

## TP transdisciplinaire en EEA Robot suiveur de ligne

### 1 Objectif

L'objet du TP longue durée transdisciplinaire est d'associer vos connaissances en électronique, en automatique et en programmation autour d'un projet de développement de robot autonome capable de suivre une trajectoire matérialisée par une ligne noire non nécessairement rectiligne tracée au sol (cette ligne est créée à l'aide de ruban adhésif standard). A tout moment, ce robot doit être capable de stopper son avancée s'il détecte un obstacle sur son chemin. Ce robot, construit sur la base d'une plateforme TRAXSTER II est présenté sur la figure 1.

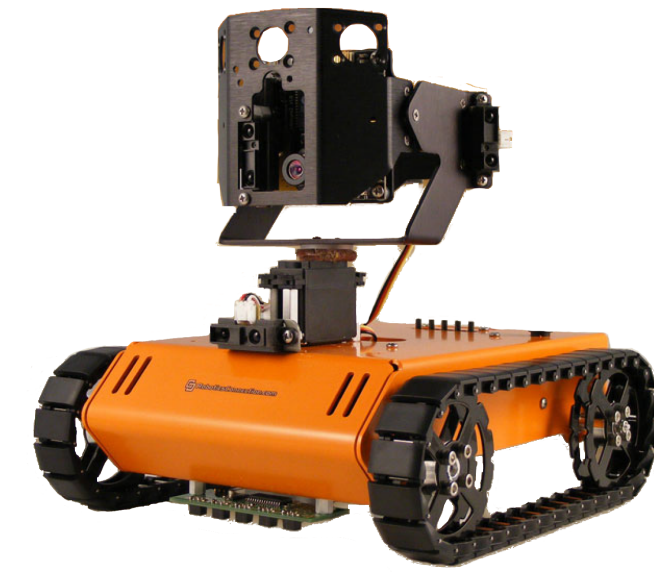


FIG. 1 : La plateforme roulante Traxster II

Ce document constitue une feuille de route destinée à fournir un support aux volets conception et programmation du projet. Il est découpé en 4 parties :

- principe de fonctionnement d'un microcontrôleur et de ses périphériques ;
- conception et programmation de l'acquisition et du traitement des données **capteurs** ;
- conception et programmation de la **commande des moteurs** ;
- conception et programmation d'une **l'interface homme-machine** et pilotage du robot.

La première partie (sections 2 et 3) est commune à l'ensemble du projet : elle doit être lue avec attention et comprise **dès la première séance de la matière**. Les sections 4, 5 et 6 constituent en revanche des sous-projets spécifiques et quasiment indépendants qui seront affectés lors de la première séance aux différents binômes du groupe. Toutefois, la lecture de la totalité du fascicule aidera à positionner chaque sous-projet dans son contexte global et n'est à ce titre pas facultative.

Lors de la première séance, chaque binôme du groupe se verra attribué la charge de réaliser un de ces sous-projets dans les 28 heures imparties. Le découpage du cahier des charges en trois sous-projets a pour objectif :

- de structurer le travail de chaque binôme et de le responsabiliser vis à vis des autres binômes et de l'enseignant ;
- de définir des interfaces minimales mais précises entre projets afin d'assurer leur indépendance mutuelle mais aussi de constituer des points de collaboration entre binômes de projets différents ;

- de répartir la charge de travail afin de maximiser les chances d'aboutissement du projet ;

Chaque sous-projet est calibré afin de développer les mêmes compétences (réalisation électronique et programmation) et de présenter un niveau de complexité identique.

### 1.1 Description rapide et attributions relatives à chaque sous-projet

Comme illustré sur la figure 2, chaque sous-projet repose sur un contrôleur constitué d'une carte de programmation identique à base de microcontrôleur fournie lors de la première séance. En revanche, chaque binôme aura la charge de développer une carte électronique spécifique à sa partie et de programmer la carte micro-contrôleur pour atteindre l'objectif assigné. La carte à concevoir sera dotée :

- d'une nappe de connexion à la carte micro-contrôleur ;
- selon le projet, de capteurs, d'une interface utilisateur ou d'organes de commande rapprochée ;
- d'une interface de communication permettant d'échanger des informations avec les autres sous-projets en vue d'aboutir à l'objectif global (le suivi de ligne).

Chaque sous-projet peut être mené indépendamment des autres car l'assemblage des réalisations en un projet complet repose sur la définition de deux interfaces — Capteurs→IHM et IHM→Moteurs — normalisées électriquement (liaison série sur niveaux TTL) et logiciellement (protocole d'échange présentés en section 4.3.2 et 5.5.1).

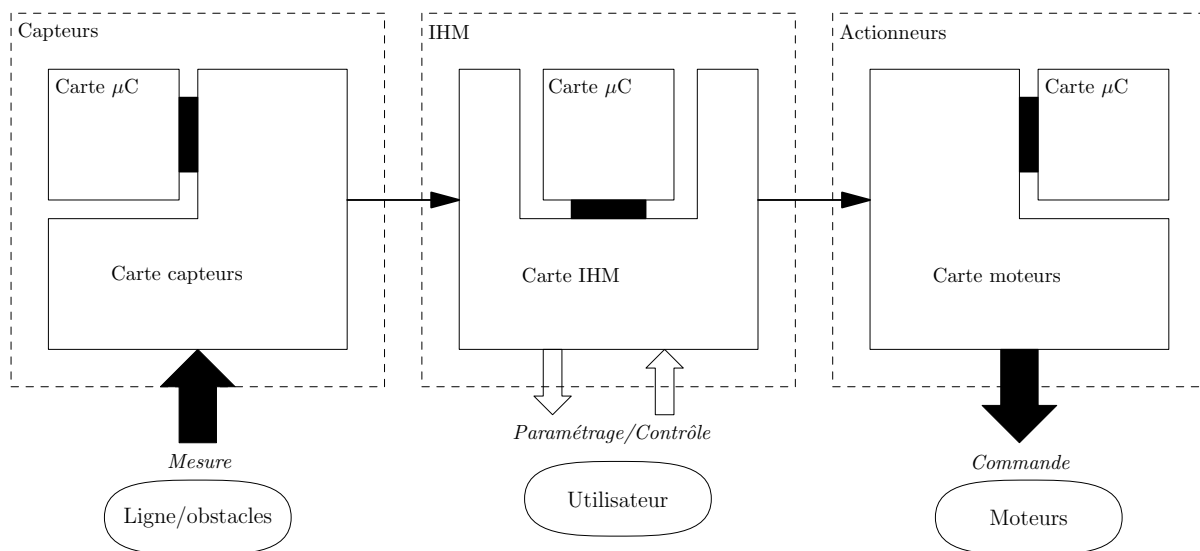


FIG. 2 : Découpage du projet en sous-projets

Les tâches propres à chaque sous-projet sont succinctement décrites ci-dessous :

#### Partie Capteurs

- Prototypage, test et calibrage des capteurs ;
- conception d'une carte électronique rassemblant les capteurs, leur électronique d'alimentation, l'interface de connexion à la carte micro-contrôleur, l'interface de communication avec la carte IHM ;
- implantation d'un algorithme élaborant la grandeur représentative de la distance entre le robot et un éventuel obstacle ;
- implantation d'un algorithme élaborant la grandeur représentative de la distance entre l'axe du robot et la ligne suivie ;
- encodage de ces deux valeurs selon le protocole imposé et transmission via une interface de communication filaire.

### Partie Interface (IHM)

- Prototypage et test de fonctionnement des éléments de l'interface homme-machine : écran LCD et boutons ;
- conception d'une carte électronique rassemblant l'IHM, l'interface de connexion à la carte micro-contrôleur ainsi que les interfaces de communication avec la carte capteurs et la carte de commande des moteurs ;
- décodage des valeurs transmises par la carte capteur via une interface de communication filaire et selon un protocole imposé ;
- gestion de l'interface graphique et des boutons ;
- implantation de l'algorithme d'asservissement de la trajectoire du robot ;
- mise en place de la détection d'obstacle ;
- encodage des consignes de vitesse et d'orientation selon le protocole imposé et transmission vers la carte de commande des moteurs via une interface de communication filaire.

### Partie actionneurs (Moteurs)

- Prototypage et test de fonctionnement des éléments de la commande rapprochée des moteurs ;
- câblage et acquisition des signaux issus des codeurs accouplés aux arbres moteurs ;
- conception d'une carte rassemblant l'électronique de commande des moteurs, les entrées d'information des codeurs, l'interface de connexion à la carte micro-contrôleur ainsi que l'interface de communication avec la carte IHM ;
- conversion des signaux issus des codeurs en une grandeur représentative des vitesses de rotation des moteurs ;
- décodage des consignes transmises par la carte IHM via une interface de communication filaire et selon un protocole imposé ;
- implantation de l'algorithme d'asservissement des vitesses de rotation des moteurs.

## 2 Structure d'un microcontrôleur

Un microcontrôleur est un composant associant dans un même boîtier un cœur de microprocesseur, de la mémoire et des périphériques.

**Microprocesseur** : le microprocesseur a pour rôle d'exécuter un programme composé d'une succession d'instructions rangées en mémoire. Pour cela, il dispose d'une *unité de décodage d'instructions*, d'une *unité arithmétique et logique* et de *registres*. Ces derniers sont destinés :

- à manipuler les données (registres de travail ou W) ;
- à indiquer l'état du processeur (registres SR) ;
- à pointer l'instruction à décodifier (registre PC) ;
- ou à piloter le moteur DSP intégré au microprocesseur.

Dans un langage évolué tel que le C, les registres ne sont presque jamais manipulés par le programmeur.

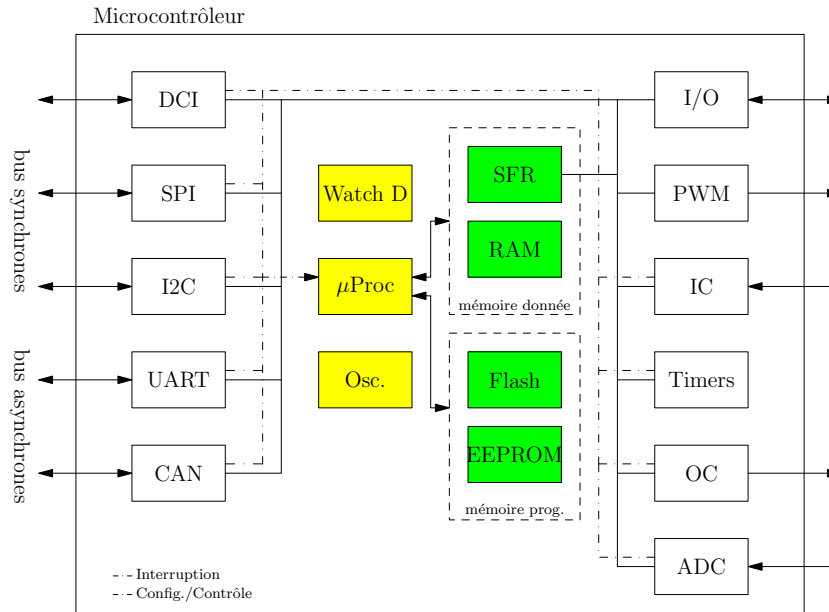


FIG. 3 : Structure interne d'un micro-contrôleur

**Mémoire** : la mémoire présente sur un microcontrôleur est utilisée différemment selon les architectures pour stocker programme et données. Dans le cas de la structure de type Harvard utilisée par Microchip, trois types de mémoires sont disponibles et utilisés comme suit :

- la RAM représente quelques ko de mémoire volatile : elle accueille les variables utilisées par le programme ;
- la FLASH représente quelques centaines de ko voire quelques Mo : elle accueille le programme ainsi que les données constantes. Cette mémoire est conservée en cas de perte de l'alimentation ;
- l'EEPROM est disponible pour stocker les données de configuration critiques d'un programme en cas de perte d'alimentation et les restituer au redémarrage. Généralement chaque microcontrôleur en embarque 1 ou 2 ko.

Les mémoires FLASH et l'EEPROM sont accédées par un bus de 24 bits permettant d'adresser potentiellement 16Mo appelés *espace programme*. l'espace programme est pointé par un registre qui dépile les instructions du programme appelé *compteur programme* ou PC. L'ensemble de cet espace d'adressage n'est pas uniquement occupé par du code programme ou par les données de l'EEPROM. Le début de la zone accueille la table des adresses des routines d'interruption et les 8Mo d'adresse haute sont réservés pour stocker quelques éléments tels l'identifiant du microcontrôleur ou ses bits de configuration. Finalement, la majorité de l'espace programme ne contient pas réellement de mémoire physique. Une grande partie des adresses sont inutilisées et pointent dans le vide.

Le microprocesseur accède à la mémoire RAM à l'aide d'un bus de données 16 bits qui ne permet d'adresser que 64ko de mémoire. Une partie de cet espace d'adressage (environ 2ko) est réservée pour accéder aux registres de périphériques appelés SFR (special function registers). Ces registres permettent de configurer et de piloter chacun des périphériques présents dans le microcontrôleur. Le découpage des espaces mémoire est présenté sur la figure 4 dans le cas spécifique du pic30F4012.

**Périphériques** : Les périphériques intégrés au microcontrôleur sont nombreux et aux fonctionnalités multiples :

- interfaces dédiées aux bus de communications synchrones (SPI, DCI, I2C) ;
- interfaces dédiées aux bus de communication asynchrones (UART, CAN) ;
- entrées-sorties TTL ou à collecteur ouvert ;
- timers ;

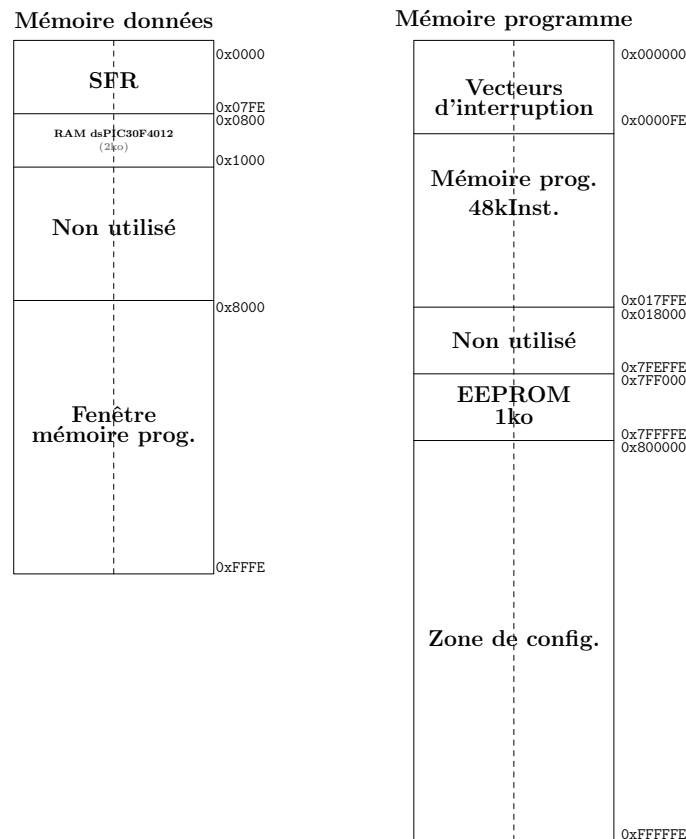


FIG. 4 : Espaces d'adressage données et programme

- input-capture ;
- output-compare ;
- générateur de modulation de largeurs d'impulsions (PWM).

Grâce au mécanisme d'interruption décrit en section 3.1, chacun de ces périphériques est apte à indiquer au microprocesseur l'occurrence d'évènements nécessitant un traitement immédiat.

### 3 Structure d'un programme sur microcontrôleur

Les programmes implantés sur les microcontrôleurs possèdent tous la même structure : les actions périodiques ou déclenchées par les périphériques sont insérées au sein de routines d'interruption. Parmi ces actions, on peut citer :

- l'acquisition de valeurs analogiques à intervalle de temps régulier ;
- la lecture d'une valeur reçue sur un bus de communication ;
- la détection d'un front sur une entrée à fréquence élevée ou lorsque l'instant du front doit être consigné avec précision.

Les actions dont la prise en charge n'est pas urgente ou dont le traitement est long sont insérées dans une boucle sans fin. Il s'agit généralement des actions d'interaction avec un opérateur humain parmi lesquelles on trouve :

- la détection d'un appui sur un bouton (méthode de scrutation aussi appelée pooling) ;
- la mise à jour d'un afficheur LCD.

