Groupe A4

RAPPORT FINAL DU PROJET D'ELECTRONIQUE

Boudjaoui Sélim, Girodet Florian, Laroche Axel, Ringwald Romain, Simon Arthur, Volot Hélène.

05 Juin 2012

Table des matières

1 Introduction 5
1.1 Introduction 5
1.2 Présentation de l'équipe 6
1.3 Planification des tâches sur l'ensemble du projet 8
1.4 Management de projet 9

2 Déroulement du projet 11
2.1 Objectif de réalisation 11
2.2 Jalon n°1 et Jalon n°2 11

3 Bilan technique 12
3.1 Ressources matérielles 12
3.2 Synoptique matériel du robot 13
3.3 Synoptique logiciel du robot 14

4 Servomoteur et télémètre 16
4.1 Présentation de l'ensemble 16
4.2 Présentation de la partie servomoteur 17
4.2.1 Description matérielle 17
4.2.2 Description logicielle 18
4.2.3 Tests logiciels servomoteur 21
4.3 Présentation de la partie télémètre 22
4.3.1 Description matérielle 22
4.3.2 Description logicielle 23
4.3.3 Tests logiciels télémètre 30
4.4 Mise en place de l'ensemble télémètre/servomoteur 31
4.5 Tests matériels du télémètre et du servomoteur 35
4.6 Tests logiciels de l'ensemble 36
4.7 Problèmes rencontrés et perspectives d'évolution 36

5 Déplacement du robot 37
5.1 Reformulation du cahier des charges 37
5.2 Solution retenue 37
5.3 Solution technique 38
5.3.1 Matériel à disposition 39
5.3.2 Synoptique et branchement de la carte Serializer 40
5.3.3 Elaboration des fonctions 40
5.4 Tests logiciels et matériels 42
5.5 Difficultés rencontrées 42
5.6 Perspectives d'évolution 43

6 Emission et réception sonore 43

6.1 Communication acoustique : la problématique 43
6.2 Communication acoustique : émission sonore 44
6.2.1 Reformulation 44
6.2.2 Solution Hardware 45
6.2.3 Solution Software 48
6.2.4 Résultats 49
6.2.5 Problèmes rencontrés 49
6.2.6 Perspectives d'amélioration 49
6.3 Communication acoustique : émission sonore 49
6.3.1 Reformulation 49
6.3.2 Solution Hardware 50
6.3.3 Solution Software 54
6.3.4 Résultats 55
6.3.5 Problèmes rencontrés 55
6.3.6 Perspectives d'amélioration 56
6.4 Consommation électrique 56
6.4.1 Reformulation 56
6.4.2 Problèmes rencontrés, perspectives d'amélioration 57

7 Mise en place de la télécommande 60

7.1 La problématique 60
7.2 Les ressources matérielles 60
7.3 Synoptique matériel envisagé 62
7.4 Solution technique 63
7.4.1 L'affichage 63
7.4.1.1 Le principe 63
7.4.1.2 Le projet 64
7.4.1.3 Les fonctions utilisées 64
7.4.1.4 Les fonctions par interruption 66
7.4.1.5 Traitement des données 67
7.4.1.6 Tests réalisés 69
7.4.1.7 Difficultés rencontrées, perspectives d'évolution 69
7.4.2 L'UART 1 70
7.4.2.1 Problématique 70
7.4.2.2 Introduction à l'UART1 70
7.4.2.3 Configuration de l'UART1 71
7.4.2.4 Commande du robot 73
7.4.2.5 Envoi d'informations 75
7.4.2.6 Tests logiciels 75
7.4.2.7 Tests matériels 75

7.4.3 La liaison SPI……73
7.4.3.1 Principe……73
7.4.3.2 Réalisation……73
7.4.3.3 Tests……78
7.4.3.4 Perspectives d'évolution ……79

8 Avancement et réalisation du robot……80
8.1 Fonctionnement du robot……80
8.2 Les perspectives d'évolution ……80

9 Bilan des membres de l'équipe ……81
9.1 Organisation de l'équipe ……81
9.2 Bilan des membres de l'équipe et aléas du projet ……82
9.3 Bilan final du projet ……85

10 Annexes ……86

1 Introduction

1.1 Introduction

Suite à la demande de la société client, nous avons mis en place un robot « Track Pinger » répondant au cahier des charges donné. La solution que nous avons apportée s'inscrit bien dans ce cahier des charges. En effet, toutes les solutions techniques que l'équipe a mise en place permettent le déplacement du robot, la détection des obstacles, la mesure des distances et la communication avec les obstacles.

Nous allons vous présenter dans ce rapport l'ensemble des solutions techniques que nous proposons pour répondre à l'appel d'offre du client.
Nous allons tout d'abord rappeler le cahier des charges et les objectifs que nous avons atteint. Nous allons ensuite faire un bilan technique matériel et logiciel du robot.
Dans la suite, nous décrirons les différents modules du robot, leur réalisation, les tests effectués et les résultats obtenus. A la fin de la description de chaque module nous ferons aussi apparaitre les problèmes que nous avons rencontrés et les solutions envisagées.

Enfin, chaque membre de l'équipe fera un bilan personnel du projet au niveau organisationnel puis au niveau technique.

1.2 Présentation de l'équipe

Répartition des postes au niveau organisationnel:

Animateur projet :

Arthur SIMON

Le rôle d'Arthur a été de :

- Coordonner les travaux de l'équipe.
- Connaitre les différents aspects techniques pour assurer les phases d'intégration.
- Concilier les différents membres de l'équipe et trancher en cas d'avis divergents.

Responsable qualité générale - gestionnaire des rendus :

Hélène VOLOT

Le rôle de Hélène a été de :

- Superviser la qualité des rendus.
- Vérifier le travail effectué par rapport aux consignes.
- Mettre en place des outils de travail.

Responsable communication :

Axel LAROCHE

Le rôle d'Axel a été de :

- Prendre contact avec le tuteur d'équipe et le client.
- Ecrire et envoyer les comptes rendus de fin de séances au client.
- Organiser la présentation orale.

Responsable qualité - tests - contrôle technique :

Romain RINGWALD

Le rôle de Romain a été de :

- Evaluer la conformité des réalisations par rapport aux jalons annoncés.
- Veiller à la bonne gestion des ressources matérielles.
- Élaborer la documentation technique.

Responsable planificateur :

Selim BOUDJAOUI

Le rôle de Sélim a été de :

- Planifier les tâches et les jalons du projet.
- S'assurer du bon avancement des taches et du respect des jalons.
- Faire évoluer le planning en fonction de la situation du projet.

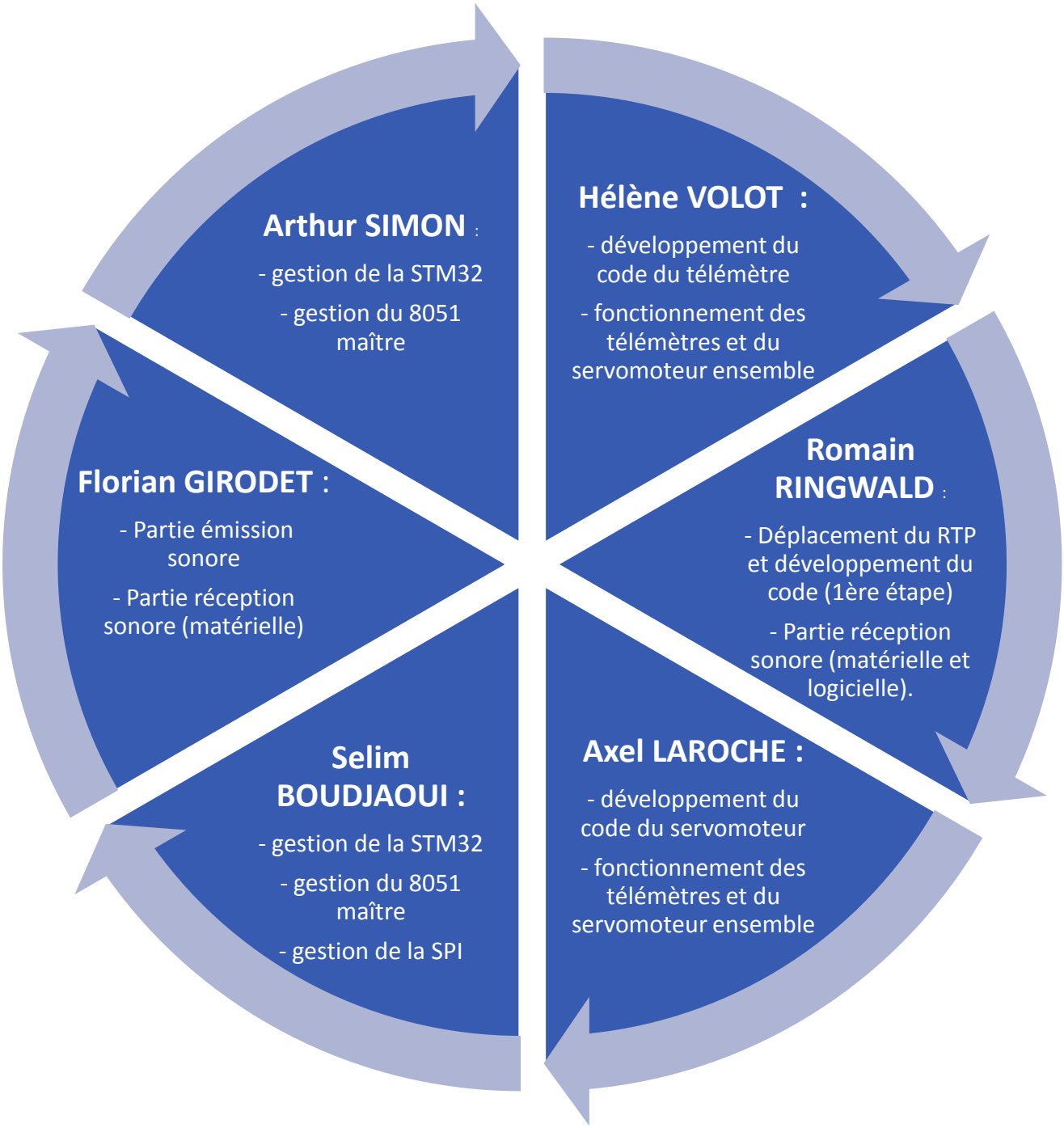
Responsable développement durable :

Florian GIRODET

Le rôle de Florian a été de :

- Surveiller la consommation électrique du robot.
- Optimiser du code pour une meilleure autonomie.

