Delay10KTCYx(20);
  }

  void EffacerLCD(void)
  {
    EnvoyerCommandeLCD(0,0,0,0,0,0,0,1);
  }
```

```
void EcrireCaractereLCD(char caractere)
{
    char b0=0,b1=0,b2=0,b3=0,b4=0,b5=0,b6=0,b7=0;
    LCD_E=0;
    LCD_RS      = 1;
    LCD_RW      = 0;
    b4 = (caractere & 0x10)>>4;
    b5 = (caractere & 0x20)>>5;
    b6 = (caractere & 0x40)>>6;
    b7 = (caractere & 0x80)>>7;
    LCD_DB7     = b7;
    LCD_DB6     = b6;
    LCD_DB5     = b5;
    LCD_DB4 = b4;
    Delay100TCYx(10);
    LCD_E=1;
    Delay1KTCYx(3);
    LCD_E=0;
    b0 = (caractere & 0x01);
    b1 = (caractere & 0x02)>>1;
    b2 = (caractere & 0x04)>>2;
    b3 = (caractere & 0x08)>>3;
    LCD_DB7     = b3;
    LCD_DB6     = b2;
    LCD_DB5     = b1;
    LCD_DB4 = b0;
    Delay100TCYx(10);
    LCD_E=1;
    Delay1KTCYx(3);
    LCD_E=0;
}
void EcrirePhraseLCD (char * chaine)
{
    while(* chaine != '\0')
    {
          EcrireCaractereLCD(*chaine);
          *chaine ++;
    }
}

void PremiereLigne(void)
{
    EnvoyerCommandeLCD(0,0,0,0,0,0,1,1);
}

void SecondeLigne (void)
{
    EnvoyerCommandeLCD(1,1,0,0,0,0,0,0);
}

void AllerA(char ligne, char colonne)
{
    char i=0,b0=0,b1=0,b2=0,b3=0,b4=0,b5=0,b6=0;
    if ((colonne<=40)&&(ligne<=2))
    {
          if (ligne==2) i=(colonne-1)+64; else i=colonne-1;
          b6=(i&0x40)>>6;
          b5=(i&0x20)>>5;
          b4=(i&0x10)>>4;
          b3=(i&0x08)>>3;
          b2=(i&0x04)>>2;
```

```
            b1=(i&0x02)>>1;
            b0=(i&0x01);
            EnvoyerCommandeLCD(1,b6,b5,b4,b3,b2,b1,b0);
        }
        else PremiereLigne;
    }

    void EcrireValeurLCD (long val)
    {
        char * ha[20];
        ltoa(val, * ha);
        EcrirePhraseLCD(* ha);
    }

    void EcrireTemperature (long val)
    {
        char * a[20];
        char * b[20];
        int x;
        x=0;
        //1023 correspond à 60° c'est à dire 3,3V
        //val correspond à X
        x=(int)((val*60)/1023); // Nous prennons la partie entière
                                            // de la conversion
        ltoa(x, * a);
        EcrirePhraseLCD(* a);   // Nous affichons la partie entière
        EcrireCaractereLCD('.');// Nous mettons le point

        x=(int)(10.0*(  (  val- ((float)(x))*1023.0/60.0  ) *60.0/1023.0)) ;
        //EcrireCaractereLCD('C');
        ltoa(x, * b);
        EcrirePhraseLCD(* b);   // Nous affichons la partie décimale

        EnvoyerCaractereLCD(1,1,0,1,1,1,1,1);    //    Nous      utilisons      une
ponctuation
                                                                    //    de
ponctuation japonaise pour
                                                                    //    afficher
le signe degré : °
        EcrireCaractereLCD('C');
    }

    void EcrireLuminosite (int valG, int valD)
    {
        char a,i;
        strcpypgm2ram (MonBuffer1,"Gau ");
        EcrirePhraseLCD(MonBuffer1);
        i=0;
        if (valG<=79) i=0;
        if ((valG>79)&(valG<=158)) i=1;
        if ((valG>157)&(valG<=236)) i=2;
        if ((valG>236)&(valG<=314)) i=3;
        if ((valG>314)&(valG<=393)) i=4;
        if ((valG>393)&(valG<=472)) i=5;
        if ((valG>472)&(valG<=551)) i=6;
        if ((valG>551)&(valG<=630)) i=7;
        if ((valG>630)&(valG<=708)) i=8;
        if ((valG>708)&(valG<=787)) i=9;
        if ((valG>787)&(valG<=866)) i=10;
        if ((valG>866)&(valG<=944)) i=11;
        if (valG>944) i=12;
```

```
if (valG>memoG)
{
      for (a=0;a<i;a++)
      {
            EnvoyerCaractereLCD(1,1,1,1,1,1,1,1);
      }
      for (a=0;a<(12-i);a++)
      {
            EnvoyerCaractereLCD(1,1,0,1,1,0,1,1);
      }
}
else
{

      for (a=0;a<(12-i);a++)
      {
            AllerA(1,16-a);
            EnvoyerCaractereLCD(1,1,0,1,1,0,1,1);