### 3.1 Routines d'interruption

Une routine d'interruption est une fonction particulière qui ne prend pas de paramètre et qui ne retourne aucun résultat. *Elle n'est jamais explicitement appelée par le programme* mais son exécution est déclenchée sur événement d'un périphérique (timer, réception d'un octet sur un bus de communication, fin de transmission d'un octet, fin de conversion analogique, détection d'un front, overflow de certains registres ...)

**Fonctionnement :** Lorsqu'un périphérique souhaite signaler un événement, un drapeau est positionné dans l'un des registres IFSx. L'interruption n'est appelée que si le programme l'a autorisée. A cet effet, les registre nommés IECx contiennent un ensemble de bits dont chacun est lié à une interruption autorisée (valeur à 1) ou non prise en compte (valeur à 0). Lorsque l'interruption est autorisée, le passage à 1 du drapeau correspondant du registre IFSx déclenche immédiatement les actions suivantes :

1. les contenus du pointeur d'exécution et du registre d'état sont mémorisés ;
2. le registre d'exécution (PC) est chargé avec l'adresse de la routine d'interruption qui s'exécute immédiatement ;
3. après exécution de la dernière instruction de la routine, le pointeur d'exécution et l'état du processeur sont restitués automatiquement ;
4. l'exécution du programme principal reprend là où il avait été quitté.

Le code présent dans la routine est responsable de l'effacement du drapeau d'interruption (appelé acquittement) pour éviter une relance immédiate de l'interruption une fois la routine quittée ;

Pour le compilateur C30, chaque source d'interruption est liée à une routine grâce à son nom : ainsi l'interruption du timer 1 doit être nommée `_T1Interrupt()`, l'interruption indiquant la réception d'un octet sur l'UART se nomme `_U1RXInterrupt()`...

### 3.2 Structure type d'un programme

Le code d'un programme micro-contrôleur est usuellement organisé comme suit :

1. les routines d'interruptions sont définies ;
2. au sein de la fonction `main`, les périphériques sont initialisés, puis activés ;
3. les drapeaux d'interruptions (dans les registres IFSx) pour lesquels les routines ont été définies sont effacés par précaution afin d'éviter un départ non souhaité en interruption lié à un événement caduque ;
4. les sources d'interruptions pour lesquelles les routines ont été définies sont autorisées (*i.e. démasquées*) via les registres IECx ;
5. une boucle sans fin est ajoutée, vide si l'ensemble des actions est déclenchée sur interruption ou contenant les actions dont le traitement ne revet pas de caractère d'urgence.

Un tel code type est présenté ci-dessous et l'organigramme correspondant sur la figure 5. Comme le montre cette figure, une routine d'interruption doit toujours être traversant (*i.e. non bloquante*). De son coté, le programme principal se contente d'exécuter des tâches non urgentes (Actions 1 et 2) après une phase d'initialisation. La communication de données entre les routines d'interruption et le corps du programme (*main*) est assurée par des variables globales, c'est à dire déclarées en dehors de tout corps de fonction.

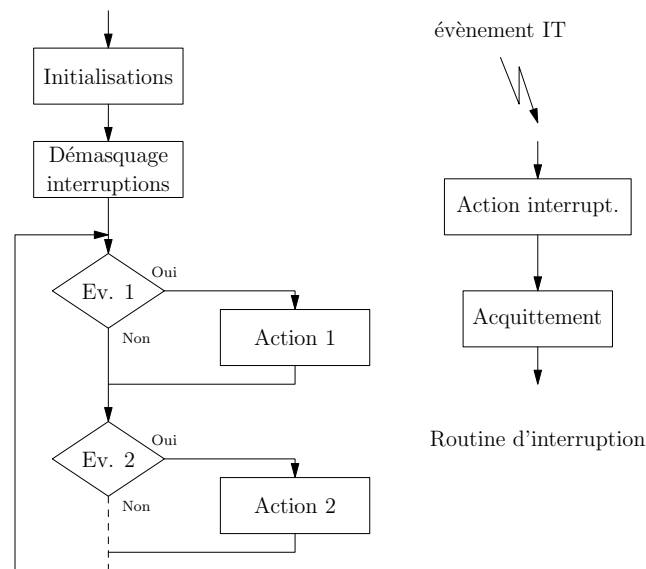
```

1  void __attribute__((__interrupt__,__no_auto_psv)) _[Periph1Name]Interrupt( void ){
2      ...
3
4      IFSxbits.[Periph1Name]IF=0;  // Acquittement de l'interruption
5  }
6
7  ...
8  void main(void){
9
```

```

10  init[Periph1Name]([Init args]); // Appel des fonctions
11  init[Periph2Name]([Init args]); // d'initialisation des périphériques
12  ...
13
14  IFSxbits.[Periph1Name]IF=0; // Acquittement préventif
15  enable[Periph1Name]Interrupt(); // Autorisation des interruptions
16  // déclenchées par Periph1
17
18  ...
19  while(1){
20      if([Event]){
21          [Low priority action]; // Pooling sur évènements peu prioritaires
22      }
23  }
24

```



Programme principal

FIG. 5 : Organigramme d'un programme type sur micro-contrôleur

### 3.3 Les ports d'entrées-sorties (I/O)

Les broches d'un microcontrôleur sont généralement associées à différentes fonctionnalités multiplexées : le nom de la broche indique alors ces fonctionnalités par priorité décroissante. Ainsi, une broche **AN7/RE8** est utilisée en entrée analogique dès lors que le module de conversion analogique est activé et que la programmation de la voie AN7 le prévoit. Selon cette nomenclature, le module I/O est généralement le moins prioritaire : il est donc important de vérifier qu'aucun périphérique ne masque ce module sur les broches devant être utilisées en I/O. Les entrées-sorties sont groupés en ports 16 bits (A0 à A15, B0 à B15, ...) et sont accessibles par des broches respectivement nommées RA0 à RA15, RB0 à RB15 ... Le nombre de broches dédiés aux I/O effectivement présentes sur un microcontrôleur dépend de sa taille et tous les bits d'un port ne sont pas nécessairement présents accessibles sous forme d'une broche. Par défaut, après reset du microcontrôleur, les broches configurées pour utiliser les ports d'entrées-sorties sont programmées en entrées pour éviter d'imposer des tensions sur des broches connectées à des circuits externes fixant leur propre niveau de tension. Sur le dsPIC30F4012, il n'est pas possible de fournir ou d'absorber un courant supérieur à **25mA** par broche avec une limite supérieure à **200mA** pour l'ensemble du microcontrôleur.

## Programmation des ports d'entrées-sorties

Chaque bit d'un port est paramétrable en entrée ou en sortie individuellement. Pour cela, on utilise le registre **TRISn** où **n** indique le port (A, B C, ...). Un bit à 1 place de bit en entrée alors qu'un bit à 0 le programme en sortie. Une fois la direction du port choisie, on lit l'état électrique ou on écrit la valeur TTL à sortir sur la broche à l'aide du registre **PORTn**. Enfin, il est possible que l'état électrique d'une broche soit différent de l'état logique fixé par le programme (court-circuit, pilotage d'un circuit capacitif en fréquence élevée). Cet état logique peut alors être relu à l'aide du port **LATn**. La circuiterie interne des entrées-sorties est illustrée sur la figure 6 .

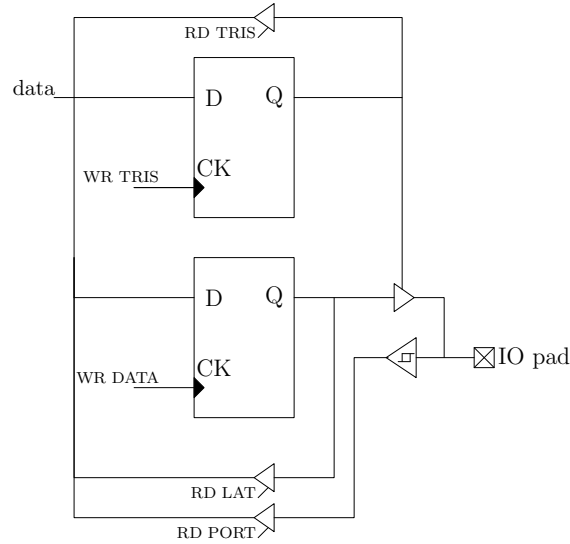


FIG. 6 : Circuiterie des entrées-sorties

## 3.4 Les timers

Les timers sont des modules utilisés pour produire des actions à intervalle de temps constant ou pour mesurer des durées entre deux événements successifs. Dans le premier cas, on parle du mode **output-compare** et dans le second de mode **input-capture**. Le dsPIC30F4012 dispose de 3 timers indépendants. Sur les microcontrôleurs Microchip, chacun de ces modes est mis en oeuvre en combinant un timer et un module spécifique au mode.

### 3.4.1 Utilisation du timer seul

Chaque timer est constitué d'un compteur de 16 bits permettant de balayer les valeurs de 0 à 65535. Le quantum temporel d'incrémement est donné par le temps de cycle  $T_{cy} = \frac{1}{F_{cy}}$  où la fréquence de cycle  $F_{cy}$  est donnée par :

$$F_{cy} = \frac{7.372 \times 10^6}{4} M_{PLL} Hz$$

$M_{PLL}$  est un facteur multiplicatif programmable qui vaut au maximum 16. Ainsi on obtient la fréquence maximale  $F_{cy} = 29.488 Mhz$  soit  $T_{cy} = 33ns$ .

A ce rythme, chaque compteur 16 bits atteint 65535 en 0,2ms, ce qui est un temps souvent trop court au regard des signaux à générer ou à mesurer. Pour allonger la durée de comptage, on peut exploiter deux pistes :

- cascader les timers 2 et 3 afin de produire un timer 32 bits et atteindre une durée de comptage pouvant atteindre de 145s ;
- ralentir l'horloge de comptage en divisant  $F_{cy}$  : c'est le rôle des prescalers.

Le fonctionnement du timer est le suivant (voir figure 7) :

1. le compteur 16 bits **TMRi** s'incrémement toutes les  $kT_{cy}$  où  $k$  représente la valeur de prescaler ;



2. lorsqu'il atteint la valeur cible définie dans le registre PRi :

- il est remis à 0 ;
- une demande d'interruption est levée ;

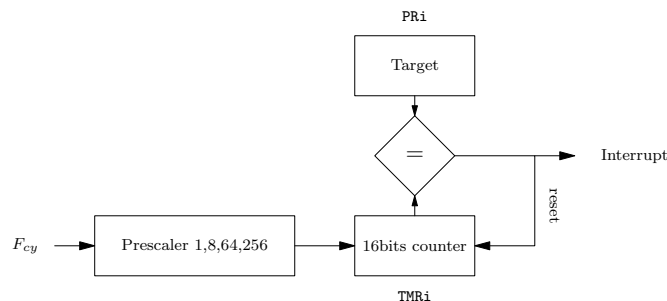


FIG. 7 : Utilisation d'un timer en générateur d'événements périodiques

En mode 32 bits le fonctionnement est identique, la valeur finale de comptage est inscrite dans la concaténation des registres PR3 (mot de poids fort) et PR2 (mot de poids faible). le registre de comptage TMR3 est incrémenté sur overflow du registre de comptage TMR2. Lorsque  $TMR3:TMR2 = PR3:PR2$ , les registres de comptage sont remis à 0 et une demande d'interruption est levée émanant du Timer 3.

### Programmation du Timer pour le déclenchement d'actions périodiques

Une librairie constituée des fichiers `periph_tmr.c` et `periph_tmr.h`, disponible sur votre environnement numérique de travail (Moodle du cours EEA0601) permet de simplifier le déclenchement d'actions à intervalles réguliers. Afin de répondre à la majorité des besoins, cette librairie exploite les timers 2 et 3 en mode 32 bits. Les fonctions disponibles sont :

- `InitTimer3(unsigned long periode)` permet d'initialiser le Timer3 en mode 32bits afin qu'il lève une demande d'interruption toute les  $3,39 \times \text{periode} \mu s$  avec  $\text{periode} \in \{0, \dots, 2^{32} - 1\}$ .
- `Timer3InterruptEnable(void)` autorise le déclenchement d'interruptions par le Timer3 et active le comptage dans TMR3.

☞ *Attention :*

La définition de la routine d'interruption `_T3Interrupt(void)` est indispensable préalablement à l'utilisation de la fonction `Timer3InterruptEnable(void)` sous peine de plantage.

#### 3.4.2 Mode output-compare

En association avec le module output-compare, un timer peut être utilisé pour activer une sortie TTL à chaque fois que le compteur atteint une valeur cible primaire inscrite dans le registre OCxR puis la désactiver lorsqu'il atteint une valeur cible secondaire inscrite dans le registre OCxSR. On peut ainsi générer des impulsions de durée variable et précise sans gestion logicielle ou même des PWM<sup>1</sup>. Ce mode n'étant généralement pas utilisé dans les projets d'EEA0601, il n'est pas décrit davantage ici.

#### 3.4.3 Mode input-capture

En association avec le module input-capture, un timer est utilisé pour mesurer précisément l'intervalle de temps entre deux événements sans scrutation logicielle. Le fonctionnement de ce mode est illustré sur la figure 8.

Sur l'occurrence d'un front sur la broche ICx, la valeur de comptage du Timer TMRy est instantanément mémorisé dans le registre ICxBUF. Le type de front détecté est paramétrable. Simultanément, l'exécution de la routine d'interruption `ICxInterrupt` est déclenchée (voir section 3.1) : cette routine a principalement pour objectif de traiter la valeur de comptage mémorisée du Timer y. Ainsi pour calculer la largeur

<sup>1</sup>cette dernière fonctionnalité est plus simple à implémenter au moyen du module PWM spécifique décrit en section 15 de la documentation de référence.

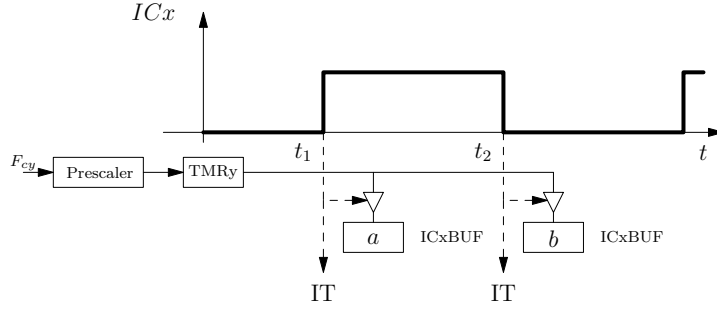
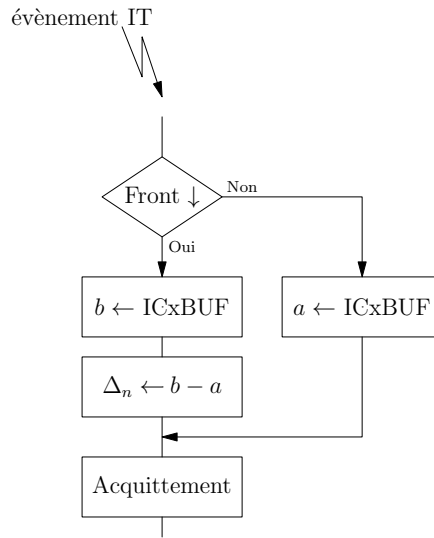


FIG. 8 : Fonctionnement du mode input-capture

$\Delta_t = t_2 - t_1$  de l'impulsion rectangulaire illustrée sur la figure 8, on utilisera l'organigramme présenté sur la figure 9.

FIG. 9 : Détermination d'un intervalle de temps entre deux fronts ( $\Delta_n$  est une image de  $\Delta_t$ )

### Programmation du mode input-capture

Le mode input-capture est facilement utilisable grâce aux bibliothèques `periph_ic.c` et `periph_ic.h` présentes sur votre environnement numérique de travail. Toutefois, il sera souvent nécessaire de compléter cette bibliothèque afin de l'adapter aux spécificités de votre application. En particulier la gestion du timer destiné à horodater l'instant d'occurrence du front est laissé à la charge du développeur.

- `ICxInterruptEnable(unsigned int edge)` active la détection de fronts de type `edge` sur l'entrée `ICx` et autorise le déclenchement d'une interruption lors de l'occurrence de cet événement. Sur un dsPic30F4012,  $x \in \{7, 8\}$ .

⚠ *Attention :*

- La définition de la routine d'interruption `_ICxInterrupt(void)` est indispensable préalablement à l'utilisation de la fonction `ICxInterruptEnable(unsigned int edge)` sous peine de plantage.
- Si vous êtes amené à modifier ou compléter la bibliothèque, il est important de renommer les fichiers associés afin de ne pas confondre votre code avec sa version originale.