1.3 Planification des tâches sur l'ensemble du projet.



1.4 Management du projet

a) Diagramme QQOQC

	REPONSES	POURQUOI ?	COMBIEN ?
QUOI ?	• Robot «Track Pingers » : C'est un engin autonome capable de se diriger par ses propres moyens	• Les divers éléments du robot sont imposés par le Client • Cette réalisation fait appel à des compétences très diverses. • Communication complexe entre les différents composants	Un seul robot
QUI ?	Pour le Client (Maitre d'ouvrage)	Cet acteur est à la fois décideur pour la mise en œuvre et une aide possible à la résolution	Un seul Client
OU ?	Dans les locaux de l'école CPE Lyon et plus précisément dans un corridor.	La seule gêne serait de considérer les murs du corridor comme un obstacle.	Un corridor
QUAND ?	Projet à rendre début Juin.	Le projet nécessite beaucoup de travail et d'investissement personnel dans un délai très court et sur très peu d'heures.	Un rapport à rendre à une date précise.
COMMENT ?	On dispose de manière aléatoire des obstacles de taille identique. En se déplaçant, le robot devra les détecter sans les percuter et donner la position de chaque obstacle à une centrale de commande.	Tous les obstacles ne sont pas les mêmes, certains contiennent des Pingers et le robot doit pouvoir s'arrêter, les détecter et leur envoyer des signaux	N obstacles

b) Diagramme de Gantt

2 Déroulement du projet

2.1 Objectif de réalisation

Compte tenu du temps limité pour le développement de ce dispositif, nous avons envisagé d'aboutir à la phase 1 comme niveau de performance pour ce robot. Dans cette phase, le robot est piloté au moyen d'une centrale de commande sans fil et est en mesure d'exécuter les fonctions d'interrogation de transpondeur et de décodage de réponses sonores provenant de pingers.

2.2 Jalon n°1 et Jalon n°2 :

Passage du jalon 1 :

Module Déplacement :

Mise en mouvement du robot grâce à la télécommande (BP) et aux fonctions implémentées sur le 8051.

Configuration de la STM32 :

Pour la communication entre la STM32 et l'HyperTerminal, on peut afficher un message prédéfini sur l'HyperTerminal du PC.

Pour la communication entre le PC et la STM32, on est capable d'afficher un caractère tapé sur le clavier sur l'écran tactile.

Détection d'obstacle :

Balayage du servomoteur avec un pas variable et commandé par le PC.

La communication ne marche pas encore avec le télémètre.

Passage du jalon 2 :

La solution du robot proposée dans le jalon 2 est dans la « phase » 1. C'est-à-dire qu'il peut se déplacer seulement sous les ordres d'une télécommande à écran tactile.

Indépendamment, le robot est capable de détecter des obstacles sur 360° à l'aide des 2 télémètres et du servomoteur, de détecter quel obstacle est le plus proche et de renvoyer l'angle et la distance de cette position. Il est capable d'interroger un pinger et récupérer sa réponse. Il renvoie également la commande de mouvement envoyée.

3 Bilan technique

3.1 Ressources matérielles

Le client impose le matériel à utiliser pour la réalisation de ce robot

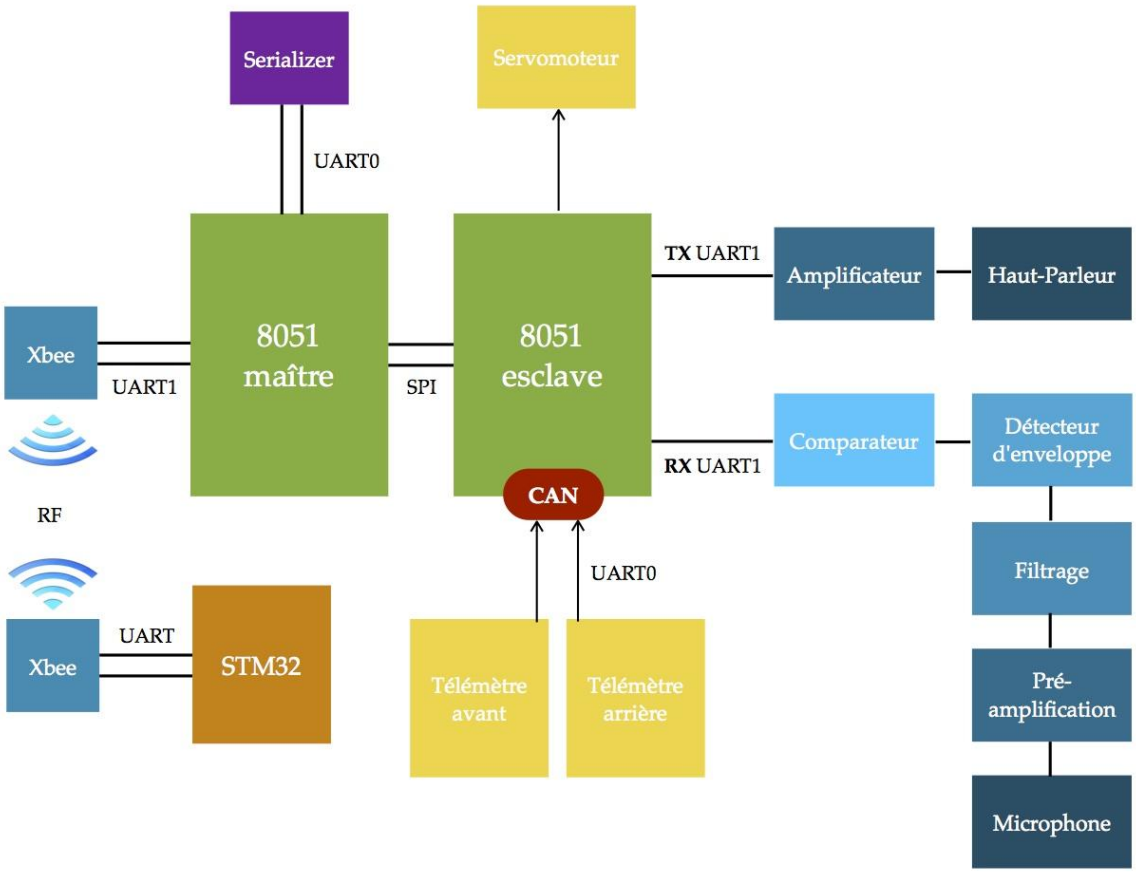
Il fournit cependant la base du robot qui est réalisée : on utilisera la plateforme « Kit Stinger Robot Kit » de Robotics Connection.com. Cette plateforme est constituée d'un châssis en aluminium, de deux roues motrices, et d'une troisième roue pivotante.

Les roues motrices sont toutes deux reliées à un motoréducteur couplé à un codeur incrémental en quadrature.

A cela s'ajoute d'autres matériels :

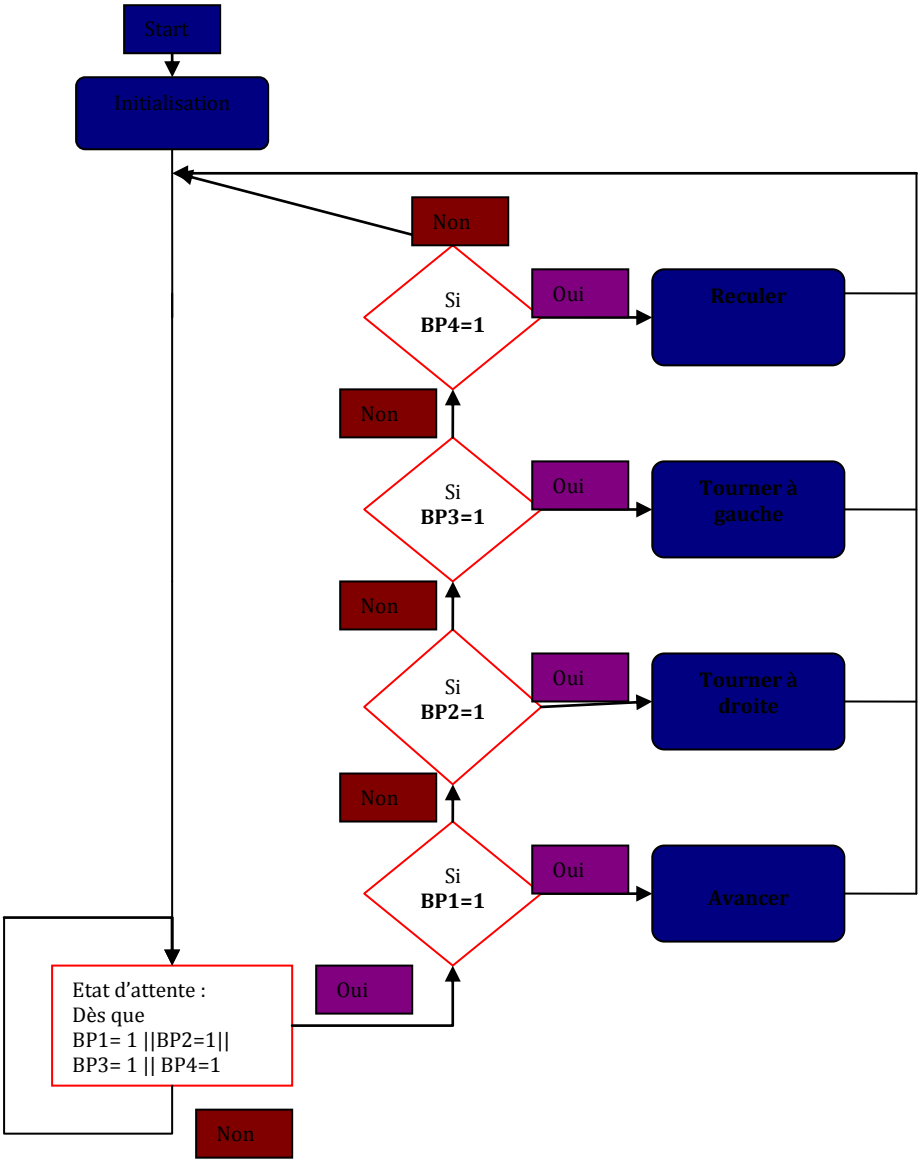
- La carte Serializer (Serializer Robot Controller).
- 2 télémètres GP2Y0A02YK0F de Sharp
- 1 servomoteur de type HS-322HD
- 1 microphone à capsule electret de référence : Panasonic WM-62A.
- 1 haut-parleur électromagnétique de 8ohms – 50mm de diamètre.
- 1 lampe LED du fabricant CREE.
- Cartes 8051F020 TB (4 disponibles mais 2 maximum sur le robot).
- Carte STM3210C (utilisée pour l'écran tactile)
- Module radio XBEE « XB24-ACI-001 »