      }
      for (a=0;a<i;a++)
      {
            AllerA(1,4+i-a);
            EnvoyerCaractereLCD(1,1,1,1,1,1,1,1);
      }
}

i=0;
SecondeLigne();
strcpypgm2ram (MonBuffer1,"Dro ");
EcrirePhraseLCD(MonBuffer1);

if (valD<=79) i=0;
if ((valD>79)&(valD<=158)) i=1;
if ((valD>157)&(valD<=236)) i=2;
if ((valD>236)&(valD<=314)) i=3;
if ((valD>314)&(valD<=393)) i=4;
if ((valD>393)&(valD<=472)) i=5;
if ((valD>472)&(valD<=551)) i=6;
if ((valD>551)&(valD<=630)) i=7;
if ((valD>630)&(valD<=708)) i=8;
if ((valD>708)&(valD<=787)) i=9;
if ((valD>787)&(valD<=866)) i=10;
if ((valD>866)&(valD<=944)) i=11;
if (valD>944) i=12;

if (valD>memoD)
{
      for (a=0;a<i;a++)
      {
            EnvoyerCaractereLCD(1,1,1,1,1,1,1,1);
      }
      for (a=0;a<(12-i);a++)
      {
            EnvoyerCaractereLCD(1,1,0,1,1,0,1,1);
      }
}
else
{
```

```
            for (a=0;a<(12-i);a++)
            {
                  AllerA(2,16-a);
                  EnvoyerCaractereLCD(1,1,0,1,1,0,1,1);

            }
            for (a=0;a<i;a++)
            {
                  AllerA(2,4+i-a);
                  EnvoyerCaractereLCD(1,1,1,1,1,1,1,1);
            }

      }
      memoG=valG;
      memoD=valD;
      PremiereLigne();
  }
  }
```

Annexe n° 3 : copie du fichier controledescapteursetdesdetecteurs.h

```
/*
      FONCTIONS RELATIVES A L'UTILISATION
      DES CAPTEURS ET DES DETECTEURS DU ROBOT
*/

/***********************************
***********************************
Déclaration des mnémoniques
***********************************
***********************************/

// Capteurs de lumière et de température
#define CAPT_LUM_G      LATAbits.LATA0
#define CAPT_LUM_D      LATAbits.LATA1
#define CAPT_TEMP LATAbits.LATA2

// Détecteur IR - Couple émetteur et récepteur infrarouge IR
#define EMET_IR         LATAbits.LATA3
#define RECEPT_IR LATBbits.LATB1

// Commande des leds blanches du détecteur de ligne
#define CMD_LEDS  LATAbits.LATA4

// Les 5 détecteurs du detecteur de ligne
#define DETECT_LB1      LATBbits.LATB2
#define DETECT_LB2      LATBbits.LATB3
#define DETECT_LB3      LATBbits.LATB4
#define DETECT_LB4      LATBbits.LATB5
#define DETECT_LB5      LATBbits.LATB6

// Le détecteurs de chocs
#define DETECT_CHOC LATBbits.LATB0

void InitialiserCAN(void);
int LireCAN (unsigned char voie);
void AllumerLEDs (void);
void EteindreLEDs (void);
```

```
void DetecterSol (void);
int ValeurTemperature(void);
int ValeurLuminositeCapteurG(void);
int ValeurLuminositeCapteurD(void);
char ValeurCapteurLigneNumero(char n);

void InitialiserCAN(void)
{
      ANSEL=0b00000111;
      ANSELH=0b00000000;      // sur les ports 2,3,4 uniquement
      ADCON1bits.VCFG1=0;     // Voltage de ref : VSS
      ADCON1bits.VCFG0=0;     // Voltage de ref : VDD
      ADCON2=0b10011101;      // Nous règlons le TAD, l'horloge de
                                    // conversion à Fosc/16 et le résultat
                                    // de la conv est justifié à droite
                                    // (2 bits présent dans ADRESH et
                                    // 8 bits présents dans ADRESL)
      ADCON0bits.ADON=1;      // Nous activons le CAN

}

int LireCAN (unsigned char voie)
{
      long res;
      res=0;
      switch (voie)
      {
            case (1):   ADCON0=0b00000000;
            break;

            case (2):   ADCON0=0b00000100;
            break;

            case (3):   ADCON0=0b00001000;
            break;
            default :
            return 0;
      }
      ADCON0bits.ADON=1;      // Nous activons le CAN
      ADCON0bits.GO_DONE=1;
      while (ADCON0bits.GO_DONE!=0);
      ADCON0bits.ADON=0;      // Nous désactivons le CAN
      res=ADRESH&0x03; // recupère poids fort de res et mets sur poids faible de
res
      res=res<<8;  // poids faible devient poids fort
      res=res+ADRESL; // complte avec le pids faible
      return res;