### 3.4.4 Détermination de la valeur d'un prescaler

Comme expliqué précédemment, le prescaler a pour rôle de diviser la fréquence d'horloge  $F_{cy}$  et donc de ralentir la vitesse d'incrémentatation des timers. Sa valeur doit être fixée en fonction :

- de la durée  $\Delta_t$  maximale souhaitée entre deux déclenchements d'actions successives ;
- de l'estimation de l'intervalle de temps maximum observable  $\Delta_t$  entre deux fronts successifs sur ICx en mode input-capture.

Dans les deux cas, le prescaler est choisi tel que :

$$Pr > \frac{\Delta_t F_{cy}}{2^{16}}$$

☞ *Attention :*

Dès lors que cette condition est respectée, il est préjudiciable de surévaluer la valeur du prescaler sous peine de perdre en résolution temporelle (incréments par seconde) : en particulier, la mesure d'intervalle entre 2 fronts rapprochés sur ICx peut devenir imprécise.

## 3.5 Le module de génération d'une modulation de largeur d'impulsion (MLI)

Le microcontrôleur dsPIC30F4012 intègre un module permettant de générer jusqu'à 6 signaux PWM (Pulse Width Modulation) de période commune dont les sorties sont couplées deux à deux. Le fonctionnement de ce module est précisément décrit en section 15 du *dsPIC30F Family reference manual* [☞](#).

### 3.5.1 Fonctionnement du module PWM

Chaque broche PWMxH et PWMxL,  $x \in \{1, 2, 3\}$  peut au choix, être utilisée par le module PWM ou comme une I/O logique. La période de la modulation est commune aux PWM1, 2 et 3 et produite par un unique timer 16 bits programmable incrémenté au rythme  $\frac{F_{cy}}{N}$  où  $N \in \{1, 4, 16, 64\}$  désigne une valeur de prescaler programmable. Dans le mode *Edge aligned*, l'ensemble des sorties PWMxH utilisées passent à l'état haut lorsque le compteur du timer PTMR atteint sa valeur finale. En revanche, chaque sortie passe à l'état bas lorsque le timer atteint une valeur spécifiée dans le registre DCYx pour cette voie comme le montre la figure 10. Ce fonctionnement permet de générer un signal MLI de période et de rapport cyclique souhaités. Lorsque des voies PWMxL sont utilisées par le module PWM, elles prennent des valeurs complémentaires aux voies PWMxH associées à moins que le mode *independent output* n'ait été explicitement requis. Ce fonctionnement complétement facilite la commande des hacheurs en interdisant aux transistors haut et bas d'un même bras de pont d'être simultanément passant ou bloqué (avec l'adjonction d'un éventuel temps mort programmable).

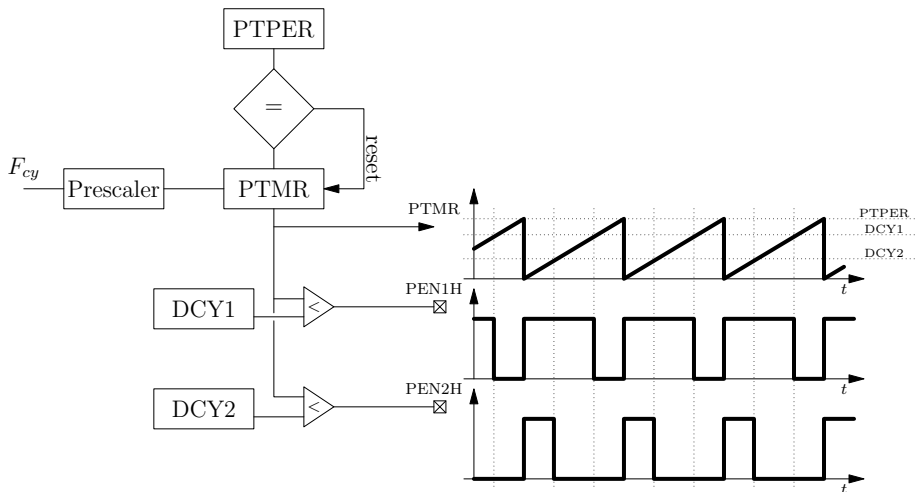


FIG. 10 : Fonctionnement du module PWM

### 3.5.2 Programmation du module PWM

Les fichiers `periph_pwm.c` et `periph_pwm.h` présents sur le cours Moodle offrent une librairie de fonctions de base permettant de programmer le module PWM. Les fonctions disponibles sont :

- `InitPWM(unsigned int periode, enum PWM_PIN_INIT_FLAGS PIN_Init)` : initialise le module PWM pour un signal de période  $\text{periode} \times 2.17\mu\text{s}$ . `PWM_PIN_INIT_FLAGS PIN_Init` fixe les broches à utiliser pour la PWM et doit être choisi selon une combinaison logique de `PEN1L`, `PEN2L`, `PEN3L`, `PEN1H`, `PEN2H` et `PEN3H`. Il est aussi possible de dissocier les broches L et H de la voie  $i$  (*independant mode*) en adjoignant l'élément `INDEPi`.  
Ainsi `InitPWM(460, PEN1L&PEN1H)` initialise une PWM de période 1ms dont les sorties sont en opposition sur `PEN1L` et `PEN1H`.
- `PWMEnable(enum MOD_ENABLE onoff)` : active ou éteint le module PWM.
- `PWM1PulseWidth(unsigned int Ton)` : fixe le rapport cyclique sur la voie 1 selon  $\alpha = \frac{\text{Ton}}{\text{periode}}$ .
- `PWM2PulseWidth(unsigned int Ton)` : fixe le rapport cyclique sur la voie 2.

### 3.6 Le module de conversion analogique-numérique

Le dsPIC30F4012 possède 9 entrées analogiques multiplexées vers quatre échantillonneurs bloqueurs étiquetés `CH0` à `CH3`. Les sorties de ces bloqueurs sont quantifiées séquentiellement sur 10 bits par un unique convertisseur analogique-numérique à approximations successives. L'étendue de mesure des données converties est paramétrable :

- soit entre `AVSS` et `AVDD` (images filtrées de la tension d'alimentation du microcontrôleur) i.e. entre 0 et 5V ;
- soit entre `Vref-` et `Vref+` qui doivent cependant rester tels que  $\text{AVSS} < \text{Vref-} < \text{Vref+} < \text{AVDD}$ .

Comme présenté sur la figure 11, le multiplexeur 1 permet de choisir les voies à numériser, en mode *single ended* (référéncées à la masse) ou *différentiel* (différence de potentiel entre deux `ANx`). Toutes les combinaisons d'entrées ne sont pas disponibles et on se référera à la documentation du microcontrôleur [☞](#) pour plus de détails. Lorsque les entrées à numériser sont connectées à différents bloqueurs, cela rend possible l'acquisition d'une image simultanée des tensions présentes sur ces entrées. L'unique convertisseur analogique-numérique est alors successivement connecté à chaque bloqueur pour afin de quantifier les tensions bloquées. Les valeurs converties sont ensuite stockées dans les registres `ADCBUFx`,  $x \in \{0, \dots, 15\}$ . Selon la programmation du module, il est possible de numériser en séquence une même voie ou plusieurs voies bloquées simultanément. De même les registres `ADCBUFx` peuvent être utilisés comme une FIFO ou au contraire en correspondance (stockage de la valeur associée à `ANi` dans `ADCBUFi`).

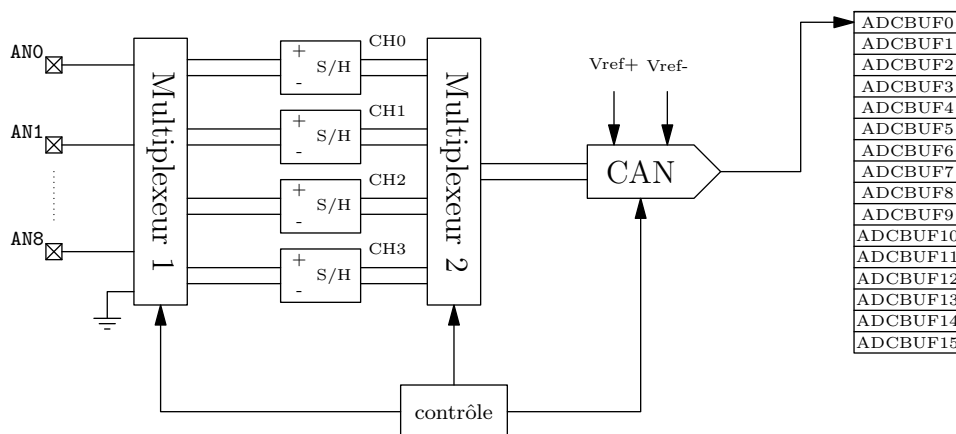


FIG. 11 : Fonctionnement du module de conversion analogique-numérique

### 3.6.1 Programmation du module de conversion analogique-numérique

La programmation du module de conversion analogique-numérique est grandement facilitée par la librairie constituée des fichiers `periph_adc.c` et `periph_adc.h` présents sur le cours Moodle de la matière. Les fonctions fournies sont les suivantes :

- `InitADC0(void)`, `InitADC0and1(void)`, `InitADC4and5(void)` et `InitADC0to3(void)` sont destinées à initialiser le module pour une acquisition simultanée sur les voies AN indiquées.
- `StartADC0(unsigned int* ADCxValue0)` bloque la valeur analogique présente sur AN0 puis lance la quantification de la voie bloquée et attend la fin de conversion. Le résultat est stocké dans la variable `ADCxValue0` qui doit être déclarée préalablement à l'appel de la fonction (passage par adresse).
- `StartADC0to3(unsigned int* ADCxValue0, unsigned int* ADCxValue1, unsigned int* ADCxValue2, unsigned int* ADCxValue3)` bloque **simultanément** les valeurs analogiques présentes sur AN $i$   $i \in \{0, 1, 2, 3\}$  puis lance successivement la conversion de chaque voie bloquée et attend la fin de l'ensemble des conversions. Les résultats sont stockés dans les variables `ADCxValue $i$` ,  $i \in \{0, 1, 2, 3\}$  qui doivent être déclarée préalablement à l'appel de la fonction (passage par adresse).
- `StartADC0seq0to4(unsigned int* ADCxValue0, unsigned int* ADCxValue1, unsigned int* ADCxValue2, unsigned int* ADCxValue3, unsigned int* ADCxValue4)` bloque et convertit **successivement** les entrées AN0 à AN4.

### 3.7 Le module de communication série asynchrone

Le module UART (*Universal Asynchronous Receiver Transmitter*) est un contrôleur de communication série présent dans de très nombreux équipements (microcontrôleurs, PC, automates, appareillage de mesure ...) et habituellement associé au standard RS-232. Une communication asynchrone série consiste en l'envoi successif de valeurs binaires transcodées en impulsions électriques sur une paire filaire ou un canal hertzien. Le caractère asynchrone de cette communication provient du fait que le signal d'horloge cadencant la communication n'est pas transmis en parallèle des données : cette horloge doit être reconstituée côté réception à l'aide d'informations *a priori* communes à l'émetteur et au récepteur (le débit par exemple) ou/et de l'observation des trames reçues. Ce format de communication nécessite donc de fixer certains paramètres identiquement à l'émission et à la réception :

- niveaux et modes électriques ;
- débit symbole (en Bauds) ;
- longueur des trames ;
- signalétique de début et de fin de trame ;
- contrôle d'erreur.

Le premier item est fixé par une norme d'interconnexion électrique telle le standard RS-232. Les autres paramètres sont réglés par programmation de l'UART.

#### 3.7.1 Fonctionnement du module UART

A l'émission, les données à transmettre sont écrites dans une pile FIFO à 4 emplacements. la donnée la plus ancienne dans la pile est mise en trame (ajout de la signalisation de start, de stop et de l'éventuel bit de parité) puis transférée dans le registre à décalage d'émission dès que ce dernier est vide. La trame est alors immédiatement transmise bit par bit sur TxD au rythme de l'horloge du *Baud Rate Generator* (BRG). Une interruption peut être déclanchée dès que la pile FIFO est vidée, ce qui permet à l'application de venir à nouveau la remplir.

A la réception du bit de start, le signal électrique acquis sur RxD est suréchantillonné à un rythme 16 fois plus rapide que l'horloge BRG pour pouvoir se synchroniser au mieux à l'émetteur et les données sont stockées dans le registre à décalage de réception. Lorsque la trame est complète et sans erreur, la donnée utile est transférée dans une FIFO à 4 emplacements et prête à être lue par le programme. Une interruption peut éventuellement être générée lors de l'occurrence de cet évènement.

Le fonctionnement de ce module est précisément décrit en section 19 du *dsPIC30F Family reference manual* [↗](#).

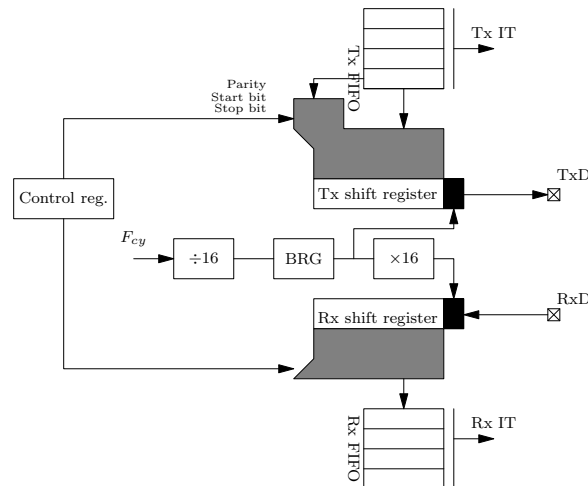


FIG. 12 : Fonctionnement du module UART

### 3.7.2 Programmation du module UART

Afin de faciliter la programmation du module, un ensemble de fonctions est mise à votre disposition dans les fichiers `periph_uart.c` et `periph_uart.h` présents sur le Moodle du cours. Ainsi :

- `serialInit(baudRate)` permet d'initialiser la liaison UART sur les broches alternatives (voir datasheet du dsPIC30f4012) sans contrôle de parité et avec un débit fixé à `baudRate` ;
- `serialCharReceived()` retourne une valeur non nulle lorsqu'un ou plusieurs octets sont présents dans la FIFO de réception ;
- `serialRead()` lit un octet dans la FIFO de réception ;
- `serialWrite()` écrit un octet dans la FIFO d'émission.

Cette librairie n'exploite pas le mécanisme d'interruption sur occurrence d'un évènement de réception : par conséquent le programme doit capter l'évènement de réception par polling.

## 3.8 Le module de communication série synchrone

Le module de communication série synchrone (SPI ou Serial Peripheral Interface), transmet ou/et reçoit des données au format 8 ou 16bits via un bus comportant quatre signaux :

- SDI qui porte le signal des données série entrant dans le module ;
- SDO qui porte le signal des données série sortant du module ;
- $\overline{SS}$  qui selectionne le module sur le bus ;
- CLK qui cadence le transfert des données.

Un bus SPI dont la connexion est présentée sur la figure 13, peut accueillir un maître et autant d'esclaves que souhaité. Le maître a la responsabilité de générer l'horloge CLK et de sélectionner le ou les esclaves émetteurs ou destinataires des données du bus. Cette sélection est opérée en plaçant à 0V la ligne  $\overline{SS}$  de chaque esclave à sélectionner durant le temps de la transmission. Cette dernière s'opère bit par bit, poids fort en premier au rythme de l'horloge CLK qui peut atteindre plusieurs MHz et doit rester inférieur à celui du module le plus lent.

### 3.8.1 Fonctionnement du module de communication série synchrone

Le module SPI dont le fonctionnement est précisé en détail en section 20 du *dsPIC30F Family reference manual* [\[7\]](#), comporte principalement un registre à décalage `SPIxSR` configurable en 8 ou 16 bits et pour lequel chaque coup d'horloge CLK provoque le décalage à droite d'un bit de son contenu. Le bit de poids fort sortant du registre est transmis sur SDO alors que le bit de poids faible entrant provient de l'échantillonnage de SDI (voir figure 14).