3.2 Synoptique matériel du robot

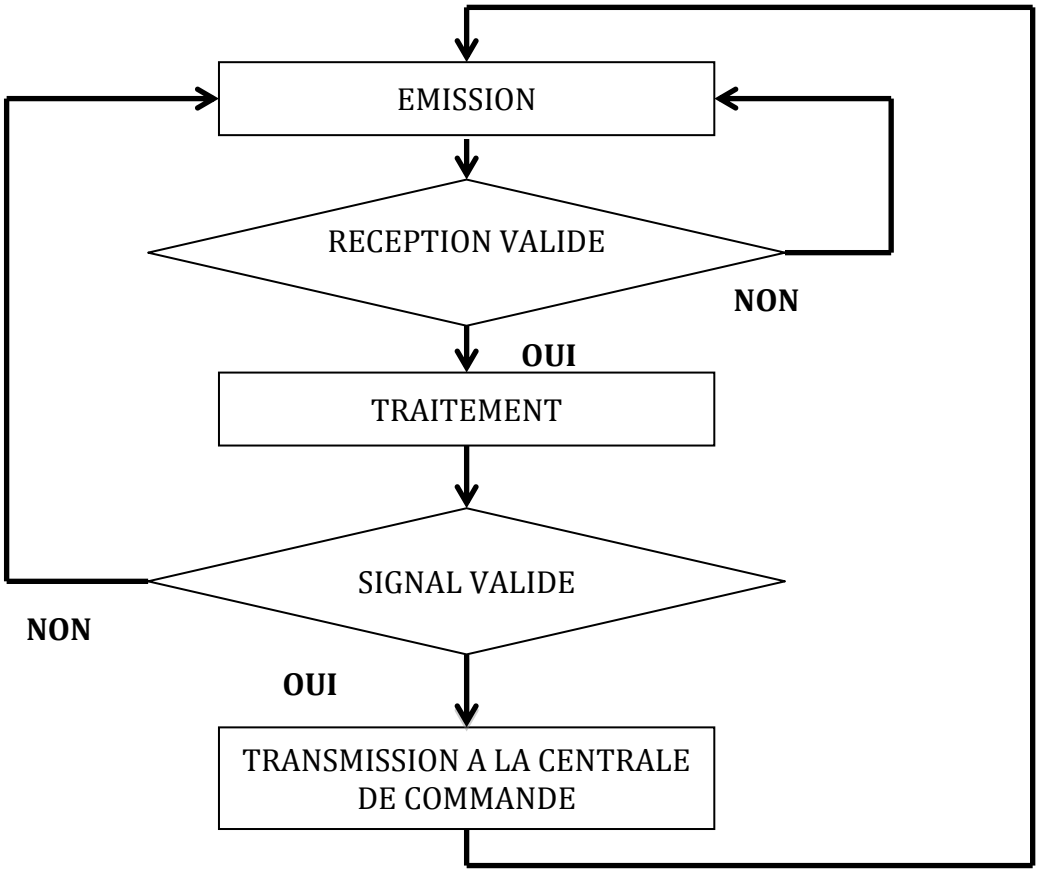


3.3 Synoptique logiciel

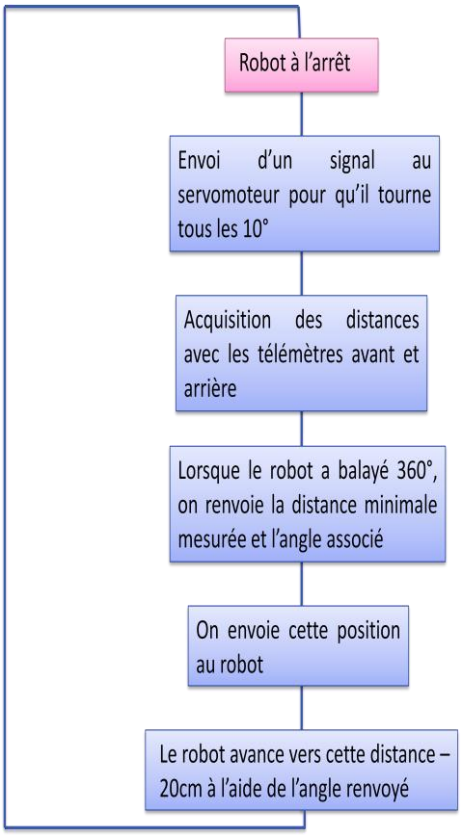
Synoptique logiciel du déplacement du robot :



Synoptique logiciel de l'émission / réception :



Synoptique logiciel de la détection d'obstacle:



4 Module Servomoteur/Télémètre

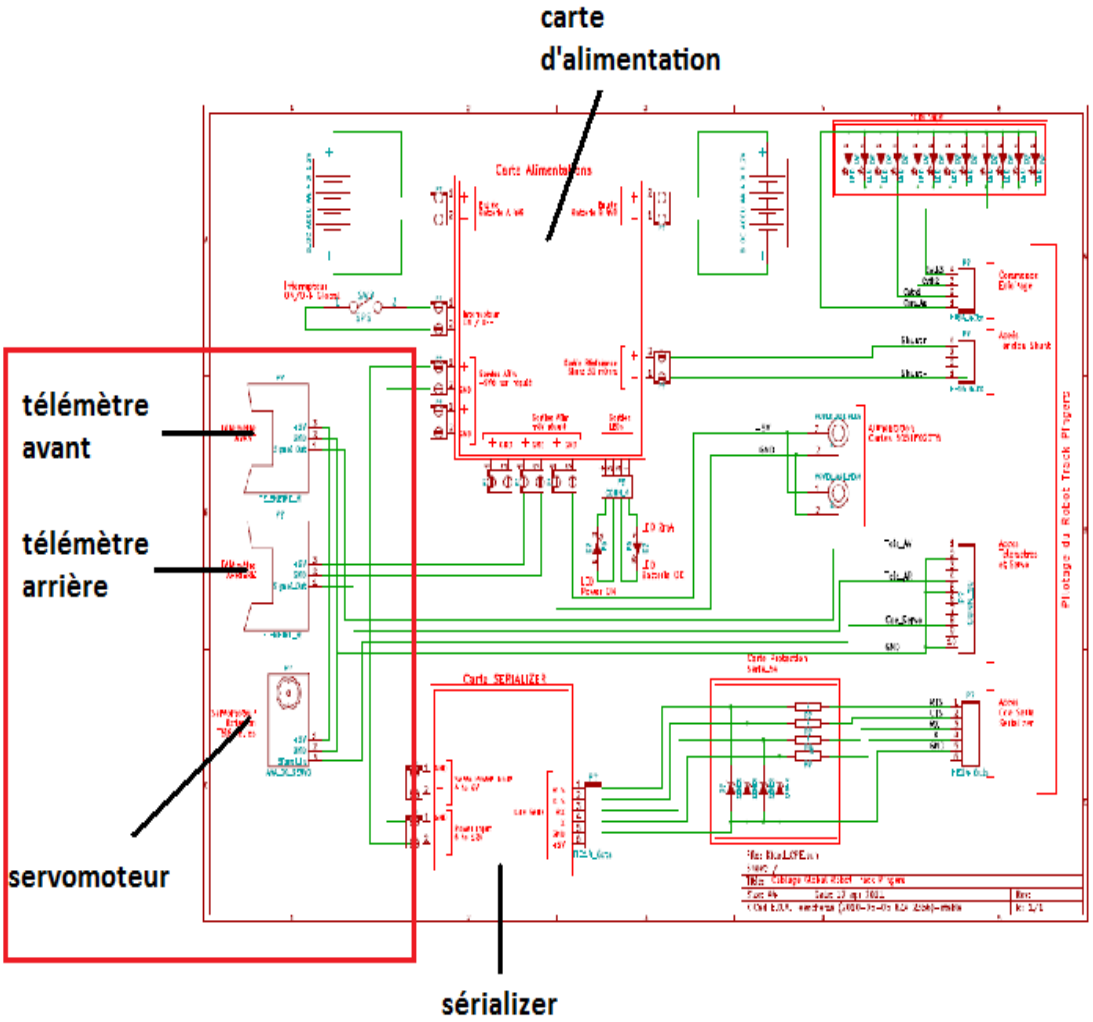
4.1 Présentation de l'ensemble télémètre et servomoteur

(Partie rédigée par Axel Laroche et Hélène Volot)

Nous allons présenter dans cette partie l'ensemble télémètre et servomoteur.

On convertit les données des télémètres à l'aide du convertisseur analogique numérique du 8051 esclave.

On utilise de plus le multiplexeur autour de l'ADC0 pour faire fonctionner les télémètres avant et arrière ensemble.



Ce schéma localise les différents éléments étudiés.

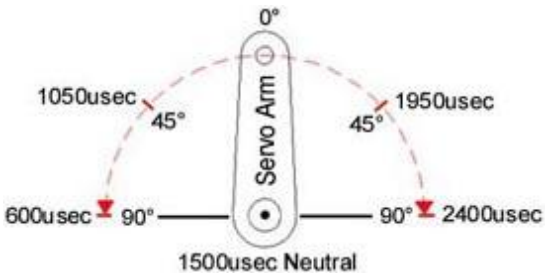
On fait ensuite un synoptique de notre partie :

4.2 Présentation du servomoteur

(Partie rédigée par Axel Laroche)

4.2.1 Description matérielle

Afin d'effectuer la détection de l'environnement autour du robot, nous avons besoin de mesurer les distances à 360° autour du robot. Pour ce balayage, nous avons utilisé un servomoteur de type HS-322HD. Le connecteur de la plateforme du robot possède 2 broches pour le servomoteur : une pour la masse (GND) et une pour la commande du servomoteur. L'alimentation en +5V du servomoteur est déjà câblée sur la plateforme.