}

void AllumerLEDs (void)
{
      CMD_LEDS = 1;
}

void EteindreLEDs (void)
{
      CMD_LEDS = 0;
}

void DetecterSol (void)
```

```
{
      EMET_IR = 1;
}

int ValeurTemperature(void)
{
      return LireCAN(3);
}

int ValeurLuminositeCapteurG(void)
{
      return LireCAN(1);
}

int ValeurLuminositeCapteurD(void)
{
      return LireCAN(2);
}
char ValeurCapteurLigneNumero(char n)
{
}
```

Annexe n° 4 : copie du fichier controledesboutons.h

```
/*
    FONCTIONS RELATIVES A L'UTILISATION DES BOUTONS
*/

// Nous déclaration de mnémotechniques pour
// faciliter l'accès aux boutons

#define BT1     LATDbits.LATD7
#define BT2     LATDbits.LATD6
#define BT3     LATDbits.LATD5
#define BT4     LATDbits.LATD4
#define BT5     LATCbits.LATC7
#define BT6     LATCbits.LATC6

unsigned char BoutonPresse(void);

unsigned char BoutonPresse(void)
{
    char bt;
    bt=0;
    if ((PORTD&0x80)==0x80) bt=1;
    if ((PORTD&0x40)==0x40) bt=2;
    if ((PORTD&0x20)==0x20) bt=3;
    if ((PORTD&0x10)==0x10) bt=4;
    if ((PORTC&0x80)==0x80) bt=5;
    if ((PORTC&0x40)==0x40) bt=6;
    return bt;
}
```

Annexe n° 5 : copie du fichier controledudeplacement.h

```
/*
    FONCTIONS RELATIVES AU DEPLACEMENT DU ROBOT
*/

/************************************
************************************
Déclaration des mnémoniques
************************************
************************************/

// Moteur droit
#define CMD1_MOT_D   LATDbits.LATD3    //    Commande    n°1    moteur    droit
CMD1_MOT_D
#define CMD2_MOT_D   LATCbits.LATC4    //    Commande    n°2    moteur    droit
CMD2_MOT_D
#define COMPT_MOT_D  LATCbits.LATC5    //      Compteur      moteur      droit
    COMPT_MOT_D
#define ERR_MOT_D    LATDbits.LATD2    // Erreur moteur droit  ERR_MOT_D

// Moteur gauche
#define CMD1_MOT_G   LATCbits.LATC3    //    Commande    n°1    moteur    gauche
CMD1_MOT_G
#define CMD2_MOT_G   LATDbits.LATD0    //    Commande    n°2    moteur    gauche
CMD2_MOT_G
#define COMPT_MOT_G  LATCbits.LATC2    //      Compteur      moteur      gauche
    COMPT_MOT_G
#define ERR_MOT_G    LATDbits.LATD1    // Erreur moteur gauche ERR_MOT_G

void Avancer(void);
void Reculer(void);
void Stopper(void);
void TournerG(int angle);
void TournerD(int angle);
void MettreAJourCompteurTotal(void);

char etat;
unsigned long compt;

void Avancer(void)
{
    CMD1_MOT_G=1;
    CMD2_MOT_G=0;
    CMD1_MOT_D=1;
    CMD2_MOT_D=0;
}

void Reculer(void)
{
    CMD1_MOT_G=0;
    CMD2_MOT_G=1;
    CMD1_MOT_D=0;
    CMD2_MOT_D=1;
}

void Stopper(void)
{
    CMD1_MOT_G=0;
```

```
    CMD2_MOT_G=0;
    CMD1_MOT_D=0;
    CMD2_MOT_D=0;
}

void TournerG(int angle)
{
    CMD1_MOT_G=0;
    CMD2_MOT_G=1;
    CMD1_MOT_D=1;
    CMD2_MOT_D=0;
    // Stopper();
}

void TournerD(int angle)
{
    CMD1_MOT_G=1;
    CMD2_MOT_G=0;
    CMD1_MOT_D=0;
    CMD2_MOT_D=1;
    // Stopper();
}

void MettreAJourCompteurTotal(void)
{
    if ((COMPT_MOT_D==1) & etat==0)
    {
          compt=compt+1;
          etat=1;
    }
    else if ((COMPT_MOT_D==0) & etat==1) etat=0;
}
```

Annexe n° 6 : copie du fichier controleduson.h

```
#define CMD_SON LATCbits.LATC1

typedef enum {DO,RE,MI,FA,SO,LA,SI}
                                          Notes;

void ChangerTempo(char tempo);
void JouerUnSon (char m, long duree);
void JouerMusiqueDemarrage(void);

void ChangerTempo(char tempo)
{

}

void JouerUnSon (char notes, long duree)
{
    long freq; // 0  16,777,215
    long i;
```

```
        int compt=0;
        int g;

        switch (notes)
        {
                case 1 : freq=500000; break;
                case 2 : freq=800135; break;
                case 3 : freq=8008; break;
                case 4 : freq=32703; break;
                case 5 : freq=346; break;
                case 6 : freq=36708; break;
                case 7 : freq=38891; break;
        }

        g=(int)(100000/freq);

        do
        {
                CMD_SON=!CMD_SON;
                for (i=0;i<g;i++)
                Nop();
                compt++;
        }
        while (compt<=(freq/1000));
    }


    void JouerMusiqueDemarrage(void)
    {
    }
}
```