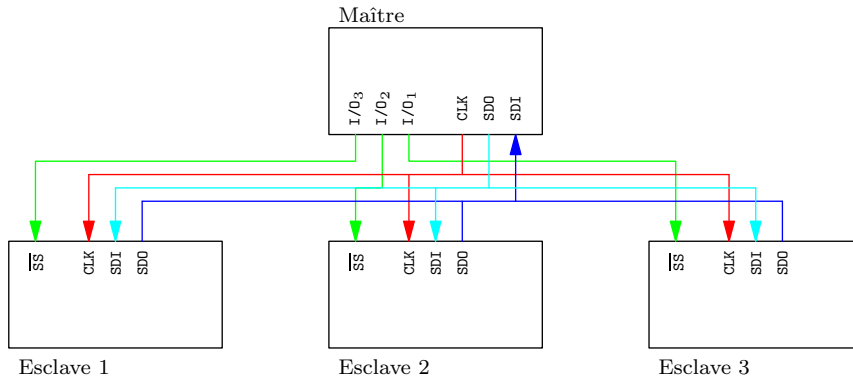


FIG. 13 : Connexion du bus SPI

Lorsque le dsPIC est maître et le module est activé, l'écriture dans le registre **SPIxBUF** stocke d'abord la donnée à transmettre dans un registre de latch (**SPIxTXB** non directement accessible) puis cette donnée initialise le contenu de **SPIxSR** dès la fin de transmission précédente puis provoque la génération de 8 ou 16 périodes d'horloge (selon le mode 8/16 bits) et donc l'envoi du mot contenu dans le registre sur **SDO**, bit par bit. Simultanément, le niveau TTL présent sur **SDI** est acquis et stocké.

Contrairement au bus série asynchrone, la transmission de l'horloge par le maître sur le bus évite au récepteur d'avoir à connaître *a priori* la fréquence de l'horloge. En revanche, la polarité initiale et le type de front d'horloge provoquant l'émission d'un bit doivent être identiques pour l'ensemble des modules présents sur le bus.

A l'issue de la transmission, une interruption peut être déclenchée pour indiquer cet évènement. Les 8 ou 16 bits reçus sur **SDI** sont accessibles par la lecture de **SPIxBUF** qui retourne les données latched dans un registre de réception (**SPIxRXB** non directement accessible). Il est à noter que les données reçues sur l'entrée **SDI** restent disponibles à la lecture y compris durant la réception de la transmission suivante, ce qui permet d'accélérer les échanges. Evidemment l'achèvement d'une transmission écrase immédiatement la réception précédente.

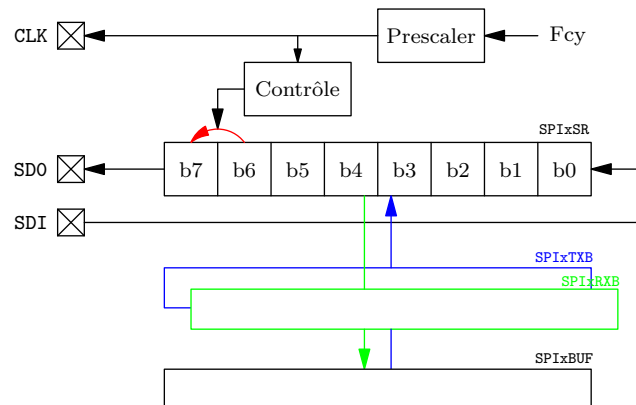


FIG. 14 : Fonctionnement du module SPI en mode maître

### 3.8.2 Programmation du module de communication série synchrone

Les projets développés en EEA0601 ne nécessitent pas de programmer directement le bus SPI même si ce dernier est utilisé pour le pilotage de l'afficheur LCD mis en œuvre en section 6. Par ailleurs, l'absence de protocole standardisé reposant sur le bus SPI rend illusoire la proposition de fonctions génériques réutilisables au sein de projets mettant en œuvre un panel diversifié d'éléments communicant en SPI. Enfin l'utilisation du SPI est souvent complétée d'autres signaux (**Enable** ou **R/ $\bar{W}$**  par exemple) qui nécessitent de respecter des chronogrammes précis. C'est pourquoi il est recommandé de travailler directement avec les registres du dsPIC associés à l'exploitation de ce bus.

- SPIxCON permet de régler la vitesse de modulation du bus, les modes (8 ou 16 bits, Maître ou esclave) ainsi que le calage de l'horloge relativement aux données à émettre ;
- SPIxSTAT permet d'activer le module SPI et informe de l'état des registres d'émission et de réception ;
- SPIxBUF contient en lecture les données reçues à l'issue d'un transfert et accepte en écriture les données à transmettre.

Un exemple d'utilisation de ces registres est disponible au sein de la librairie permettant l'initialisation et l'écriture sur l'afficheur DOGM163 (fichiers LCD.c et LCD.h disponibles sur Moodle).

## 4 Acquisition et traitement des mesures issues des capteurs

Il s'agit ici de créer une carte permettant au robot de se repérer dans l'espace et embarquant

- un capteur ultra-sons destiné à détecter un obstacle ;
- des capteurs infra-rouge exploités pour effectuer un suivi de ligne ;

ainsi que de programmer le microcontrôleur afin de restituer ces informations toutes les 100ms via une liaison série de type RS232.

### 4.1 Acquisition et traitement des données issues du capteur ultra-sons (séance 3)

L'acquisition de l'information de distance à un obstacle repose sur l'émetteur-récepteur d'ultra-sons SRF05. Celui-ci s'interconnecte au micro-contrôleur grâce aux signaux *trigger* et *echo* dont le chronogramme est présenté sur la figure 15.

- ✓ **Q.4.1-1** Etudiez la datasheet de ce système, et expliquez en quelques phrases son principe et son fonctionnement.
- ✓ **Q.4.1-2** Câblez le capteur sur plaque Labdec. On utilisera pour cela un générateur d'impulsions pour générer le signal *trigger* et un oscilloscope pour observer les signaux *trigger* et *echo*.
- ✓ **Q.4.1-3** Quelle est la durée minimale de l'impulsion du signal *trigger*,  $T_{pulse}^{min}$ . Donnez la relation entre la distance de l'obstacle et la largeur d'impulsion du signal *echo*  $\Delta T_{IC}$ .
- ✓ **Q.4.1-4** Quelle période des impulsions *trigger*,  $T_{PWM}$  vous semble adaptée au problème ? Justifiez.

séance

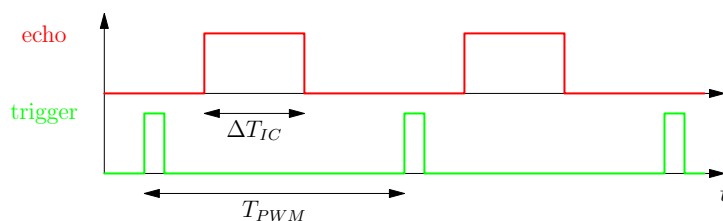


FIG. 15 : Chronogramme des signaux de pilotage du SRF05

- ✓ **Q.4.1-5** Vérifiez la conformité du fonctionnement du capteur ultrasons à celui indiqué dans la datasheet en traçant la courbe d'étalonnage *distance de détection (d)* vs *durée de l'état haut de l'écho ( $\Delta T_{IC}$ )*. Pour cela, on utilisera un obstacle suffisamment grand, plan et rigide pour permettre une réflexion franche des ondes (une plaque de carton par exemple).

#### 4.1.1 Génération périodique du trigger

<sup>2</sup> A présent, on souhaite utiliser la fonctionnalité PWM du microcontrôleur dont le fonctionnement est décrit au chapitre 15 du *dsPIC30F Family Reference Manual* [↗](#) afin de générer l'impulsion *trigger* à intervalle régulier.

<sup>2</sup>Cette partie peut être réalisée à compter de la séance 8 lors de la phase de programmation.



- ✓ **Q.4.1-6** Expliquez en quelques phrases le fonctionnement de ce module puis l'utilité des principaux registres mis en jeux.

On choisit de générer une impulsion trigger de largeur  $1ms$  et de période  $100ms$ .

- ✓ **Q.4.1-7** Sachant que par défaut le registre 16 bits de comptage PTMR s'incrémente  $F_{cy} = 29488000$  fois par seconde, quel prescaler (bits PTCKPS du registre PTCN) vous semble le plus adapté à ces durées ? Quelle doit alors être la valeur du registre PTPER ?
- ✓ **Q.4.1-8** Proposez une fonction `void startTrigger(int Periode)` qui enclenche la génération d'impulsions trigger sur la pin RE01 de largeur souhaitée et de période `Periode`.

☞ *Indications :*

Cette fonction devra :

- Réinitialiser le compteur à 0 (registre PTMR) ;
  - initialiser la valeur limite de comptage ;
  - sélectionner la sortie PEN1L ;
  - régler le registre de rapport cyclique DCY pour obtenir la largeur d'impulsion souhaitée ;
  - positionner le prescaler et activer la génération du signal.
- ✓ **Q.4.1-9** Appelez cette fonction dans votre fonction `main` en la faisant suivre d'un `while(1)` ; pour mettre le programme en bouclage infini une fois la PWM activée. Visualisez la sortie PEN1L sur oscilloscope. On enregistrera une copie d'écran où l'on fera apparaître les périodes et largeur d'impulsion observées puis on comparera cette période et largeur d'impulsion aux valeurs attendues.

#### 4.1.2 Acquisition de la largeur d'impulsion d'écho

L'acquisition de la largeur d'impulsion de l'écho nécessite de détecter les instants d'un front montant  $T_r$  et du front descendant suivant  $T_f$ , puis de calculer  $\Delta T_{IC} = T_f - T_r$ .

Pour cela, il est simple d'utiliser la fonctionnalité d'Input-Capture du microcontrôleur décrite en section 3.4 et de manière plus précise dans la section 13 du *dsPIC30F Family Reference Manual* [☞](#).

- ✓ **Q.4.1-10** Expliquez en quelques phrases le principe de l'input-capture ainsi que le rôle des registres de configuration associés (ICxCON et ICxBUF).
- Le dsPIC30F4012 dispose de 2 entrées d'input-capture (IC7 et IC8) et chacune de ces entrées est susceptible de déclencher une interruption sur occurrence d'un front. Dans la suite, on utilise IC7 que l'on souhaite coupler au timer 3.
- ✓ **Q.4.1-11** Compte-tenu de la période du signal *echo* déterminée dans la question Q.4.1-4, quelle valeur de prescaler du timer 3 semble la plus adaptée ?
- ✓ **Q.4.1-12** En appelant  $N_f$  et  $N_r$  les valeurs respectives du compteur TMR3 aux instants respectifs de front descendant et montant, quelle est alors l'équation permettant de relier la valeur à renvoyer sur la liaison UART à la valeur  $\Delta N_{IC} = N_f - N_r$  mesurée .
- ✓ **Q.4.1-13** Vous êtes vous demandé ce qui se passe lorsque le compteur franchi sa limite de comptage (0xFFFF) entre les instants de front montant et descendant ? Pourquoi cela ne pose-t'il pas de problème lorsque les variables utilisées pour le calcul d'intervalle sont non signées ?
- ✓ **Q.4.1-14** Proposez une fonction `void startIcMonitoring(void)` qui :
- initialise IC7CON pour utiliser le Timer 3 et détecter tout type de fronts ;
  - initialise les registres PR3 et TMR3 du Timer 3 aux valeurs respectives 0xFFFF et 0 ;
  - s'assure que les interruptions du Timer 3 sont masquées (registre IEC0) puis active le timer ;
  - efface le drapeau d'interruption IC7 (registre IFS1) puis démasque l'interruption (registre IEC1).

A présent, il faut implémenter la routine d'interruption qui doit se déclencher lors de la détection des fronts.

✓ **Q.4.1-15** Proposez un organigramme décrivant les actions prises en charge par la routine.

✓ **Q.4.1-16** Programmez la routine d'interruption dont la syntaxe est la suivante :

```

1 void __attribute__((__interrupt__,__no_auto_psv)) _IC7Interrupt( void )
2 {
3     ...
4
5     IFS1bits.IC7IF=0; //raz flag
6 }
```

☞ *Indication :*

En C, le mot clef `static` permet de conserver la valeur d'une variable locale à une fonction entre deux appels de celle-ci.

✓ **Q.4.1-17** Interfacez à présent le microcontrôleur en lieu et place du générateur d'impulsions et de l'oscilloscope puis testez l'ensemble PWM/Input-Capture pour différentes distances de détection.

☞ *Astuce :*

On pourra utiliser le mode pas à pas et la visualisation de variables (Window|Debugging|Variables) pour observer la variable contenant distance à l'obstacle. La valeur de la variable n'est lisible que lorsque le programme est stoppé dans une zone programme située dans son scope.

☞ *Attention :*

Pour plus de sécurité, il vaut mieux lancer la surveillance des fronts avant l'activation de la PWM. Voyez-vous pourquoi ?

## 4.2 Programmation de l'acquisition et du traitement des données issues des capteurs IR

### 4.2.1 Partie expérimentale (séance 4)

✓ **Q.4.2-1** Parcourez la documentation du TCRT5000 identifiant les valeurs électriques et mécaniques importantes.

✓ **Q.4.2-2** Proposez un schéma de câblage en précisant les valeurs des composants.

Il s'agit à présent de réaliser le montage électronique et mécanique d'un capteur pour pouvoir effectuer des relevés expérimentaux.

✓ **Q.4.2-3** Réalisez une platine (en époxy de CI) dotée de 4 vis périphériques de réglage de hauteur et d'un capteur TCRT5000 fixé en son centre comme présenté sur la figure 16.

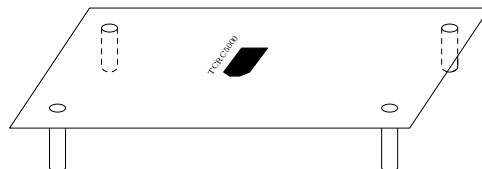


FIG. 16 : Dispositif de mesure

✓ **Q.4.2-4** En supposant qu'une source de tension de 5V est utilisée, déterminez les valeurs des résistances à placer en série avec la diode d'émission et sur le collecteur du phototransistor pour obtenir un fonctionnement correct. Câblez le TCRT5000 avec ces composants en identifiant le point de mesure à exploiter.

- ✓ **Q.4.2-5** En utilisant votre dispositif ainsi que Matlab ou Excel, effectuez un relevé pour différentes hauteurs de la courbe de tension du phototransistor en fonction de la position latérale à la ligne.

☞ *Attention :*

Validez la robustesse de vos mesures en effectuant si besoin plusieurs relevés identiques et en modifiant les conditions d'éclairage. Conclusion.

- ✓ **Q.4.2-6** Quelle sont d'après vous les valeurs optimales d'écartement et de distance des capteurs par rapport au sol ? Pourquoi ?

- ✓ **Q.4.2-7** Déduisez-en le nombre  $N$  de capteurs adéquat pour le projet.

#### 4.2.2 Réalisation de la carte sous Eagle® (séances 5 à 7)



Il est nécessaire pour poursuivre de disposer de la carte capteurs.

- ✓ **Q.4.2-8** Recensez les fonctionnalités mobilisées par la carte ainsi que les pins du microcontrôleur associées.

☞ *Indication :*

Pour éviter de consommer des I/O nécessaires à l'acquisition analogique ou à la détection d'obstacle, on utilisera en priorité les pins du port E (sauf RE0) pour commander l'allumage des LEDs IR. On prendra également soin de prévoir un montage à transistor permettant de limiter le courant consommé sur chacune de ces sorties.

- ✓ **Q.4.2-9** En utilisant :

- la *librairie de composants* ,
- les consignes de réalisation prodiguées par vos enseignants,
- le *mode d'emploi d'Eagle®* ,

disponibles sur le cours Moodle, réalisez le schéma de principe et le routage de la carte. Le typon sera réalisé en deux couches avec plan de masse sur la couche supérieure. On prendra soin de positionner les connecteurs en périphérie de la carte.

☞ *Informations :*

- N'utilisez jamais l'auto-routage ;
- Placez précisément vos capteurs IR (ils doivent être alignés et régulièrement espacés) ;
- Soyez rigoureux : composants positionnés du bon côté de la carte, pistes dimensionnées en fonction du courant qui y circule, présence de condensateurs de filtrage d'alimentation, angles des pistes à  $45^\circ$ , limitation du nombre de vias ...
- Après validation par l'enseignant, vous transmettez vos fichiers à `david.carton@univ-reims.fr` et/ou `maxime.colas@univ-reims.fr` pour tirage de la carte.

#### 4.2.3 Acquisition de la mesure sur microcontrôleur (séance 9)

Suite aux questions précédentes, vous disposez :

- du nombre  $N$  de capteurs nécessaires TCRC5000 au projet ;
- de l'entraxe  $D$  des capteurs permettant une estimation optimale de la distance à la ligne.
- d'une courbe tension versus distance à la ligne (centre du ruban pris pour origine) pour chaque capteur.

Le microcontrôleur est doté de convertisseurs analogique-numérique 10 bits multiplexés.