Pour commander le servomoteur, il faut appliquer en entrée de celui-ci un échelon de tension compris en 3 et 5V, de largeur d'impulsion adaptée à l'angle de rotation que l'on veut faire adopter au servomoteur.
La durée de ce signal doit être comprise entre 600µs et 2400µs. Nous tacherons de rester dans cet intervalle afin de ne pas demander au servomoteur d'effectuer une rotation hors de sa butée, ce qui entrainerait une hausse de la consommation électrique.

4.2.2 Description logicielle

Le servomoteur effectue une rotation dont l'angle dépend de la largeur de l'impulsion qu'on lui fournit. Afin de générer ce signal de largeur d'impulsion variable, nous avons recours au PCA (Programmable Counter Array) intégré au 8051.
Pour effectuer la rotation du servomoteur, nous utiliserons trois fonctions :

1. Initialisation du PCA : void PCA_init()

```
void PCA_init (void)
{
   PCA0CN = 0x00;  // desactivation du PCA
   PCA0MD = 0x09;  // base temps = SYSCLK
   PCA0CPM0 = 0xC2;  // mode PCA : modulation de largeur d'impulsion sur 16 bit
}
```

Le module PCA peut être paramétré pour adopter différents modes : nous avons ici choisi le mode 16-bit Pulse Width Modulator, le mode 8-bit n'est pas suffisant car nous devons compter jusqu'à 2.4ms, donc 12 bits sont nécessaires. Pour cela, nous utilisons le registre PCA0MD, comme indiqué dans la documentation technique.

PCA0CPM0 = 0xC2 = 1100 0010

Table 23.2. PCA0CPM Register Settings for PCA Capture/Compare Modules

PWM16	ECOM	CAPP	CAPN	MAT	TOG	PWM	ECCF	Operation Mode
1	1	0	0	X	0	1	X	16-Bit Pulse Width Modulator

De plus, nous avons choisi de prendre l'horloge système pour l'horloge du PCA, en paramétrant le registre PCA0MD.

PCA0MD = 0x09 = 0000 1001

Table 23.1. PCA Timebase Input Options

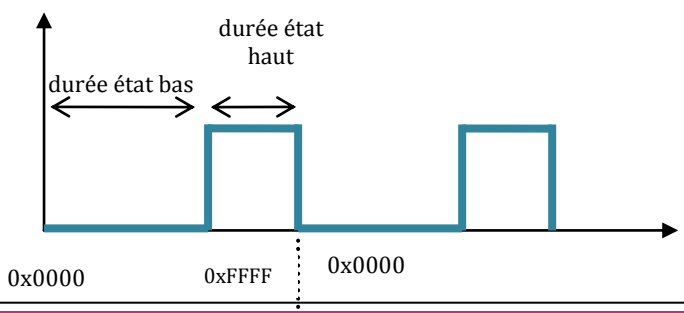
CPS2	CPS1	CPS0	Timebase
1	0	0	System clock

2. Génération de l'impulsion : void pulse (int duree)

```
void pulse (int duree)
{
  long dutycycle = 65536 - duree*221184/10000; // calcul du dutycyle (durée à l'état bas) en fonction de la durée

   char CPL, CPH;  // conversion en hexadecimal

        CPL = (char)dutycycle; // 8 bits de poids faible dans la valeur low de PCA0CP0
        CPH = (char)(dutycycle>>8); // décalage pour obtenir le 8 bits de poids fort dans la valeur high de PCA0CP0

   PCA0L = 0x00;
   PCA0H = 0x00; // initialisation du timer à 0

   PCA0CPL0 = CPL;
   PCA0CPH0 = CPH;     // définition de la valeur à l'état bas voulue dans le registre PCA0CP0
   EA=1;          // activation des interruptions
   PCA0CN |= 0x40;  // activation du  PCA
}
```

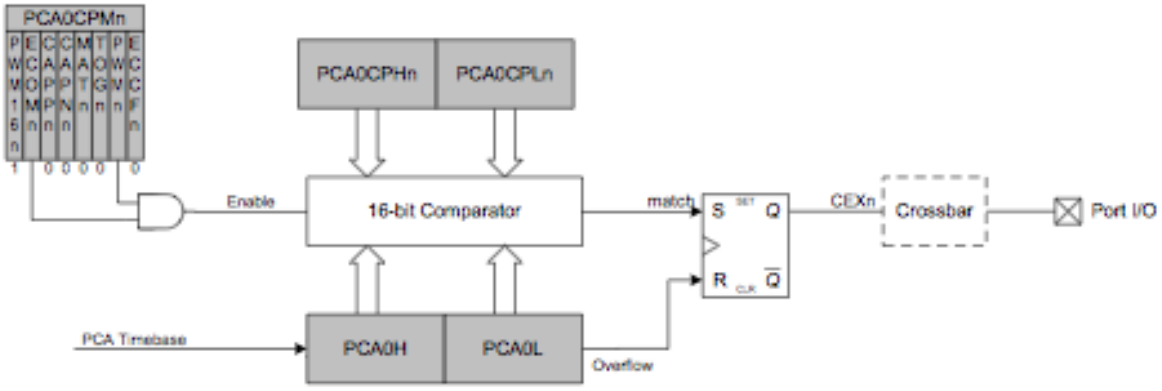
Le PCA est un compteur qui va compter de 0x0000 à 0xFFFF. Lorsque le compteur est enclenché, on compare sa valeur de comptage avec le registre PCA0CP0 : si elles sont égales, alors la sortie passe à l'état haut. À l'overflow du compteur (passage de 0xFFFF à 0x0000), la sortie repasse à l'état bas.

Nous stockons donc dans les registres PCA0CPH0 et PCA0CPL0 (respectivement octet de poids fort et de poids faible de PCA0CP0), la durée à l'état bas du rapport cycle.



Nous devons donc calculer la valeur à stocker dans le registre PCA0CP0 en fonction de la durée de l'impulsion à générer qui est donc la période moins la durée à l'état haut.
Cette valeur est ensuite convertie en hexadécimal puis stockée dans les registres PCA0CPH0 et PCA0CPL0. Nous initialisons également notre compteur à 0 avec les registres PCA0L et PCA0H. Enfin nous activons les interruptions puis le PCA.
La sortie du module PCA est reliée aux ports du 8051 via le crossbar. Nous y accédons sur le port P0.2. Pour cela, nous avons utilisé les registres XBR0 pour router la sortie CEX0 du PCA sur le port P0.2 et activer le crossbar.

Figure 23.9. PCA 16-Bit PWM Mode



3. Rotation du servomoteur : void rotation (int angle)

```
void rotation (int angle)
{
   int duree = 0;
   if (angle>=0 && angle <=180) duree = 600 + 10*angle; // conversion de l'angle en temps
   else duree = 600; // si angle hors intervalle on reste à la position 0

         pulse(duree); // génération de l'impulsion
}
```

La fonction *rotation* nous permet d'effectuer la rotation du servomoteur d'un angle donné en paramètre. Il nous suffit pour cela de convertir l'angle donné en une durée que l'on passera ensuite en paramètre de la fonction *pulse* créee précédemment.
L'angle de rotation est un angle compris entre 0° (durée d'impulsion de 600µs) et 180° (durée d'impulsion de 2400µs).

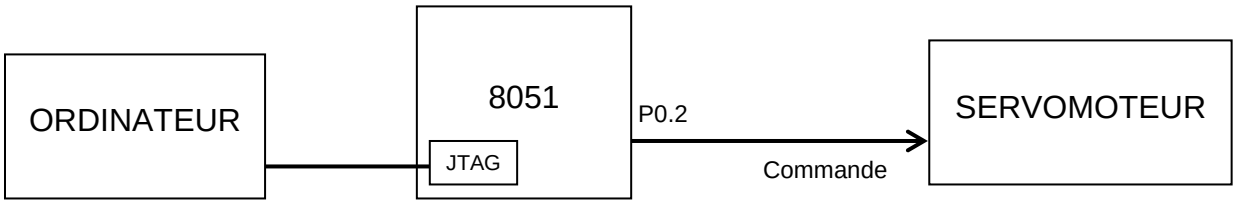
On remarque que la durée pour un angle correspond à la durée pour 0° à laquelle on ajoute 10 fois l'angle voulu. Nous obtenons donc la formule suivante :

duree = 600 + 10*angle

Si l'angle demandé n'est pas dans l'intervalle [0° ; 180°], nous resterons à la position 0° pour ne pas demander au servomoteur de tourner au-delà de sa butée. Cela entrainerait une consommation électrique excessive de la part du servomoteur.

4.2.3 Tests logiciels servomoteur

Après avoir réalisé les fonctions nécessaires au fonctionnement, nous avons effectué les premiers tests logiciels sur le servomoteur.



Dans une boucle infinie, nous faisons réaliser différents rotations successives au servomoteur. A chaque fois la rotation effectuée correspond bien à celle demandée.

Nous avons ensuite demandé au servomoteur de réaliser un balayage complet des 180° avec un pas variable. Pour visualiser le pas, nous avons réalisé une fonction de temporisation *sleep* expliquée plus loin (cf **4.4**).

Ces tests étaient concluants : le balayage s'effectue comme demandé. Pour un pas de 10° nous visualisons 18 rotations successives du servomoteur, pour un pas de 20°, 9 rotations, etc.

4.3 Présentation du télémètre

(Partie rédigée par Hélène Volot)

4.3.1 Description matérielle :

1. Le Télémètre :

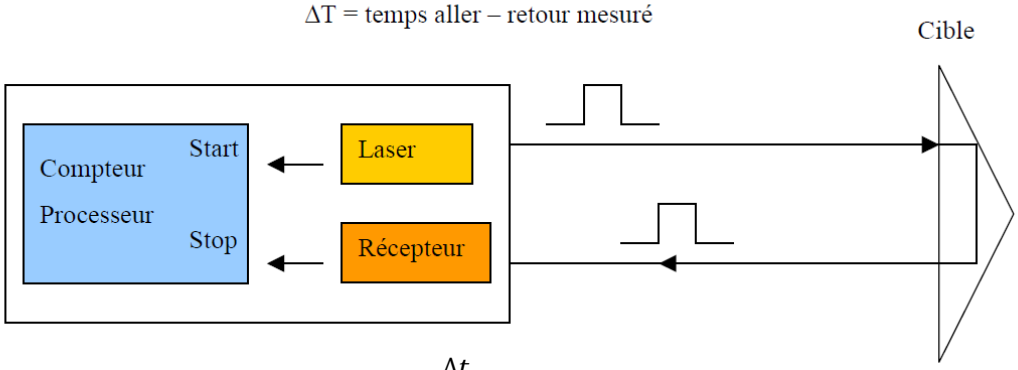
■Features

1. Distance measuring range : 20 to 150 cm
2. Analog output type
3. Package size : 29.5×13×21.6 mm
4. Consumption current : Typ. 33 mA
5. Supply voltage : 4.5 to 5.5 V

Le télémètre utilisé est un télémètre GP2Y0A02YK0F. Nous utiliserons deux télémètres montés dos à dos sur le servomoteur ce qui va ainsi permettre une mesure des distances à 360° autour du robot. Les télémètres utilisés permettent de déterminer des distances comprises entre 20 cm et 150 cm.