- ✓ **Q.4.2-10** Quelles devraient être les valeurs numériques min et max observées lors de l'acquisition sur un capteur ?
- ✓ **Q.4.2-11** Repérez sur le connecteur 26 points de la carte microcontrôleur la connection à l'entrée AN0 du convertisseur analogique-numérique et raccordez-y la sortie collecteur du phototransistor.
- ✓ **Q.4.2-12** Ecrire un programme qui acquiert en boucle les données d'un capteur en utilisant les fonctions `initADC0()` pour initialiser le convertisseur et `startADC0(&IRValue)` pour récupérer la valeur convertie dans la variable `IRValue`.

☞ *Astuce :*

On pourra utiliser le mode pas à pas et la visualisation de variables (Window|Debugging|Variables) pour observer les valeurs acquises sur le convertisseur et ainsi valider les résultats de la question Q.4.2-10. La valeur des variables n'est lisible que lorsque le programme est stoppé dans une zone programme située dans son scope.

Vérifiez les valeurs min et max obtenues expérimentalement en faisant varier latéralement et verticalement les distances au centre d'une ligne noire.

Il s'agit à présent de récupérer les mesures sur l'ensemble des capteurs. Pour cela, on propose d'utiliser une interruption périodique de période  $T = \frac{1}{100Nf}s$  durant laquelle on acquerra la mesure d'un seul capteur à la fois. Pour éviter les interférences entre capteurs, on propose de n'allumer que la led correspondant au phototransistor mesuré. Lorsque l'ensemble des capteurs aura été passé en revue, c'est à dire toute les 10ms, on utilisera les acquisitions pour estimer la position du ruban puis les transmettre.

- ✓ **Q.4.2-13** Donnez l'organigramme décrivant le traitement à effectuer dans cette interruption.
- ✓ **Q.4.2-14** Sachant que la fréquence d'incrémentement des timers 16bits est fixée par défaut à  $F_{cy} = 29,488\text{MHz}$ , quel est le plus faible prescaler permettant d'assurer une période supérieure ou égale à  $T$  ?
- ✓ **Q.4.2-15** Initialisez le timer1 pour obtenir une interruption selon cette période. Pour cela, on définira une fonction `startTimer1(int periode)` qui :
  - initialise la valeur maximale de comptage (registre PR1),
  - le compteur (registre TMR1),
  - le prescaler,
  - réinitialise le drapeau d'interruption (registre IFS0) et demasque l'interruption (registre IEC0) du timer 1.
  - active le timer (registre T1CON).

☞ *Indication :*

Utilisez les informations fournies dans la section 12 du *dsPIC30F Family Reference Manual* [↗](#).

- ✓ **Q.4.2-16** Sur la base de l'organigramme proposé dans la question Q.4.2-13, écrivez la routine d'interruption du Timer1 permettant d'acquérir la mesure infrarouge d'un seul capteur à la fois et collecte les acquisitions dans un vecteur `measuresIR[nSensors]`. On rappelle la syntaxe de déclaration de la routine d'interruption du timer 1 :

```
1 void __attribute__((__interrupt__,no_auto_psv)) _T1Interrupt( void )
2 {
3     ...
4
5     IFS0bits.T1IF=0; //raz flag
6 }
```

#### 4.2.4 Conversion des mesures en position (séances 9 et 10)

##### Quantification simple

La conversion des mesures en position est une opération qui peut être menée de façon plus ou moins complexe selon la précision de mesure souhaitée. Dans le cas le plus simple, il s'agit de définir pour chaque capteur un seuil  $S_p$  sur la mesure au delà duquel la ligne est supposée présente. Si l'espace entre capteurs successifs  $E$  et le seuil  $S_p$  sont correctement choisis, il est possible d'observer une présence sur deux capteurs consécutifs lorsque la ligne se situe entre ces capteurs. La figure 17 présente pour un dispositif à 5 capteurs, les 10 observations distinctes valides avec cette méthode en fonction de la position de la ligne au sol. Pour chacune de ces observations, un secteur est alors associé qui à son tour sera converti en position.

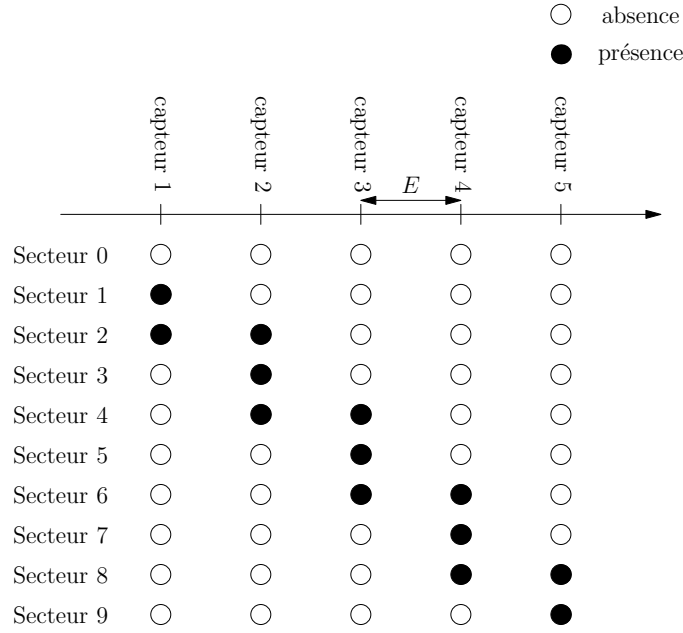


FIG. 17 : Correspondance entre secteurs et observations dans le cas d'une quantification simple de la position par rapport à la ligne

- ✓ **Q.4.2-17** Proposez une méthode simple permettant de convertir ces observations en secteur puis en mesure de distance du capteur central par rapport à une ligne au sol en mm.
- ✓ **Q.4.2-18** Produisez le code C associé à votre méthode et insérez-le dans une fonction `char sensorsToDist(unsigned int sensors[], int Nsensors, unsigned int e, unsigned int sp)` où `sensors[]` représente le tableau de taille `Nsensors` contenant les mesures des capteurs, `e` est l'entraxe des capteurs, et `sp` est le seuil de binarisation des valeurs infra-rouge converties.

##### Quantification fine (étape facultative)

Compte tenu de l'allure de la réponse des capteurs infra-rouge observée dans la question Q.4.2-5, il est possible d'obtenir une estimation plus précise de la position que celle issue d'une quantification simple.

- ✓ **Q.4.2-19** En modélisant les courbes obtenues dans la question Q.4.2-5 comme présenté sur la figure 18, expliquez en quoi la conversion proposée par l'organigramme de la figure 19 améliore l'estimation tout en restant simple à implémenter. Dans cet organigramme, on retiendra les définitions suivantes :

$$\begin{cases} f_1(c) &= -2E + (c[1] - 2^{Nb}) \frac{E}{2^{Nb}} \\ f_9(c) &= 2E + (2^{Nb} - c[5]) \frac{E}{2^{Nb}} \\ f_P(c) &= \frac{(Sec-5)E}{2} + (c[\frac{Sec}{2} + 1] - c[\frac{Sec}{2}]) \frac{E}{2^{Nb+1}}; \\ f_I(c) &= 5(Sec-5)E + (c[\frac{Sec+3}{2}] - c[\frac{Sec-1}{2}]) \frac{P}{2^{Nb}}; \end{cases}$$

☞ *Indication :*

On pourra notamment comparer (dans Matlab par exemple) les erreurs maximales des 2 méthodes ainsi que l'erreur quadratique moyenne de chacune.

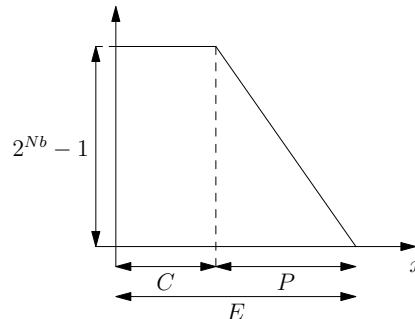


FIG. 18 : Modélisation de la réponse des capteurs IR en fonction de leur distance  $x$  au centre d'une ligne au sol de largeur fixée.  $N_b$  correspond au nombre de bits du convertisseur analogique numérique utilisé.

### 4.3 Transmission des mesures et tests (séance 11)

Les informations de position et de distance doivent être transmises à la carte IHM, développée par un autre binôme. La mise en place de cette communication nécessite d'utiliser un standard de signalisation électrique et un protocole d'échange d'informations compréhensibles par l'émetteur et le récepteur.

#### 4.3.1 Standard d'interconnexion

Le standard retenu met en œuvre le module UART du microcontrôleur (voir section 3.7) pour piloter une liaison série asynchrone sur niveaux TTL, paramétrée comme suit :

- 9600 Bauds ;
- 1 bit de stop ;
- pas de contrôle de parité.

#### 4.3.2 Format des remontées d'information de position et de distance

Les mesures sont transmises sur ce bus sous la forme d'un octet dont le bit de poids fort indique la nature de l'information :

- 0 : position par rapport à la ligne
- 1 : ultra-son

Les 7 bits de poids faibles doivent être interprétés comme une valeur  $V$  :

- **Mesures infra-rouge :**  $V$  représente la distance algébrique (donc signée) à la ligne exprimée en mm. Lorsque la ligne est perdue ou que la mesure est incohérente, la valeur -64 est retournée.
- **mesure ultra-son :**  $V$  représente la détection d'un objet situé à  $\frac{V}{2}$  cm. au delà de 63cm, la valeur 127 est retournée.

- ✓ **Q.4.3-1** Créez une fonction destinée à mettre les mesures infra-rouge et ultra-son au format requis par le protocole.
- ✓ **Q.4.3-2** En considérant un rythme de modulation de 9600 bauds sur la liaison asynchrone, quel est le nombre maximal de transmissions par seconde de chaque information (infra-rouge et ultra-son) possible.

Compte tenu de la dynamique attendue du robot, il peut être considéré qu'un rythme de transmission de chacune des mesures toute les 100ms est suffisant.

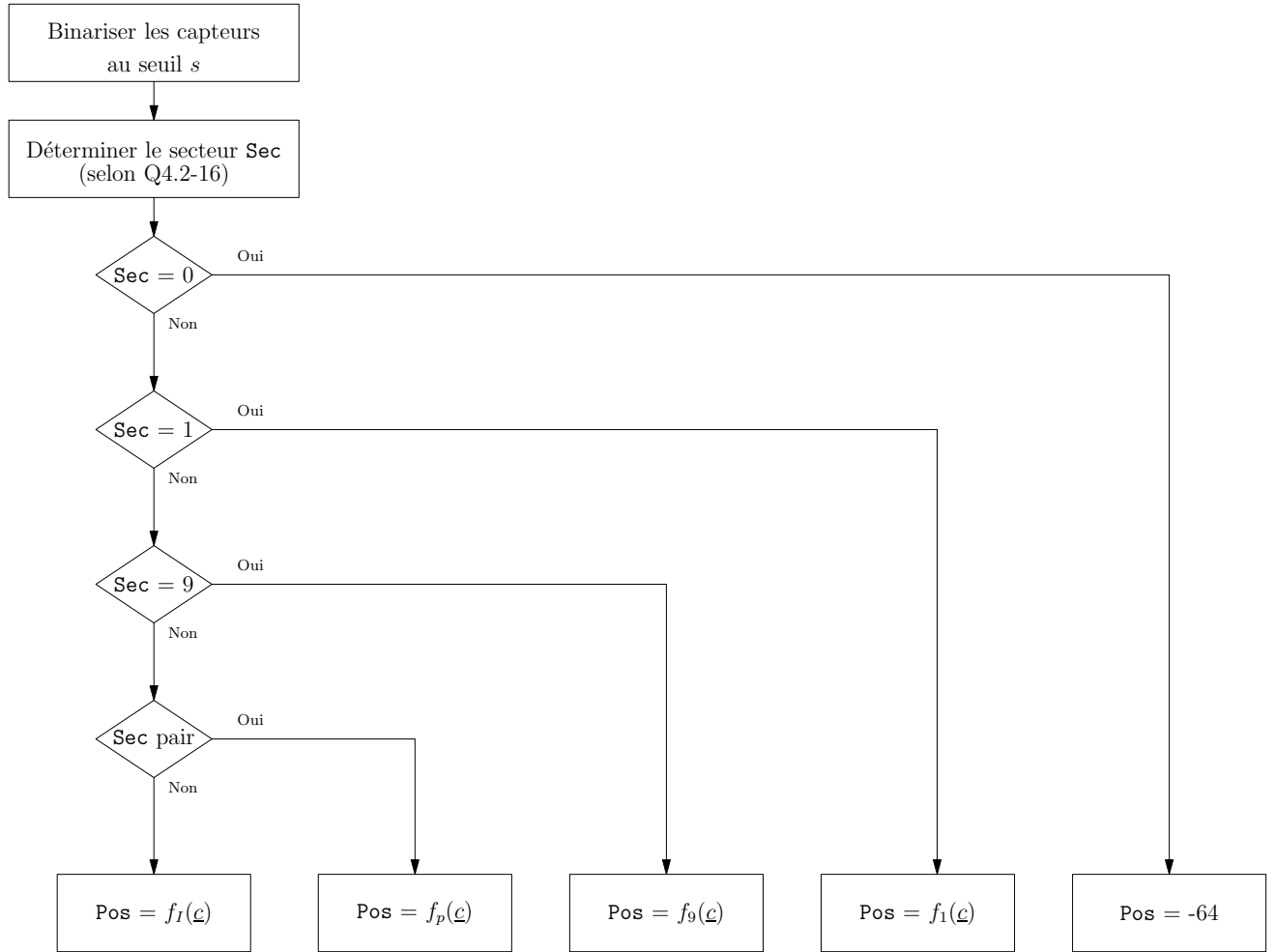


FIG. 19 : Estimation fine de la distance latérale du centre du robot par rapport à une ligne au sol de largeur fixée (cas  $N = 5$ )

## 5 Commande des moteurs du Traxter II

Le robot Traxter II est équipé de deux moteurs à courant continu monté en tête bêche et munis de motoréducteurs. Ils possédant les caractéristiques suivantes :

- Tension nominale : 7.2V
- Courant maximal d'induit : 4A
- Vitesse maximale en sortie de motoréducteur : 160 T/mn
- Couple de démarrage : 7.2kg.cm

Un codeur rotatif, fixé sur l'axe de chaque moteur permet de récupérer une image de sa vitesse de rotation. Ces codeurs sont alimentés en 5V et délivrent 624 impulsions par tour.

### 5.1 Rappels : variation de la vitesse d'un moteur à courant continu

Un moteur à courant continu est constitué d'un inducteur (généralement le stator) destiné à créer un champ magnétique  $B_s$  au travers d'un induit (généralement le rotor). Lorsque l'induit est le siège d'un courant  $I_i$ , le couple généré par la force de Laplace le met en rotation. Ce couple est donc proportionnel au courant  $I_i$  qui traverse le conducteur de l'induit :

$$C = kI_i$$

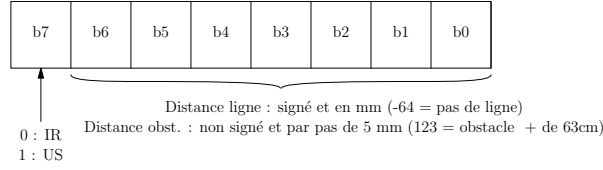


FIG. 20 : Protocole de communication

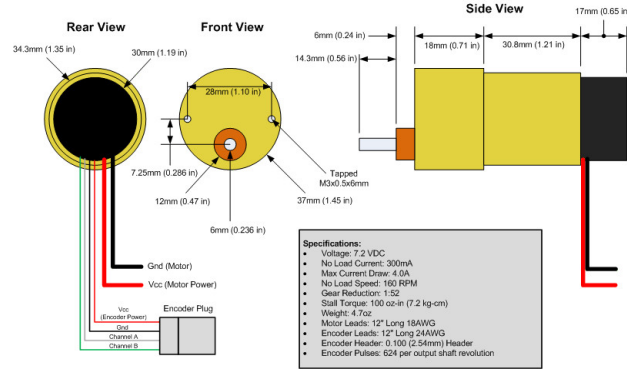


FIG. 21 : Caractéristiques des moteurs à courant continu du Traxter II

Il est également lié à la vitesse de rotation du moteur par l'équation mécanique :

$$J \frac{d\Omega}{dt} = C - C_r$$

où  $J$  est le moment d'inertie du moteur et des éléments accouplés et  $C_r$  un couple résistant (charge, frottement secs ou visqueux).

En tournant, une force contre-électromotrice formalisée par la loi de Lenz est engendrée aux bornes de l'induit proportionnelle à la vitesse de rotation :

$$E = k'\Omega$$

et l'équation de l'induit (en alimentation continue) vaut alors :

$$U = RI_i + E$$

En supposant un rendement idéal, l'ensemble de la puissance électrique est convertie en puissance mécanique si bien que :

$$P = UI_i = EI_i = C\Omega$$

soit  $k = k'$ .

Cette approximation est valable si  $RI_i^2$  est faible au regard des puissances mécaniques en jeu, ce qui est vrai par exemple lorsque le couple résistant est faible. Selon cette approximation on trouve :

$$\Omega = k^{-1}U$$

La vitesse du moteur est alors directement proportionnelle à la tension d'alimentation appliquée sur son induit.

## 5.2 Commande numérique en vitesse du moteur

Il est simple de faire varier la tension moyenne appliquée sur l'induit en utilisant le hacheur monophasé présenté sur la figure 22.

En fermant  $K_2$  et en ouvrant simultanément  $K_1$  durant une proportion  $\alpha$  de chaque période  $T$  la tension moyenne appliquée à l'induit vaut :

$$\langle U \rangle = \alpha V_s$$



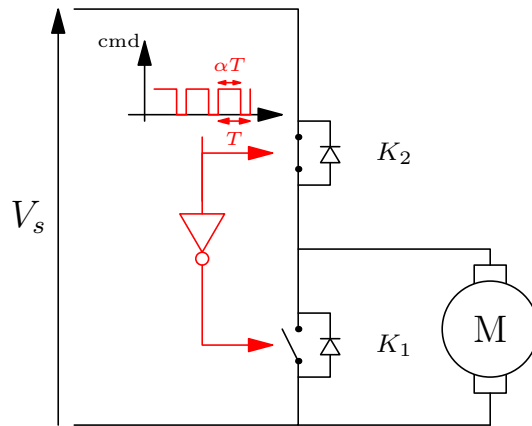


FIG. 22 : Hacheur monophasé

Compte tenu de l'inertie du moteur, les acoups de vitesse ne sont pas perceptibles dès que la période de commande  $T$  est suffisamment faible soit :

$$\Omega = k\alpha V_s$$

Enfin, étant constitué d'un bobinage, l'induit présente une auto-induction non négligeable lorsque le courant d'induit est fluctuant si bien qu'un modèle électrique plus fidèle de l'induit est en fait :


$$u(t) = Ri_i(t) + e(t) + L \frac{di_i(t)}{dt}$$

Là encore, une commande à période suffisamment faible permet de lisser le courant d'induit et donc les variations de couple.

- ✓ **Q.5.2-1** Connectez directement un des moteurs à une alimentation stabilisée (capable de débiter le courant requis) puis relevez la vitesse de rotation pour différents niveaux de tension et différents couples résistants : que constatez-vous à faible tension ? Pourquoi ?
- ✓ **Q.5.2-2** Estimez la vitesse maximale d'avancement du robot Traxter II en km/h.

### 5.2.1 Le BTN8982 (séances 1 et 2)

Le BTN7970 intègre le bras de pont présenté sur la figure 22 dans un unique circuit intégré.

- ✓ **Q.5.2-3** Etudiez la documentation technique du BTN7970  en précisant notamment :
  - les tension et courant maximums d'alimentation ;
  - la tension de commande logique ;
  - la fréquence maximale de commande ;
  - les caractéristiques thermiques importantes (température maximale de fonctionnement, résistance thermique) ;
  - le  $R_{DS(on)}$  des MOSFET ; les temps de montée et de descente ;
  - le rôle de chaque broche du circuit.
- ✓ **Q.5.2-4** Proposez un schéma de câblage du BTN8982 sur plaquette Labdec.

☞ *Attention :*

N'oubliez pas la capacité de filtrage de l'alimentation ainsi que les résistances de protection sur IN et INH destinées à limiter les pics de courant circulant dans ces broches.

- ✓ **Q.5.2-5** Validez par des observations à l'oscilloscope, et à l'aide d'un générateur d'impulsions, le fonctionnement du BTN sur une charge résistive de  $100\Omega$ . Quelle est la puissance moyenne dissipée dans la résistance pour une tension d'alimentation de 7V et un rapport cyclique  $\alpha$  ?

☞ *Attention :*

Une résistance standard est généralement construite pour dissiper un maximum de 0.25W.

### Estimation des dissipations thermiques

Les dissipations thermiques provoquées par les MOSFET en commutation proviennent :

- des pertes de commutation :
- de la résistance du transistor à l'état passant.

Les pertes de commutation sur charge inductive dépendent de la fréquence de commutation, de la tension d'alimentation et des temps d'établissement et d'extinction du courant et sont évaluables lorsque les MOSFET sont modélisés comme des interrupteurs imparfaits. Lorsque  $K_2$  s'ouvre et même sans commande de fermeture sur  $K_1$ , ce dernier devient passant du fait de sa diode de roue libre intrinsèque et du caractère inductif de la charge qui tend à maintenir son courant constant. La tension aux bornes de  $K_2$  vaut donc immédiatement  $V_S$ . Le courant dans  $K_2$  s'éteint aproximativement linéairement sur une durée  $t_f$ .

Lorsque  $K_2$  se ferme,  $K_1$  s'ouvre mais le courant ne circulant pas instantanément dans  $K_2$ , la diode de roue libre intrinsèque à  $K_2$  est en conduction si bien que  $V_{K2} = V_S$ . Le courant croît aproximativement linéairement dans  $K_2$  durant le temps  $t_r$  puis la diode cesse de conduire et la tension  $V_{K2}$  s'annule instantanément. Un raisonnement identique d'applique sur  $K_1$ . La puissance dissipée par les commutations de chaque transistor est alors :

$$P_{comm} = \frac{(t_r + t_f)V_S I_i}{2T}$$

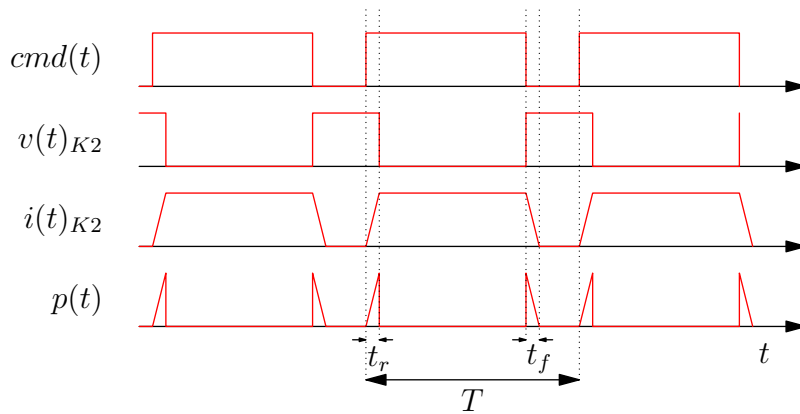


FIG. 23 : Pertes de commutation

En conduction, la puissance dissipée provient de la résistance résiduelle du canal  $R_{DSON}$ . Elle vaut :

$$P_{cond} = \begin{cases} R_{DSON} I_i^2 \alpha & \text{dans le transistor supérieur} \\ R_{DSON} I_i^2 (1 - \alpha) & \text{dans le transistor inférieur} \end{cases}$$

- ✓ **Q.5.2-6** Calculez la puissance dissipée par un BTN pour une tension d'alimentation de 7V, un courant de 4A et une fréquence de commutation de 20kHz.

En l'absence de dissipateur, l'élévation de température de la jonction est obtenue par l'expression :

$$\Delta T[^{\circ}C] = R_{th(j-a)}[^{\circ}C/W] P[W]$$

où  $\Delta T$  est l'élévation de la température de la jonction par rapport à celle du milieu,  $R_{th(j-a)}$  est la résistance thermique jonction-ambiant du composant et P la puissance qu'il dissipe.

- ✓ **Q.5.2-7** Déduisez-en la température du boîtier nu dans un environnement à 20°C.
- ✓ **Q.5.2-8** Reprenez le calcul en utilisant un dissipateur pour boîtier TO-263 (ref 573300 AAVID THERMALLOY) puis discutez de son utilité.
- ✓ **Q.5.2-9** Remplacez la résistance par un des moteurs du Traxter II et validez le fonctionnement de l'ensemble pour différents rapports cycliques du générateur d'impulsion.

**Hacheur 4 quadrants**

Dans l'état actuel, la commande du moteur n'est possible que selon un sens de rotation. Pour pouvoir inverser ce sens, il est nécessaire de mettre en place un hacheur 4 quadrants.

- ✓ **Q.5.2-10** Proposez un schéma de câblage destiné à piloter les moteurs dans les deux sens de rotation. La commande en opposition des deux bras de pont nécessite d'inclure un inverseur dans votre schéma : proposez une référence de composant et insérez-la dans votre schéma.
- ✓ **Q.5.2-11** Donnez l'expression de la tension moyenne aux bornes du moteur en fonction du rapport cyclique du signal de commande du pont.
- ✓ **Q.5.2-12** Câblez et testez ce schéma sur plaquette Labdec et sur une charge résistive de  $100\Omega$ . Quelle est la puissance moyenne dissipée dans la résistance pour une tension d'alimentation de 7V en fonction du rapport cyclique ?
- ✓ **Q.5.2-13** Lorsque les tests sont probants, remplacez la résistance par le moteur et validez la commande pour différents rapports cycliques.
- ✓ **Q.5.2-14** Quelle est la vitesse de rotation attendue pour une commande MLI de rapport cyclique 50% ? Dans ces conditions, est-il nécessaire d'alimenter le moteur ? Quel fonctionnement du BTN8982 pourriez-vous utiliser pour couper l'alimentation logiquement ?

**5.2.2 Commande du pont par microcontrôleur**

<sup>3</sup> Comme expliqué en section 3.5, le module PWM permet de générer un signal MLI de période et de rapport cyclique souhaité.

- ✓ **Q.5.2-15** En utilisant les bibliothèques `periph_pwm.h` et `periph_pwm.c`, produisez un signal PWM de période  $50\mu s$  et de rapport cyclique 25% sur la sortie PEN1H et son complément sur la sortie PEN1L.
- ✓ **Q.5.2-16** Connectez ces sorties à votre hacheur et actionnez le moteur pour différentes valeurs programmées de rapports cycliques.
- ✓ **Q.5.2-17** En vous basant sur la vitesse de rotation obtenue pour une tension de commande continue  $V_s = 7V$ , exprimez le lien entre le rapport cyclique et la vitesse de déplacement du robot.
- ✓ **Q.5.2-18** Déterminez la valeur de prescaler la mieux adaptée à un fonctionnement à des vitesses comprises entre -2.5km/h et 2.5km/h par pas maximal de 0.02km/h.
- ✓ **Q.5.2-19** Quel serait l'inconvénient de choisir un prescaler trop grand ?
- ✓ **Q.5.2-20** Créez une fonction `void SetLeftMotorSpeed(int speed)` permettant d'obtenir une vitesse algébrique de roulage `speed` en mm/s sur le moteur gauche (on suppose ici qu'aucun couple résistant n'existe sur le moteur).
- ✓ **Q.5.2-21** Dupliquez cette fonction pour le second moteur en notant que pour une même commande, ils tournent en sens opposés.
- ✓ **Q.5.2-22** Validez le bon fonctionnement de vos commandes.
- ✓ **Q.5.2-23** Que constatez-vous à nouveau à faible vitesse identique sur chaque moteur ? Comment remédier à ce problème ?

**5.3 Obtention d'une image de la vitesse de rotation des moteurs (séance 3)**

Compte tenu du problème soulevé à la question précédente, il est nécessaire de prévoir un système d'asservissement de la vitesse des moteurs. Cela nécessite d'acquérir une image de leur vitesse réelle de rotation. La vitesse de rotation d'un moteur peut être obtenue en fixant un codeur rotatif à son arbre de rotation. Ces codeurs sont soit optiques, soit basés sur l'effet Hall mais leur principe de fonctionnement est identique : ils sont alimentés et délivrent des impulsions rectangulaires dont la période dépend de

---

<sup>3</sup>Cette partie peut être réalisée à compter de la séance 8 lors de la phase de programmation.

la résolution du capteur (mesurée en impulsions par tour) et varie avec la vitesse de rotation. Dans le cas optique, un disque perforé à intervalle angulaire régulier et à rayon constant est intercalé entre l'émetteur et le récepteur d'une fourche optique. Le faisceau lumineux est passant lorsqu'il se trouve à l'aplomb d'une perforation et bloqué sinon. Les codeurs rotatifs à effet Hall détectent la présence d'un champ magnétique généré par de petits aimants disposés sur le disque identiquement aux perforations des capteurs optiques à l'aplomb d'un capteur.

La figure 24 présente le schéma d'un codeur optique. L'utilisation de deux séries d'encoches concentriques disposées en quadrature permet de doubler la résolution angulaire du capteur mais aussi de discriminer son sens de rotation. Les signaux A et B fournissent une image électrique de la détection de ces encoches. Ainsi, une rotation horaire (resp. anti-horaire) correspondra à un front descendant de B sur état haut (resp. bas) de A ou à un front montant de B sur état bas (resp. haut) de A. Une variable booléenne vraie lorsque le sens de rotation est horaire et fausse sinon pourra donc être définie par :

$$X = B \uparrow . A + B \downarrow . \bar{A}$$

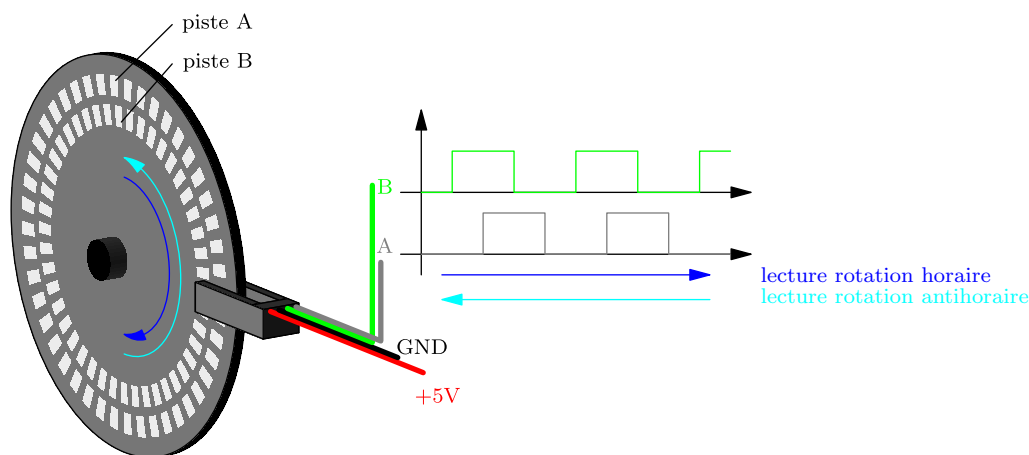


FIG. 24 : Principe du codeur optique

- ✓ **Q.5.3-1** Cablez un des codeurs rotatifs (alimentation et observation des voies A et B sur oscilloscope) puis entraînez le moteur associé en lui appliquant une tension variable. Que constatez-vous ?

### 5.3.1 Acquisition de la vitesse de rotation sur microcontrôleur

<sup>4</sup> L'image de la vitesse du moteur peut être construite par mesure de l'intervalle de temps entre deux impulsions grâce aux entrées d'input-capture IC7 et IC8.

- ✓ **Q.5.3-2** A partir des grandeurs mécaniques et des caractéristiques des codeurs, exprimez la vitesse de roulement du robot en fonction de l'intervalle de temps entre deux impulsions sur une même voie (A ou B). Sur la base de la question Q.5.2-18, déduisez-en les intervalles min et max observables.
- ✓ **Q.5.3-3** Quelle valeur de prescaler vous semble-t-il judicieux d'utiliser ? Pourquoi ?
- ✓ **Q.5.3-4** Que se passe-t-il lorsque la vitesse de rotation s'annule ? Comment résoudre ce problème ?
- ✓ **Q.5.3-5** Sous MPLab®, initialisez le module input-capture, un timer et son prescaler afin d'estimer la vitesse de rotation d'une roue.
- ✓ **Q.5.3-6** Mettez en place la routine d'interruption associée : celle-ci doit calculer l'intervalle entre deux impulsions (voir figure 9) puis fournir une image de la vitesse.
- ✓ **Q.5.3-7** Testez le fonctionnement de votre code à vitesse non nulle en plaçant judicieusement un point d'arrêt dans la routine d'interruption et en utilisant les *watches* de MPLab®.
- ✓ **Q.5.3-8** Ajoutez le code permettant de prendre en charge une vitesse nulle et testez le bon fonctionnement de l'ensemble.

<sup>4</sup>Cette partie peut être réalisée à compter de la séance 8 lors de la phase de programmation.