D = (c.ΔT) / 2 où c = vitesse de propagation de la lumière dans le vide

ΔT = temps aller – retour mesuré



On calculera la distance grâce a la relation $d = \frac{\Delta t}{2.c}$ où c est la vitesse de déplacement.

De plus, pour des résultats de mesure des distances optimaux, nous utilisons des obstacles de couleur blanche pour avoir une meilleure réflectance du matériau.

Chaque télémètre possède un connecteur à trois broches : deux sont utilisées en entrée pour l'alimentation (V_{cc} et GND) et une broche constitue la sortie V_{out}, qui fournit un signal analogique correspondant à la distance mesurée. Nous alimenterons nos télémètres avec une tension V_{cc} = +5V.

2. Le convertisseur analogique numérique, le CAN :

Il faudra aussi prévoir un convertisseur analogique numérique entre la sortie des télémètres et l'entrée du 8051 afin d'adapter les signaux où l'un a une plage de [-Vref, +Vref] et l'autre une plage de [0, Vref].
J'ai choisi d'utiliser l'ADC0 du 8051 pour avoir une meilleure précision dans les résultats. J'utilise l'ADC0 sur 12 bits et j'ai choisi de fonctionner à l'aide de l'interruption 15 de l'ADC0.

3. Le filtre RC

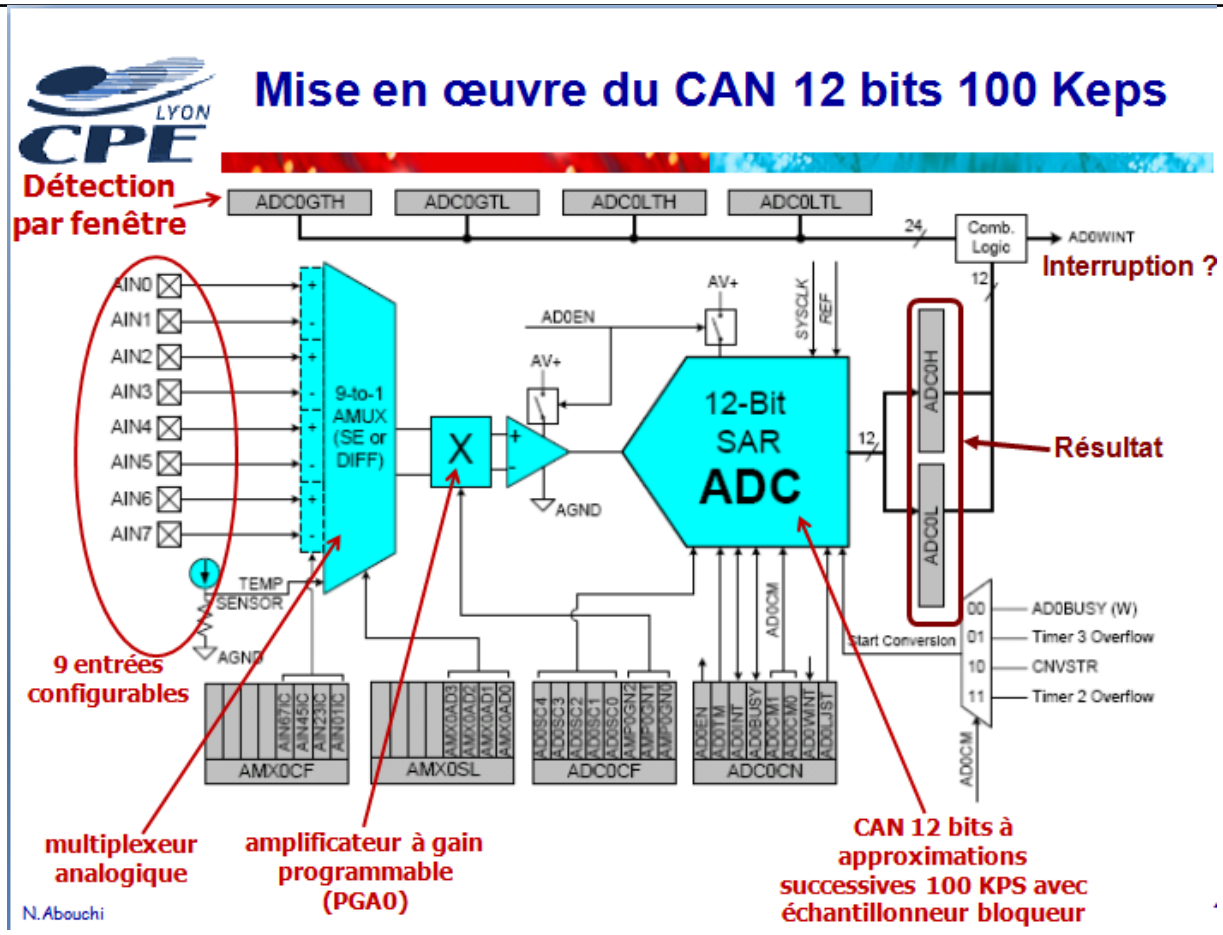
Au vu des tests effectués sur le télémètre, on constate que la tension que l'on récupère n'est pas stable. Pour lutter contre ce problème, on utilisera un filtre RC.

4.3.2 Description logicielle :

Le but de cette partie est de configurer l'ADCO du 8051 afin de récupérer la valeur renvoyée par le télémètre, la convertir en Volts puis à l'aide de calculs la convertir en centimètres. On utilise l'UART0 pour faire communiquer le PC à l'HyperTerminal.
Pour cela, il faut configurer le 8051 afin de pouvoir réceptionner les données renvoyées par le télémètre.

C'est le rôle de la fonction **void ADC0_init(void)**

```
void ADC0_init(void)
{
   AMX0CF = 0x00;         // on sélectionne le mode "single-ended"
   AMX0SL =0x00;          // on sélectionne la voix AIN0.0 pour le Vout du télémetre
                         // (port A32 du 8051)
   ADC0CF =0x50;                  // on choisit un GAIN=1
   ADC0CN = 0x41;         // on utilise le mode 'low power track      '
   REF0CN =0x03;          // on initialise la reference de tension
   EIE2 |= 0x02;          // EADC0=1 -> on n'autorise pas l'acces à l'interruption générée par la fin de la
conversion de ADC1
}
```



D'après le schéma précédent, l'initialisation de l'ADC0 se fait en plusieurs étapes :

- Il faut d'abord choisir notre entrée, c'est-à-dire le port où l'on va brancher le Vout du télémètre. On choisit AIN0.0.
- Il faut ensuite s'intéresser au multiplexeur analogique. Dans un premier temps, on teste le code sur un seul télémètre donc on ne l'utilise pas. On s'en est servi lorsqu'on a mis ensemble les deux télémètres.
- On a ensuite choisi un gain pour le PGA0 de 1
- Enfin, on a choisi de travailler en mode « single ended » soit le mode monostable.

J'ai aussi choisi de travailler par interruptions en utilisant AD0BUSY.

C'est pourquoi, j'ai fait une fonction de début de conversion qui va activer l'ADC0, autoriser les interruptions et ainsi permettre de démarrer la conversion.

C'est la fonction **void Debut_conversion(void)**

```
void Debut_conversion(void)
{          // ADC0N: registre de contrôle de ADC0

 conversion=0;              // on commence la conversion

ADC0CN |= 0x80;             //on active l'ADC0  qui est prêt pour une conversion de données

EA=1;                       //on active les interruptions

ADC0CN |= 0x10;             //on demande la conversion en initialisant AD0BUSY (bit4)  à 1
}
```

On demande bien dans cette fonction une conversion en utilisant AD0BUSY.

Une fois que l'ADC0 est activé et que l'on autorise les interruptions (EA=1), on va dans la fonction **void Fin_Conversion_ADC0(void).**

```
void Fin_Conversion_ADC0(void) interrupt 15    // fonction qui se met en route quand on a le signal de fin de conversion
{
  unsigned long valconv, calctemp = 0;                  // on utilise un unsigned pour avoir une capacité maximale de 32 bits
  valconv=ADC0;                          // valconv récupère l'octet ADC0
                                  // Vin=(Code*Vref)/(Gain*65520)
                                  // on a choisi un gain=1
                                  // dans cette formule le "Code" est représenté par la variable "valconv"
  // or Vref=2400mVolts  (on convertit en mV pour éviter d'avoir des float)
  calctemp=valconv*2400;

  // on utilise AD0LJST=1
  //   [ 2400/(FFF0H) ] =  [ 2400/65520 ] = 1/27
  // on en déduit :
  Vin=ADC0/27;

 // printf("Tension en V: %d\n",Vin);

  ADC0CN &= 0xDF;    //ADC0 est actif et prêt pour une conversion
                     // on choisit le mode low power track car PGA change fréquement

  ADC0CN &= 0x7F;          // on désactive l'ADC0 car c'est la fin de la conversion
                           // on utilise la conversion de type 1 c'est à dire que l'on écrit '1' au bit AD0BUSY de ADC0CN.
                           // Pendant la conversion, AD0BUSY est à '1' et retourne à '0' quand la conversion est terminée

/* on va ainsi procéder en 4 étapes :
                    - on ecrit '0' à AD1INT
                    - on écrit '1' to AD1BUSY
                    - on scrute AD1INT pour '1'
                    - Process ADC1 data */
   conversion=1;            //ADC0 ne fonctionne plus
}
```

L'ADC0 récupère un octet renvoyé par le télémètre.

Dans cette fonction, on récupère cet octet stocké dans l'ADC0.

On convertit ensuite cette valeur en millivolts. Pour ce faire, j'ai choisi de centrer cet octet sur la gauche. En effet, après avoir enregistré une valeur :

ADC0 vaut FFFH (111111111111b)

Comme le CAN que l'on a choisi est sur 12 bits, il faut choisir de centrer ces 12 bits à droite ou à gauche. J'ai choisi de les justifier à gauche soit « left justified » donc :

ADOLJST = 1

Alors : ADC0H:ADC0L = FFF0H (11111111-11110000b)

Une fois que l'on a récupéré cet octet, il faut le convertir en millivolts.

Pour cela, on utilise la formule :

$$ADC0Code = Vin \times \frac{Gain}{VREF} \times 2^n$$

Avec : - n=12 qui est le nombre de bits

- Gain =1 : c'est la valeur à laquelle on l'a initialisé
- ADC0Code est l'octet que l'on a récupéré
- Vref est la tension de référence et vaut 2400 mVolts
- Vin est la tension que l'on recherche après la conversion

On a donc : $Vin = \frac{ADC0Code.Vref}{Gain.2^n}$

On utilise ensuite la table de conversion suivante :

AIN0 – AGND (Volts)	ADC0H:ADC0L (AD0LJST=0)	ADC0H:ADC0L (AD0LJST=1)
$VREF \times \frac{4095}{4096}$	0FFFH	FFF0H
$\frac{VREF}{2}$	0800H	8000H
$VREF \times \frac{2047}{4096}$	07FFH	7FF0H
0	0000H	0000H

On est dans ce cas là avec AD0LJST=1

On a donc $\boldsymbol{Vin = \frac{ADC0.2400}{65521} = \frac{ADC0}{27}}$

On a donc notre octet converti par le CAN en mVolts.

On désactive ensuite l'ADC0 pour une nouvelle conversion.

L'interruption utilisée est la 15 que l'on utilise avec ADC0BUSY suivant les indications de la datasheet.

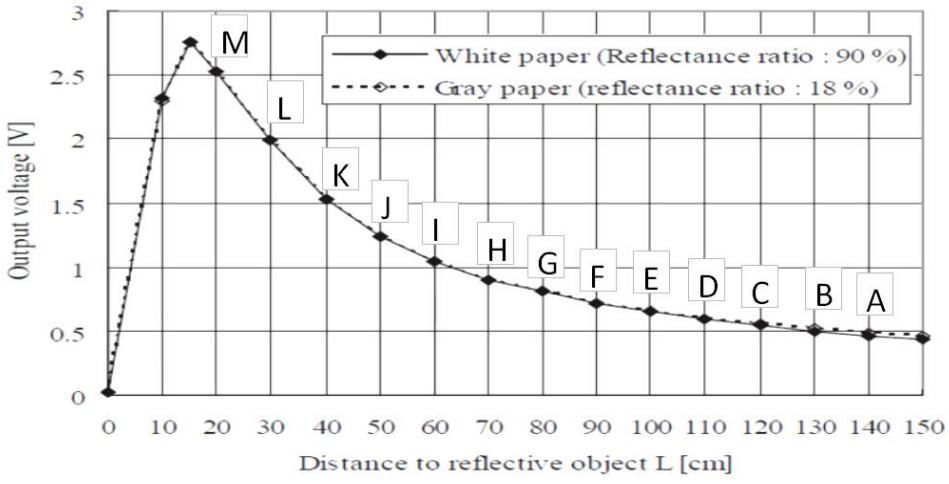
Il faut ensuite convertir cette valeur en millivolts en mètres.

C'est le rôle de la fonction **double Distance_telemetre(unsigned int Vin)**

```
double Distance_telemetre(unsigned int Vin)
{
  double Distance=0;// distance de objet/telemetre
  if(Vin > 2500 || Vin < 400)
   {
     return -1;
   }
   else
   {
     if(Vin < 500 && Vin > 400)
     {
        Distance=(923-Vin)/(3);
        return Distance;
     }
     else
     {
        if(Vin < 600 && Vin > 500)
        {
           Distance=(1126-Vin)/(5);
           return Distance;
```

```
        }
        else
        {
          if(Vin < 800 && Vin > 600)
          {
            Distance=(1225-Vin)/(6);
            return Distance;
          }
          else
          {
            if(Vin < 1200 && Vin > 800)
            {
              Distance=(1805-Vin)/(13);
              return Distance;
            }
            else
            {
              if(Vin < 1700 && Vin > 1200)
              {
                Distance=(2729-Vin)/(30);
                return Distance;
              }
              else
              {
                if(Vin < 2500 && Vin > 1700)
                {
                  Distance=(3339-Vin)/(48);
                    return Distance;
                }
              }}}}}
}
```

On détermine la distance en fonction de la tension grâce à la courbe donnée dans la datasheet.



On va travailler sur des segments de droites et on va faire des approximations linéaires de la courbe.

On travaille entre les segments [AB] puis [BCD] puis [DEFGH] puis [HI] puis [IJK] puis [LKM].

On a considéré chaque segment comme une droite d'équation :

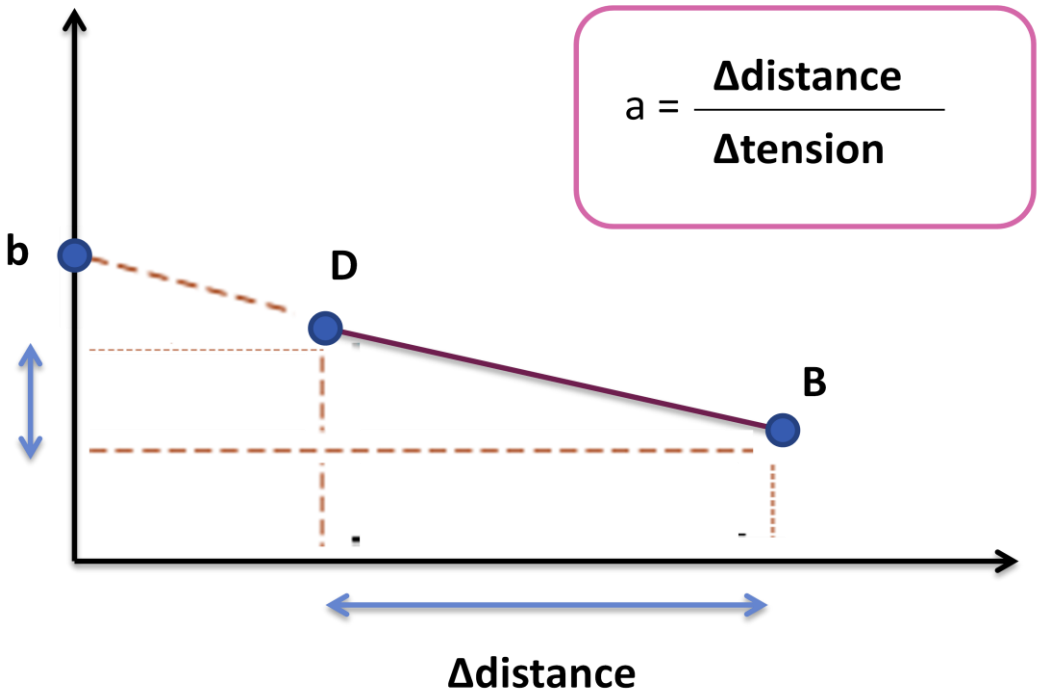
$$\mathbf{Vin = a.distance + b}$$

a est le coefficient de la pente de la droite que l'on calcule en faisant

$$a = \frac{différence\ des\ tensions\ aux\ bornes\ du\ segment}{différence\ des\ distances\ correspondant\ aux\ tensions}$$

b est l'ordonnée du prolongement du segment en zéro.

Par exemple, pour le segment [BCD]



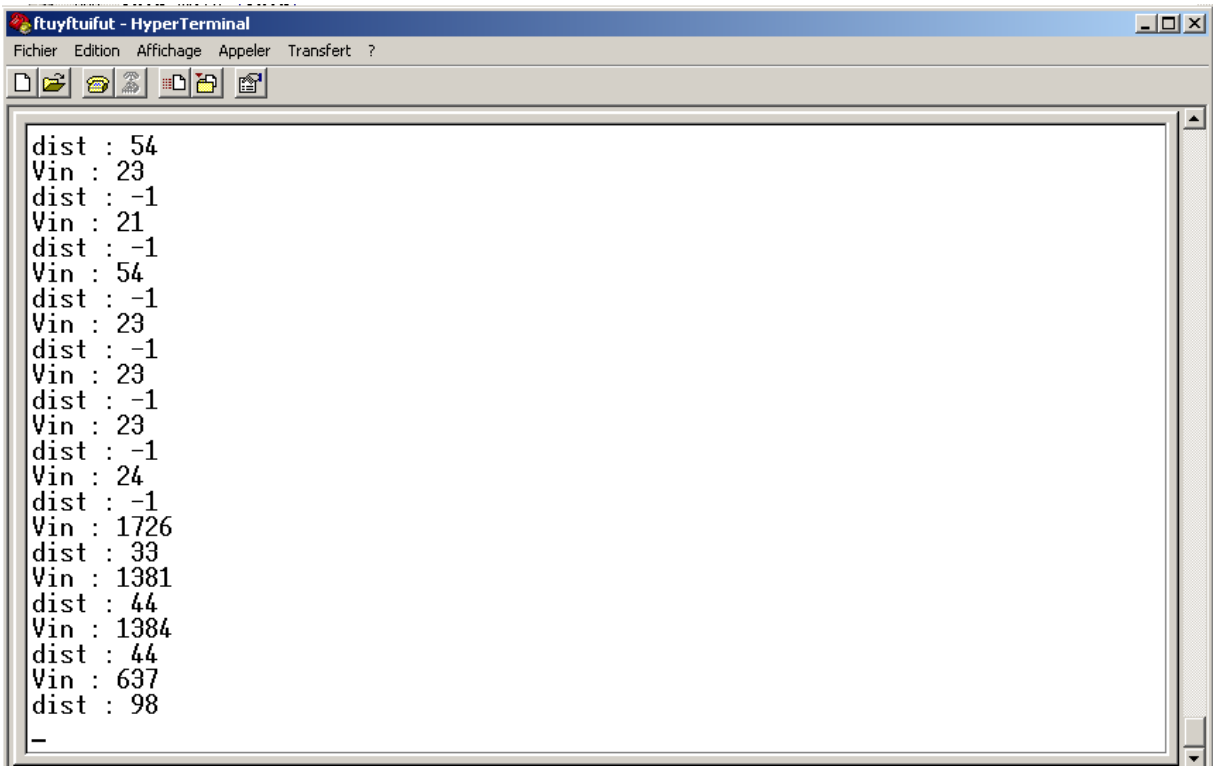
De ces valeurs, on en déduit la distance pour chaque segment :

$$Distance = \frac{b - Vin}{a}$$

4.3.3 Tests logiciels télémètre

Grâce aux fonctions que l'on a réalisées, nous sommes donc en mesure d'afficher les distances entre un objet et le télémètre sur l'HyperTerminal.