## 5.4 Réalisation de la carte sous Eagle® (séances 5 à 7)

### ✓ Q.5.4-1 En utilisant :

- les librairies de composants ,
- les consignes de réalisation ,
- le mode d'emploi d'Eagle®,

disponibles sur le cours Moodle, réalisez le schéma de principe et le routage de la carte. Le typon sera réalisé en deux couches avec plan de masse sur la couche supérieure. On prendra soin de positionner les connecteurs en périphérie de la carte.

#### ☞ Informations :

- N'utilisez jamais l'auto-routage ;
- Soyez rigoureux : composants positionnés du bon côté de la carte, pistes dimensionnées en fonction du courant qui y circule, présence de condensateurs de filtrage d'alimentation, angles des pistes à 45°, limitation du nombre de vias ...
- Après validation par l'enseignant, vous transmettez vos fichiers à `david.carton@univ-reims.fr` et/ou `maxime.colas@univ-reims.fr` pour tirage de la carte.

### ✓ Q.5.4-2 Percez, placez puis soudez au four de refusion les composants de surface.

### ✓ Q.5.4-3 Soudez au fer les composants traversants puis testez votre carte sur charge résistive dans un premier temps.

## 5.5 Récupération des consignes par liaison série et tests (séance 8)

En fin de projet, les consignes de commandes des moteurs devront provenir de la carte de contrôle du robot, dite carte IHM, développée par un autre binôme. La mise en place de cette communication nécessite d'utiliser un standard électrique d'interconnexion et un protocole d'échange d'informations compatibles sur l'émetteur et le récepteur. L'interface série RS-232 est le standard électrique et de connexion retenu dans ce projet. Sa connectique au format DB-9 telle que présente sur de nombreux PC est précisée sur la figure 25.

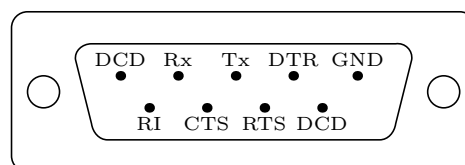


FIG. 25 : Connectique RS232 SUB-D coté PC

Lors de la mise en place d'un échange bidirectionnel sans gestion de trafic, seules les broches Tx, Rx et GND sont utiles. Si l'application se limite à la réception (resp. l'émission) de données sur la carte, seules les broches GND et Tx (resp. Rx) doivent être raccordées.

#### ☞ Attention :

Un PC comme un microcontrôleur sont des *DTE* (Data Terminal Equipment). A ce titre, ils émettent des données sur Tx et en reçoivent sur Rx. Leur interconnexion nécessite donc de croiser les lignes Rx et Tx.

Le protocole de communication exploite le module UART présent dans le micro-contrôleur mais aussi sur la majorité des ordinateurs PC pour transmettre des octets sur une liaison série. Il est convenu d'utiliser la configuration suivante des UART pour l'ensemble des projets (Capteurs, IHM et Commande) :

- 9600 Bauds ;
- 1 bit de stop ;
- pas de contrôle de parité.

### 5.5.1 Format des échanges

Au niveau applicatif, le protocole spécifie le contenu de deux octets transmis consécutivement ou non :

- un octet encode la vitesse tangentielle du robot ;
- un octet encode sa vitesse de lacet (rotation autour de son centre i.e. intersection des diagonales joignant ses roues opposées).

La signification de ces octets est précisées sur la figure 26.

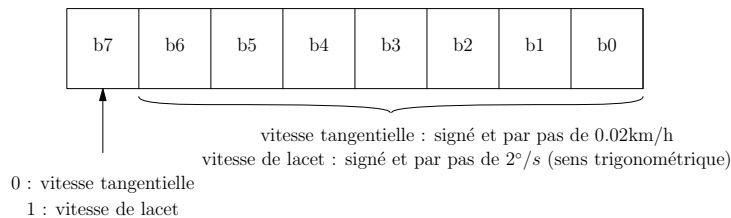


FIG. 26 : Protocole de communication

### 5.5.2 Lecture et conversion des consignes

Le déplacement du robot résulte d'une composition de son déplacement rectiligne localement et d'une rotation autour de son centre.

- ✓ **Q.5.5-1** En considérant nulle la vitesse de déplacement rectiligne  $R$ , calculez la différence de vitesse de rotation des moteurs  $\Delta V$  en fonction de la vitesse de lacet  $\omega_L$ , de la voie du robot  $D$  et du diamètre de ses roues.
- ✓ **Q.5.5-2** En déduire les vitesses de rotation de chaque moteur  $\omega_G$  et  $\omega_R$  lorsque la vitesse rectiligne (ou vitesse tangentielle) n'est plus nulle.
- ✓ **Q.5.5-3** En considérant le protocole décrit sur la figure 26, quelles sont les vitesses de rotation maximales des moteurs atteignables ?
- ✓ **Q.5.5-4** En considérant les informations de vitesse délivrées par le code développé en section Q.5.3-2, pour un couple  $(R, \omega_L)$ , déduire les consignes  $C_G$  et  $C_D$  à transmettre aux moteurs.

### 5.5.3 Interfaçage PC

Comme indiqué en début de section 5.5, tant que la carte IHM n'est pas disponible pour les tests, ceux-ci doivent être menés à l'aide d'un système tierce tel un PC. Toutefois, votre carte microcontrôleur ne peut être directement raccordée à l'un des ports série présent sur un PC car les niveaux électriques de la norme RS-232 ( $\pm 8V$ ) ne sont pas compatibles avec les tensions délivrées ou attendus sur les broches U1ATX et U1ARX (0-5V). La translation de tension bidirectionnelle indispensable à cette interconnexion est fournie par un circuit intégré nommé MAX232 [↗](#).

- ✓ **Q.5.5-5** Après avoir étudié sa documentation technique, câblez le MAX232 sur plaquette Labdec puis raccordez-le au PC : on câblera pour cela un connecteur SUBD-9 femelle.

Dans un premier temps, il est aisé de transmettre des informations (codes ASCII) sur le port RS-232 en utilisant un *terminal*.

- ✓ **Q.5.5-6** Après avoir configuré le terminal TeraTerm, vérifiez que les niveaux de tension et les débits présents sur le MAX232 coté 0-5V sont bien ceux attendus.
- ✓ **Q.5.5-7** En utilisant les informations fournies en section 3.7, insérez le code permettant la configuration de l'UART et la réception des données sur le microcontrôleur.

#### 5.5.4 Utilisation de Matlab pour l'émulation de la carte commande

Matlab® permet d'ouvrir une interface de communication série de type RS-232 présente sur la majorité des ordinateurs PC puis d'y injecter ou d'y lire des octets très simplement. A la différence d'un terminal, les octets transmis ainsi ne sont pas nécessairement des codes ASCII issus d'un clavier mais n'importe quelle valeur binaire. Ainsi :

- `s=serial('COM1','BaudRate', 9600,'Parity','even')` crée un objet `s` associé au port COM1 pour un débit de 9600 bauds et une parité paire. L'interface est ensuite ouverte identiquement à un fichier par `fopen(s)` ;
- la transmission de la valeur numérique `n` au format 8 bits non signés utilise alors `write(s,n,'uint8')` ;
- La lecture de `n` octets non signés sur le port COM1 est réalisée par `val = read(s,n, 'uint8 ')`.

Pour plus d'informations, on consultera la documentation Matworks® [!\[\]\(9dfdaff1d86ba3c1f8353b4d1b61b8c5\_img.jpg\)](#).

- ✓ **Q.5.5-8** Créez un script Matlab permettant d'ouvrir un port de communication puis à communiquer la succession de 2 octets selon le protocole décrit sur la figure 26.
- ✓ **Q.5.5-9** Côté microcontrôleur, développez le code d'interprétation de ces octets et de réglage des consignes moteurs.
- ✓ **Q.5.5-10** Validez le fonctionnement de l'ensemble.

## 5.6 Asservissement de la vitesse de rotation des moteurs (séances 9 et 10)

L'asservissement de la vitesse de rotation des moteurs à la consigne assure un fonctionnement à basse vitesse du robot ainsi que le respect des trajectoires souhaitées (absence de déviation en ligne droite par exemple).

Typiquement, un asservissement est mis en œuvre selon le schéma illustré sur la figure 27 où  $C$  est un correcteur généralement de type PID.

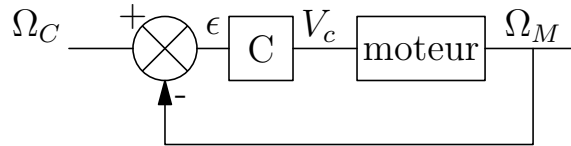


FIG. 27 : Schéma d'asservissement de vitesse

- ✓ **Q.5.6-1** Identifiez chacune des grandeurs du schéma.
- ✓ **Q.5.6-2** Donnez l'expression de Laplace d'un correcteur PID puis expliquez brièvement l'intérêt de chacune des actions (P, I et D).

## 5.7 Identification de la fonction de transfert d'un moteur

Pour valider la période d'échantillonnage choisie et régler ensuite les paramètres du correcteur, il est nécessaire de procéder à l'identification de la fonction de transfert des moteurs. Classiquement, la fonction de transfert  $M(p) = \frac{\Omega}{V}(p)$  d'un moteur à courant continu est modélisée par un second ordre :

$$M(p) = \frac{A}{(1 + \tau_e p)(1 + \tau_m p)}$$

où  $\tau_e = \frac{R}{L}$  est une constante de temps électrique et  $\tau_m = \frac{RJ}{k^2 + Rf}$ ,  $f$  frottement visqueux, est une constante de temps mécanique. Généralement  $\tau_e \ll \tau_m$  si bien que le modèle peut être approché par un premier ordre aux fréquences d'utilisation :

$$M(p) = \frac{A}{(1 + \tau_m p)}$$

Pour identifier  $A$  et  $\tau_m$ , le plus simple est d'appliquer un échelon de tension et de façon synchrone, de stocker à chaque période d'échantillonnage la vitesse mesurée du moteur.

- ✓ **Q.5.7-1** Créez dans votre code une table pouvant accueillir 500 entiers (variables globales) puis écrivez une routine d'interruption déclenchée sur événement d'input-capture dans laquelle vous mémoriserez l'instant des fronts sur la voie A du codeur relié à l'un des moteurs. Seuls les instants d'un front sur trois seront mémorisés. Dans le programme principal, activez la PWM du moteur considéré avec un rapport cyclique de 0.75.
- ✓ **Q.5.7-2** Placez un point d'arrêt dans la routine de telle sorte qu'il ne soit atteint qu'après remplissage de la table puis positionnez un *watch* dans MPLab® sur la table. Exécutez ensuite votre programme.
- ✓ **Q.5.7-3** Lorsque l'exécution stoppe sur le point d'arrêt, sauvez les données du *watch* au format *csv* (menu contextuel : **Export Data|CSV file|Displayed format**).
- ✓ **Q.5.7-4** Exploitez ce fichier dans Matlab® afin d'identifier les paramètres de  $M(p)$ . Vous pouvez réitérer l'opération pour différents rapports cycliques et différentes charges (réalistes) sur l'arbre moteur.

### 5.7.1 Discrétisation de l'asservissement

L'implantation d'une commande numérique sur un microcontrôleur requiert de travailler avec une boucle d'asservissement numérique. Cela nécessite d'agir sur le système à période  $T_e$  constante, et de définir une version discrète du correcteur. Concernant la période d'échantillonnage, plusieurs solutions sont envisageables :



- Se baser sur le rythme de réception des octets de consigne depuis la carte de commande ;
- Définir une horloge propre à la carte moteur quitte à être asynchrone avec les changements de consigne.

- ✓ **Q.5.7-5** Compte tenu de l'identification menée en section 5.7, mais aussi d'autres considérations (capacités de calcul du microcontrôleur, résolution des codeurs...) discutez du choix d'une période d'échantillonnage adaptée à votre système.

L'interface entre le système de commande numérique et le moteur par nature analogique utilise un convertisseur numérique-analogique dont la forme la plus simple est appelée bloqueur d'ordre 0. Celui-ci a pour effet de maintenir en sortie une valeur analogique constante sur toute la période d'échantillonnage égale à la valeur numérique d'entrée. Un bloqueur d'ordre 0 associé à une période d'échantillonnage  $T_e$  a pour fonction de transfert :

$$BOZ(p) = \frac{1 - e^{-T_e p}}{p}$$

et en conséquence, la fonction de transfert discrète de l'ensemble constitué du bloqueur et du moteur est modélisée par :

$$G(z) = (1 - z^{-1}) \mathcal{Z} \left[ \frac{H(p)}{p} \right] = K \frac{1 - z_0}{z - z_0} \text{ avec } z_0 = e^{-\frac{T_e}{\tau}}$$

- ✓ **Q.5.7-6** Sur la base de l'équivalence d'expression de la dérivée dans l'espace en Z et celui de Laplace ( $sT_e = 1 - z^{-1}$ ), donnez une expression en z du correcteur PID.

Dans ce projet, on se limite à une correction PI.

- ✓ **Q.5.7-7** Donnez un organigramme modélisant l'implantation détaillée du correcteur.
- ✓ **Q.5.7-8** Programmez ce correcteur pour chaque moteur.
- ✓ **Q.5.7-9** Adaptez la sortie de votre correcteur pour qu'elle puisse s'interfacer avec le dispositif de commande des moteurs (PWM).

☞ *Attention :*

La programmation d'un correcteur incluant un intégrateur nécessite de prendre en considération les éventuels dépassements de capacité des variables utilisées dans l'algorithme ainsi que les limites min et max admissibles par la commande des moteurs.

- ✓ **Q.5.7-10** Déterminer les valeurs des gain P et I empiriquement ou en vous aidant d'une simulation. Matlab®.
- ✓ **Q.5.7-11** Testez le bon fonctionnement de la boucle d'asservissement d'un moteur à faible vitesse ou en appliquant sur l'axe un couple résistant variable lorsque la consigne est maintenue constante.

## 6 Commande du robot et interface graphique

Si la section 4 constitue les yeux du robot et que la section 5 décrit ses jambes, cette partie s'intéresse à son cerveau. Il s'agit dans ce sous-projet d'atteindre les deux objectifs suivants :

- Concevoir une interface homme-machine destinée à reporter ou fixer les paramètres de fonctionnement et à mettre en action ou stopper le robot ;
- Programmer l'algorithme de suivi de ligne en tirant partie des informations de position délivrées par la partie capteurs et en fournissant la commande à la partie moteurs.

### 6.1 Développement électronique de l'interface homme-machine

L'IHM à développer est constituée d'un afficheur LCD alphanumérique de 3 lignes et 16 caractères par ligne ainsi que de trois boutons.

#### 6.1.1 Afficheur LCD (séances 1 à 3)

L'afficheur est alimenté en 5V et piloté via une interface SPI. Le principe de fonctionnement et le câblage d'une interface SPI sont fournis en section 3.8. l'afficheur est équipé d'un rétro-éclairage indépendant qui doit être soudé et alimenté au travers d'une ou plusieurs résistances comme indiqué en page 2 de la datasheet du composant [↗](#).

- ✓ **Q.6.1-1** Soudez le rétroéclairage aux broches de l'afficheur puis câblez-le sur plaquette Labdec conformément au schéma de la figure 28.

📖 *Indications :*

On connectera un cable nappe à la carte micro-contrôleur sur laquelle on prendra soin d'isoler l'alimentation (utilisée pour alimenter l'afficheur), la masse ainsi que les signaux nécessaires au fonctionnement de l'écran LCD : `SD0_PIC`, `CLK_PIC`, `RB0`, `RB1` et `RB2`. Pour cela, on pourra consulter le schéma électrique de la carte DSPIC [↗](#).

Un soin particulier doit être apporté à ce câblage pour éviter les confusions. En particulier :

- la longueur des fils doit être limitée au maximum ;
- leur couleur doit être choisie avec soin ;
- si plusieurs broches consécutives sont fixées au même potentiel, le plus simple est d'utiliser des embases à broches au pas 2.54mm type [↗](#) que l'on soudera entre-elles ;
- les contacts non utilisés sur la nappe doivent *absolument* être isolés.