Voilà ce que nous obtenons :



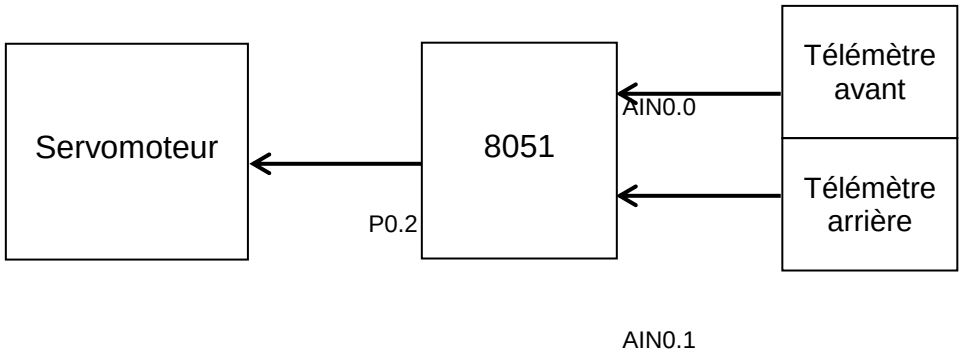
On constate bien que si la distance est inférieure à 25cm, on renvoie un signal d'erreur égale à -1.

Par contre, pour des distances comprises entre [150 cm, 25 cm] on renvoie bien des valeurs adéquates.

4.4 Mise en place de l'ensemble télémètre/servomoteur

(Partie rédigée par Axel Laroche)

Après avoir mis en place chaque partie séparément, nous les avons mises en commun.



Pour initialiser cet ensemble, nous avons utilisé deux fonctions : l'initialisation de l'horloge système et l'initialisation des ports du 8051.

1. Initialisation de l'horloge système : void SYSCLK_Init ()

```
void SYSCLK_Init (void)
{
   int i;                    // Compteur de délai

   OSCXCN = 0x67;                    // Démarrer l'oscillateur externe avec un quartz de
22.1184MHz

   for (i=0; i < 256; i++);          // Attente de 1ms

   while (!(OSCXCN & 0x80));         // Attente que le quartz soit stable

   OSCICN = 0x88;                    // Activation de l'horloge système sur l'oscillateur externe
}
```

La fonction SYSCLK_Init nous permet d'initialiser l'horloge système du microprocesseur. Nous choisissons ici d'utiliser l'oscillateur externe composé d'un quartz de 22.1184 MHz. Il faut ensuite attendre que le quartz soit stabilisé puis on active l'horloge externe.

2. Initialisation des ports : void PORT_Init ()

```
void PORT_Init (void)
{
   // desactivation watchdog
        EA=0;
        WDTCN=0xDE;
        WDTCN=0xAD;
        EA=1;

   P0MDOUT =0xFF; // ports de P0 en mode Push-Pull
   P1MDOUT =0xFF; // ports de P1 en mode Push-Pull

   XBR0    = 0x0C;   // on recupere CEX0 sur le port P0.2 (A-12)
   XBR2    = 0x40;   // activation du crossbar
}
```

Grâce à cette fonction, nous paramétrons les ports du 8051. Nous désactivons tout d'abord le watchdog pour ne pas avoir de problème lors de l'exécution de notre code. Nous paramétrons tous les ports de P0 et P1 en mode Push-Pull pour les utiliser sans problème. Ensuite, les registres XBR0 et XBR2 nous permettent de router la sortie CEX0 du module PCA sur le port P0.2 et d'activer le crossbar.

3. Fonction balayage : void balayage(double angle)

```
void balayage (double angle)
{
        int i=0, j=0;
        int k=0, l=0;

        for (i=0; i<180; j++)
        {
                rotation(i);
                angle_actuel=i;
                i+=angle;
                dist[j][0]=angle_actuel;
                printf("angle_actuel : %d\n",angle_actuel);

                Debut_conversion(0);    // on commence la conversion
                sleep(1);                 // on attend 1s pour que la conversion puisse se faire
                EA=0;                     // on active les interruptions
                dist_av=Distance_telemetre(Vin);          // on convertit la tension mesurée en distance
                dist[j][1]=dist_av;
                printf("dist av : %d\n",dist_av);         // on affiche la distance avant
                EA=1;

                Debut_conversion(1);    // on commence la conversion
                sleep(1);                 // on attend 1s pour que la conversion puisse se faire
                EA=0;                     // on active les interruptions
                dist_ar=Distance_telemetre(Vin); // on convertit la tension mesurée en distance
                dist[j][2]=dist_ar;
```

```
                printf("dist ar : %d\n",dist_ar);              // on affiche la distance arrière
                EA=1;
        }

        for(l=0; l<18; l++)
        {
                printf("  %d    %d    %d  \n", dist[l][0], dist[l][1], dist[l][2]);
        }

        if(dist[0][1] < dist[0][2])   // si dist_av < dist_ar
        {
                dist_min = dist[0][1];        // alors dist_min = dist_av
                avant=1;                                        // c'est le télémètre avant qui a mesuré cette
distance
                arriere=0;
                angle_dist_min = dist[0][0]; // on définit l'angle associé à cette distance min
        }
        else                                                    // si dist_ar < dist_av
        {
                dist_min = dist[0][2];        // alors dist_min = dist_ar
                avant=0;                                        // c'est le télémètre arrière qui a mesuré cette
distance
                arriere=1;
                angle_dist_min = dist[0][0]; // on définit l'angle associé à cette distance min
        }

        for (k=1; k<18; k++)                  // on remplit le tableau pour des angles allant de 0 a 170°
        {
                if(dist[k][1] < dist_min)
                {
                        dist_min = dist[k][1];
                        avant=1;
                        arriere=0;
                        angle_dist_min = dist[k][0];
                }
                if(dist[k][2] < dist_min)
                {
                        dist_min = dist[k][2];
                        avant=0;
                        arriere=1;
                        angle_dist_min = dist[k][0];
                }

        }

        printf("dist_min = %d\n", dist_min);
        if(avant==1 && arriere==0) printf("avant\n");
        if(avant==0 && arriere==1) printf("arriere\n");
        printf("angle_dist_min = %d\n", angle_dist_min);


}
```

La fonction balayage est la fonction globale de notre ensemble servomoteur/télémètre. Nous effectuons ici plusieurs taches : la mesure des distances autour du robot, le stockage de ces distances et le calcul de la plus faible distance autour du robot.

Le servomoteur va effectuer un balayage sur 180° et tous les 10° demander la mesure des distances avant et arrière avec la fonction *Debut_conversion*. Les mesures effectuées sont stockées dans un tableau *dist* contenant l'angle de mesure et les distances avant et arrière. Nous avons également utilisé l'HyperTerminal pour afficher les résultats en temps réel (angle de mesure et distances avant et arrière).

A la fin du balayage, on affiche le tableau *dist* pour visualiser la totalité des mesures effectuées. Puis, nous avons réalisé une fonction de comparaison de toutes les distances mesurées qui permet d'obtenir la plus petite distance mesurée lors du balayage. Nous renvoyons et affichons la plus faible distance avec l'angle correspondant ainsi qu'une indication qui permet de savoir si la distance a été mesurée à l'avant ou à l'arrière du télémètre.

4. Temporisation : void sleep(double duree)

```
void sleep(double sec)
{
        int i=0,j=0;
        while(i<(7000*sec))
        {
        j=0;
        while(j<200)
        {
                j++;
        }
        i++;
        }
}
```

Nous avons utilisé ici une temporisation entre chaque rotation de 10° et mesure de distance. Pour effectuer cette fonction, nous avons simplement utilisé des incrémentations successives pour obtenir un temps correspondant à la durée passée en paramètre. Nous avons été obligés d'inclure une boucle « while » dans une autre car si nous n'utilisions qu'une seule boucle, le compilateur l'ignorait lors de la compilation et la temporisation ne fonctionnait donc pas.

4.5 Tests matériels du télémètre et du servomoteur

(Partie rédigée par Hélène VOLOT)

Avant de réaliser les différentes fonctions, nous avons fait des tests du télémètre à l'aide d'un oscillateur.

Test du télémètre :

On a vérifié pour différentes distances que l'on retrouvait bien les tensions attendues d'après la courbe.

Pour d= 24cm => Vout = 2,7V
Pour d= 51 cm => Vout = 1,16V
Pour d= 110 cm => Vout = 0,58mV

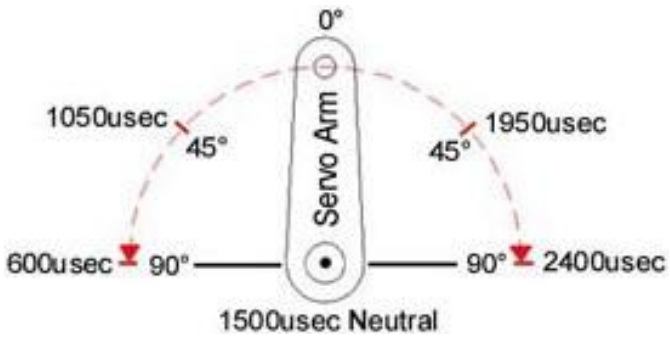
On a vérifié ces valeurs sur la courbe du télémètre et on constatait qu'elles étaient en accord avec la datasheet. Donc le télémètre fonctionne bien.

Test du servomoteur :

On a testé le servomoteur en utilisant une pulse.
On a vérifié son fonctionnement en faisant varier le temps de l'impulsion et en vérifiant que ce temps correspondait au bon angle de rotation.