#### Validation du fonctionnement

La programmation de l'afficheur repose sur l'intégration à votre projet des fichiers `LCD.c` et `LCD.h` disponibles sur le Moodle II0601. Ces fichiers définissent les fonctions suivantes :

- `void InitLCD(void)` : paramétrage du bus SPI et des I/O utilisées par l'afficheur puis transmission des commandes d'initialisation ;
- `void LCDDataWrite(char c)` : affichage d'un caractère à l'emplacement courant du curseur ;
- `void LCDWriteStr(char * str)` : affichage d'une chaîne de caractères à partir de l'emplacement courant du curseur ;
- `void LCDGoto(unsigned int r, unsigned int c)` : déplacement du curseur en ligne `l` et colonne `c` ;
- `void LCDContrastSet(unsigned int c)` : réglage du contraste de l'écran entre 0 (contraste nul) et 63 (contraste fort) ;

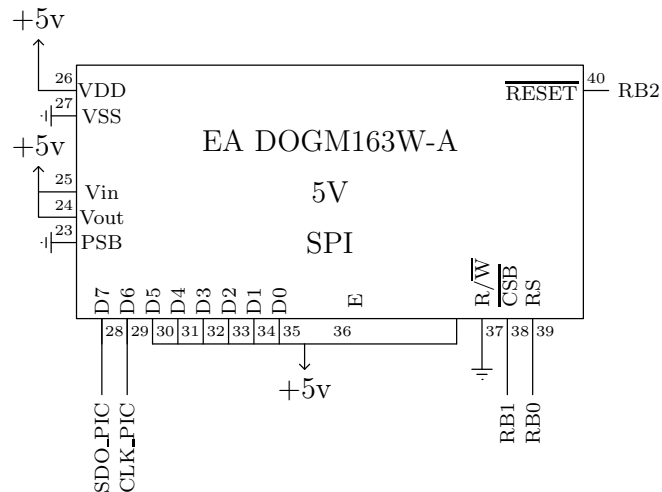


FIG. 28 : Câblage de l'afficheur LCD

- `LCDDisplayOn()` : activation de l'afficheur (et non du rétro-éclairage) ;
- `LCDDisplayOff()` : extinction de l'afficheur (et non du rétro-éclairage) ;
- `LCDClearDisplay()` : effacement de l'écran ;
- `LCDHome()` : retour du curseur en ligne 0 et colonne 0 (haut à gauche de l'écran).

✓ **Q.6.1-2** Créez un programme minimale inscrivant une chaîne de caractères sur la première ligne de l'afficheur.

### 6.1.2 Boutons (séances 1 à 4)

Afin que l'utilisateur puisse interagir avec l'IHM, il est proposé de mettre en place trois boutons dont les fonctionnalités sont les suivantes :

- un bouton `+` destiné à incrémenter des valeurs numériques ou à sélectionner l'entrée précédente dans un menu ;
- un bouton `-` destiné à décrémenter des valeurs numériques ou à sélectionner l'entrée suivante dans un menu ;
- un bouton *entrée* servant à valider une valeur ou à sélectionner une entrée dans un menu.

Un appui prolongé (plus d'une seconde) sur les touches `+` et `-` provoque un comptage continu au rythme de deux incréments par seconde. Ce rythme est triplé après trois secondes de pression continue et ce jusqu'au relâchement du bouton.

✓ **Q.6.1-3** Choisir trois entrées sur le microcontrôleur puis câbler les boutons sur ces entrées.

#### ☞ Indications :

On utilisera des boutons de type *DT6* ainsi qu'un tirage à 5V. Ce dernier sera câblé dans un premier temps à l'aide de résistances qui pourront ensuite être supprimées grâce à l'activation des pullup du micro-contrôleur sur les entrées sélectionnées (voir le paragraphe *CN Control register* de la section 11 du *dsPIC30F Family reference manual* ☞).



✓ **Q.6.1-4** Programmez une fonction permettant d'initialiser les entrées du micro-contrôleur.

✓ **Q.6.1-5** Créez une fonction reportant l'état des boutons à l'écran. Un `X` représentera un bouton pressé et un `0` un bouton relâché.

### 6.1.3 Réalisation de la carte sous Eagle®(séances 5 à 7)

Il est fortement conseillé pour poursuivre de disposer d'une carte électronique sur circuit imprimé.

✓ **Q.6.1-6** En utilisant :

- la *librairie de composants* ,
- les consignes de réalisation prodiguées par vos enseignants,
- le *mode d'emploi d'Eagle®* ,

disponibles sur le cours Moodle, réalisez le schéma de principe et le routage de la carte. Le typon sera réalisé en deux couches avec plan de masse sur la couche supérieure. On prendra soin de positionner les connecteurs en périphérie de la carte. Les boutons de type *DT6* seront placés de sorte à faciliter l'utilisation de l'interface. Enfin, on tâchera de ne pas oublier les connecteurs de communication série de type *SPOX* permettant de s'interfacer aux cartes capteur et moteur.

📖 *Informations :*

- L'afficheur doit impérativement être monté sur support.
- N'utilisez jamais l'auto-routage ;
- Soyez rigoureux : composants positionnés du bon côté de la carte, pistes dimensionnées en fonction du courant qui y circule, présence de condensateurs de filtrage d'alimentation, angles des pistes à 45°, limitation du nombre de vias ...
- Après validation par l'enseignant, vous transmettez vos fichiers à `david.carton@univ-reims.fr` et/ou `maxime.colas@univ-reims.fr` pour tirage de la carte.

✓ **Q.6.1-7** Percez, placez puis soudez les composants avec soin.

## 6.2 Maquettage de l'IHM (séances 8 et 9)

La figure 29 propose un exemple d'enchaînement de menus adaptés au projet. Bien entendu toute évolution justifiée est recevable.

Dans cet exemple, le signe > représente un item de menu actif, c'est à dire pour lequel un appui sur la touche **entrée** provoque :

- une navigation vers un autre menu (e.g. item **Action** du menu **racine**) ;
- l'activation de la zone de saisie associée le cas échéant (e.g. item **Gauche** du menu **Manuel**) ;
- la sélection de l'item le cas échéant (e.g. item **Vit. moteurs** du menu **Aff. etat**) ;
- l'exécution d'une fonction (e.g. item **Marche lente**) du menu **Action**.

Lorsqu'une zone de saisie est activée, la variable numérique associée est modifiée à l'aide des touches + et -.

Le menu **Aff. etat** offre une sélection des mesures à afficher lorsque le robot est en action. Chaque type de mesure utilise un écran complet et il est donc proposé de prévoir un cyclage de l'affichage comme présenté sur la partie droite de la figure 29. Le menu **Manuel** est principalement destiné aux tests de fonctionnement. Il permet de modifier la consigne de chaque moteur entre 0 et 100%. La modification de ces consignes prend effet en cours de réglage, c'est à dire avant même l'appui sur la touche **entree**. Après appui sur cette touche, les valeurs de consigne sélectionnées sont conservées et l'affichage passe immédiatement en mode cyclage.

## 6.3 Programmation de la navigation

### 6.3.1 Gestion des touches

Concernant la gestion des appuis sur les touches, on propose d'utiliser la structure d'état suivante :

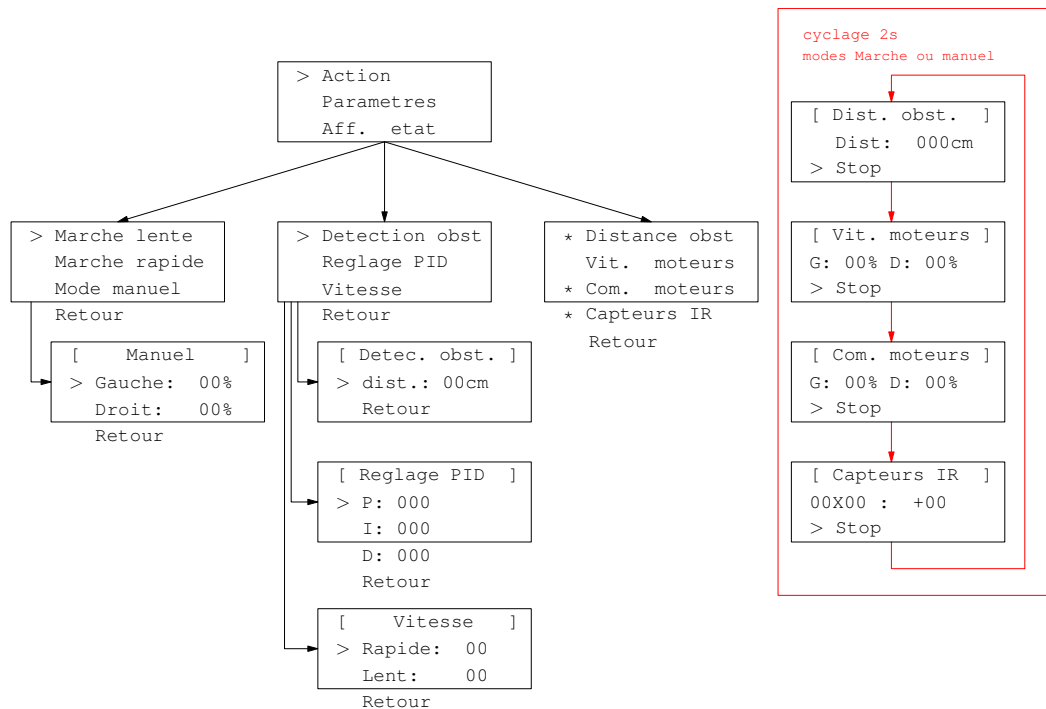


FIG. 29 : Navigation offerte par l'interface graphique

```

1  typedef struct {
2      int upState : 2;
3      int upEdge : 2;
4      int downState : 2;
5      int downEdge : 2;
6      int enterState : 2;
7      int enterEdge : 2;
8  } Keyboard;

```

- `upState`, `downState`, `enterState` indique l'état respectif de ces boutons. Ces états peuvent prendre les valeurs suivantes : 0 : released, 1 : pressed, 2 : pressed1s, 2 : pressed3s où `pressedxs` indique un état enfoncé depuis plus de x secondes.
- `upEdge`, `downEdge`, `enterEdge` indique l'occurrence d'un front sur ces boutons. Les valeurs associées peuvent valoir : 0 : noEdge, 1 : risingEdge, 2 : fallingEdge.

✓ **Q.6.3-1** Créez une fonction `void ihmKeyb(Keyboard *)` non bloquante permettant de mettre à jour un élément de type `Keyboard` qui sera déclaré en portée globale.

📖 *Indications :*

Pour les mesures de durée de pression, on utilisera la librairie `ticks.c` et `ticks.h` qui exploite le timer 1.

### 6.3.2 Gestion de l'afficheur

La programmation de la navigation dans les menus de l'afficheur nécessite une approche structurée afin de conserver un code propre et évolutif. Pour cela, il est proposé de définir deux types :

- le type `Menu` représente un écran de l'IHM :

```

1  typedef struct {
2      char * title;
3      Menu * root;
4      MenuItem * items;
5      short offset : 6;
6      short selected : 2;
7  } Menu;

```

où

- **title** représente le titre du menu. Il est inscrit sur la première ligne entre crochets et ne peut se déplacer. Si le pointeur associé est nul, il n’y a pas de titre au menu et les trois lignes de l’afficheur sont alors utilisées pour présenter les items de menu.
  - **root** est un pointeur vers le menu parent ou celui vers lequel se brancher après appui sur la touche **retour**.
  - **items** pointe sur le tableau des items du menu. Ils sont affichés à partir de la seconde ligne de l’écran (si un titre est présent) et dans leur ordre d’apparition. Les items non visibles peuvent être affichés à l’aide des touches **+** et **-** par scrolling. Par exemple, l’item d’indice 2 du menu [ **Reglage PID** ] n’est initialement pas visible. Un premier appui sur **sur -** déplace le curseur de sélection **>** devant l’item 1 (**I: 000**) puis un second appui remplace l’item d’indice 0 par celui d’indice 1 et l’item d’indice 1 (**P: 000**) par celui d’indice 2 (**D: 000**).
  - **offset** mémorise le décalage vertical compté algébriquement (positif vers le haut) de l’item d’indice 0 par rapport à l’affichage initial du menu. Ainsi, suite aux opérations décrites ci-dessus, l’offset du menu vaut 1 .
  - **selected** indique la ligne de l’élément sélectionné : dans l’exemple précédent, **selected** vaut 2.
- le type **MenuItem** représente une entrée d’un menu :

```

1  typedef struct {
2      char * label;
3      void * action;
4      short type : 2;
5      short checked : 1;
6  } MenuItem;

```

- **label** est la chaîne de caractère affichée à l’écran pour cette entrée du menu ;
- **action** est un pointeur dont la signification dépend de la valeur du champ **type** ;
- **type** peut prendre les valeurs suivantes : 0 : **selectable**, 1 : **function**, 2 : **submenu** ou 3: **value**. **selectable** désigne une entrée commutable par appui sur la touche **entrée**. Lorsqu’une telle entrée est active, elle est précédée d’une étoile (\*) et d’une espace dans le cas contraire. **function** indique qu’**action** est un pointeur vers une fonction à exécuter sur appui de la touche **entrée**. **submenu** indique qu’**action** est un sous-menu à afficher sur appui de la touche **entrée**, enfin **value** indique une valeur numérique modifiable et dans ce cas, **action** pointe vers une structure de type **MenuValue** décrite ci-dessous.
- **checked** indique soit un item sélectionné (type **selectable**) ou une action engagée (type **action**).

Le type **MenuValue** est défini comme suit :

```

1  typedef struct {
2      int tuningValue;
3      int * tunedValue;
4      int min;
5      int max;
6      short type;
7  } MenuValue;

```

où `type` peut valoir :

- `0:direct` : ce paramétrage indique une variable à évolution immédiate , i.e. dont la grandeur pilotée change instantanément lors du réglage ;
- `1:batch` : ce paramétrage indique une variable dont l'évolution ne sera prise en compte qu'après appui sur la touche **entrée**.

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

- ✓ **Q.6.3-2** Déclarez un tableau d'éléments **Menu** au niveau de visibilité global ainsi que les tableaux d'items associés afin d'implémenter l'enchaînement des écrans présentés sur la branche gauche de la figure 29.
- ✓ **Q.6.3-3** Créez une fonction non bloquante `void ihmScreen()` de gestion de l'état de l'afficheur pour ces mêmes menus.

☞ *Indications :*

On utilisera une variable d'indexation statique permettant de mémoriser l'état de la navigation dans l'interface. En cas de sélection d'un item d'action, cet index sera fourni en argument à la fonction **action**.

- ✓ **Q.6.3-4** Ajoutez à votre fonction `void ihmScreen()` la capacité de gestion d'un item de type **value** et testez cette fonctionnalité sur le menu **Manuel**.
- ✓ **Q.6.3-5** Implémentez le menu **Aff. etat** et étendez votre fonction `void ihmScreen()` afin qu'elle gère le type **selectable**.
- ✓ **Q.6.3-6** Terminer le développement de votre fonction par la gestion du cyclage.

☞ *Indications :*

Le cyclage pourra être implémenté sous la forme d'une fonction `void cyclicView(Menu *)` gérant à la fois la liste des états à afficher, l'état courant, le timing de l'affichage et les actions associées à l'item **Stop** en fonction des items sélectionnés dans le menu dont l'adresse est passée en argument.

## 6.4 Pilotage du robot (séance 10)

Lorsque le mode *Marche lente* ou *Marche rapide* est actif, le pilotage du robot consiste à exécuter de manière récurrente une routine d'asservissement numérique destinée à maintenir le robot au centre de la ligne représentant la trajectoire souhaitée. Dans ces conditions, la consigne peut être considérée comme une constante nulle représentant la distance souhaitée entre l'axe du robot et la ligne et les méandres de la ligne au sol comme une perturbation du système régulé.

Le cadencement de la boucle peut être basé sur le rythme de réception des informations capteur.

### 6.4.1 Lecture de l'information de position

Les informations de position en provenance de la carte capteur sont encodées selon le protocole décrit en section 4.3.2 et sur la figure 20.

- ✓ **Q.6.4-1** Programmez une fonction permettant de lire les informations transmises par la carte capteur.

### 6.4.2 Transmission de la commande des moteurs

Les informations de commande des moteurs sont communiquées par liaison série à cette carte selon un protocole en section 5.5.1 ainsi que sur la figure 26.

- ✓ **Q.6.4-2** Proposez une fonction permettant de transmettre ces informations à la carte moteur.

### 6.4.3 Détection d'obstacle et asservissement du robot à la ligne

- ✓ **Q.6.4-3** Formalisez le problème de suivi de ligne comme un problème d'automatique continue en identifiant chaque bloc de votre système.
- ✓ **Q.6.4-4** Donnez un organigramme décrivant l'implantation du suivi de ligne sans oublier d'y intégrer la problématique de la détection d'obstacle.
- ✓ **Q.6.4-5** Programmez cet organigramme dans MPLAB®.
- ✓ **Q.6.4-6** Testez votre carte, si besoin en simulant des entrées comme décrit en section 5.5.