Pour t=2400us => angle = 90° vers la droite
Pour t=1905us => angle = 45° vers le droite
Pour t=1050us => angle = 45° vers la gauche
Pour t=600us => angle = 90° vers la gauche

On retrouve bien ainsi le schéma de la datasheet :

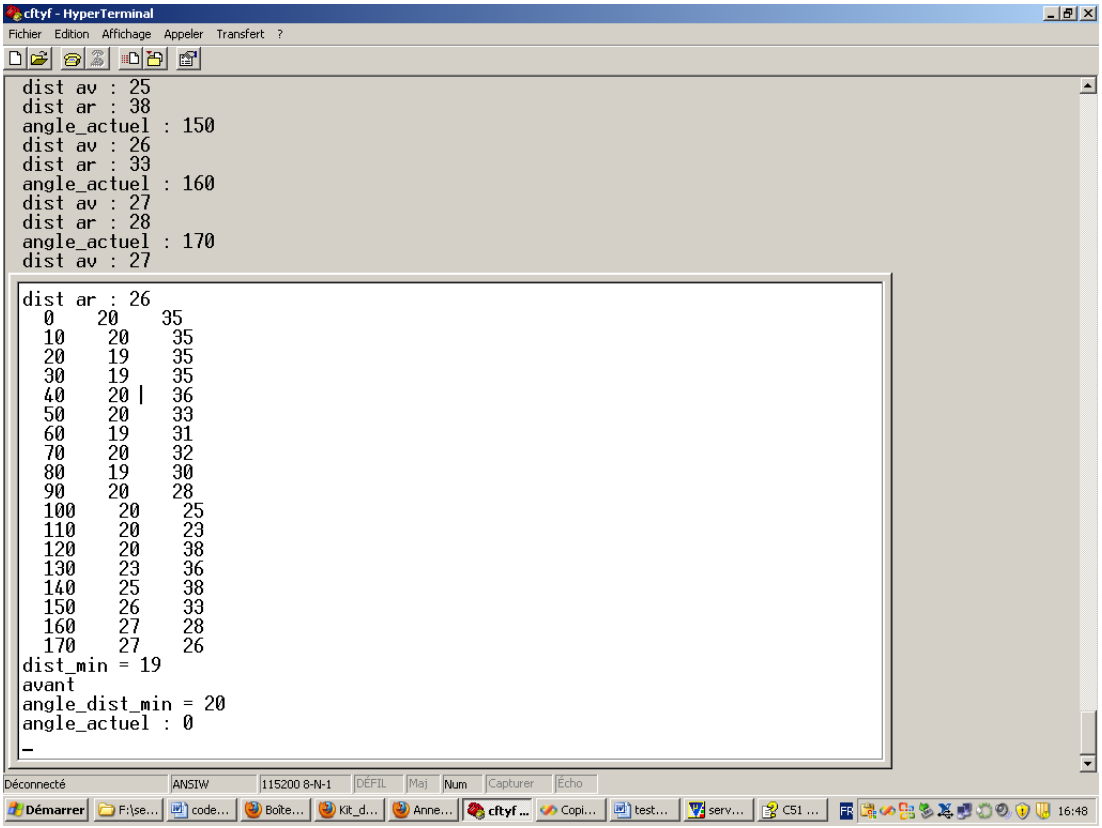


4.6 Tests logiciels de l'ensemble

(Partie rédigée par Hélène VOLOT)

Après avoir mis en commun le télémètre et le servomoteur, on a effectué des tests sur l'HyperTerminal afin de vérifier nos résultats.

Comme notre télémètre avant était cassé, nous avons utilisé le télémètre test .



On obtient bien la distance minimale lors de la rotation et l'angle associé.

Nos tests sont donc concluants.

4.7 Problèmes rencontrés perspectives d'évolution

(Partie rédigée par Axel Laroche et Hélène Volot)

Au départ, nous souhaitions réaliser la fonction de comparaison séparément de la fonction de balayage. L'idéal aurait été une fonction prenant pour paramètre le tableau à analyser. Cependant, nous avons rencontré un problème au niveau de la prise en charge du tableau à deux dimensions et de la méthode à adopter pour le mettre en paramètre. Nous avons finalement intégré cette fonction à la fonction *balayage*.

Nous avons aussi rencontré des problèmes dans la partie télémètre. En effet, au départ on utilisait l'ADC1 car on pensait que la conversion sur 8 bits serait plus simple à réaliser. Cependant, au bout de 5 séances de TP, la fonction ne marchant toujours pas, nous avons pris la décision d'utiliser l'ADC0.
Nous avons aussi rencontré un problème au niveau de la conversion c'est pourquoi nous utilisons des millivolts.
De plus, toutes les tensions que l'on obtenait n'étaient pas stables. On a donc rajouté un filtre RC afin de stabiliser la tension de sortie du télémètre.

5 Déplacement du robot

(Partie réalisée par Romain Ringwald)

5.1 Reformulation du cahier des charges

Le Robot Track Pinger est un engin autonome capable de se diriger par ses propres moyens dans un environnement doté d'obstacles (murs ou Pingers). Il évolue à petite vitesse (quelques km/h) sur un sol plat. Notre RTP possède donc les fonctionnalités d'avancer, de reculer, de tourner à gauche ou à droite afin d'éviter la collision avec un obstacle. Ce déplacement se fera soit en mode manuel (les mouvements du robot sont imposés par une télécommande) soit en mode automatique (les mouvements du robot sont automatiques et évoluent en fonction des données extérieures).

5.2 Solution retenue

Nous avons opté pour le mode manuel. Dans ce mode, notre robot est capable de se mouvoir grâce à une télécommande, dans un premier temps, constituée par des boutons poussoir puis par une télécommande munie d'une carte STM32 (développée dans la partie « mise en place de la télécommande »).

Nous avons fait ce choix car ce mode est réalisable indépendamment de tous les autres modules. En effet, le mode automatique est plus complexe à mettre en place car nous devons tout d'abord mettre en place toutes les fonctions de localisation et de détection pour permettre au robot de se mouvoir par lui même. Il nous ait donc paru préférable de s'axer sur ce mode avant d'aborder le mode automatique.

5.3 Solution technique

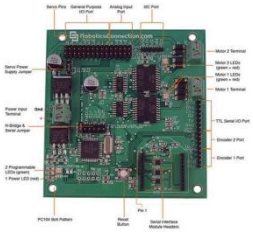
Dans ce paragraphe, nous allons mettre en œuvre une solution pour le déplacement en mode manuel de notre RTP. Ce dernier aura les fonctionnalités d'avancer, de reculer, de tourner à gauche ou à droite selon les ordres reçus par la télécommande.

5.3.1 Matériel à disposition

Le matériel mis à notre disposition afin de répondre à notre problématique est :
Un Kit Robot Track Pinger muni de deux roues motrices, d'une roue de trainage et de deux moteurs alimentés en 7.5 VDC.
Une carte Serializer™ WL intégrée au châssis du Robot Track Pinger.
Un microcontrôleur 8051F020.

Microcontrôleur 8051F020

Carte Serializer™ WL

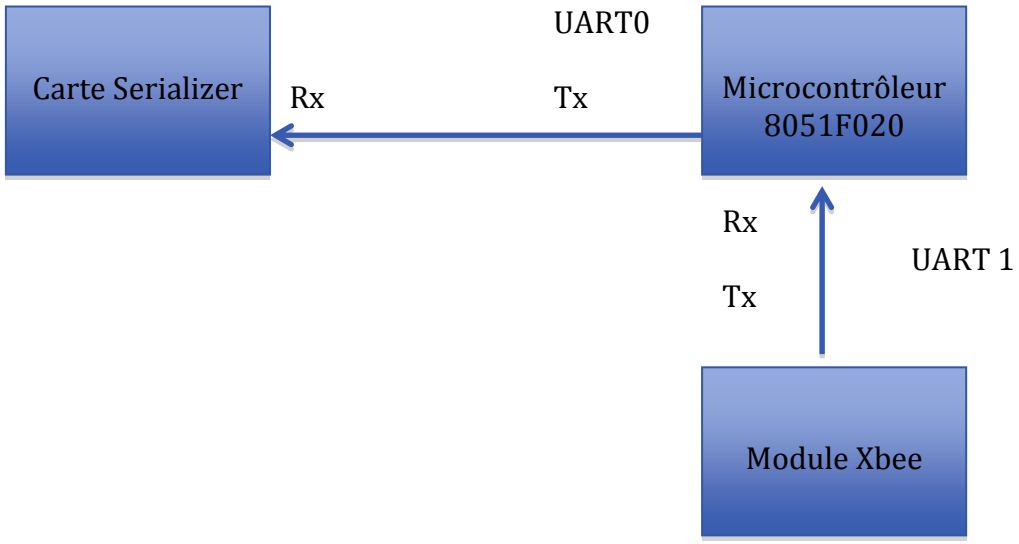


Robot Track Pinger

Le "Serializer™ WL" est un contrôleur universel capable de gérer des périphériques et est doté d'interfaces de puissance capables de prendre en charge le pilotage de différents moteurs via le microcontrôleur. La platine s'interface très simplement au moyen d'un port de communication série (niveau TTL).

5.3.2 Synoptique et branchement de la carte Serializer

Synoptique :



Pour répondre à notre problématique, nous avons eu besoin d'utiliser deux UART du 8051F020 :

- L'UART 0 (19200 bauds), afin d'envoyer les commandes reçues par le 8051F020 à la carte Serializer pour faire bouger le robot.
- L'UART 1, qui reçoit les mouvements imposés par une télécommande (développé dans la partie « mise en place de la télécommande »)

Dans ce paragraphe, nous allons donc nous intéresser à la communication, à travers l'UART 0, entre le 8051F020 et la carte Serializer. Nous utilisons cet UART pour envoyer les commandes implémentées dans le microcontrôleur au Serializer pour contrôler les moteurs et recevoir les acquittements de ces fonctions. En effet, l'utilisation du Serializer est basée sur un protocole simple de communication avec le microcontrôleur : le Serializer est contrôlable par une liaison série, à l'aide de caractères ASCII 8 bits. Par exemple, pour le contrôler avec un ordinateur, il est possible d'utiliser un logiciel de type Hyperterminal. L'utilisation de caractères ASCII permet à l'utilisateur de donner des ordres mais aussi recevoir des informations telles que le nombre de pas courant de la roue codeuse.

Pour ce faire, nous allons utiliser les ports Rx et GND du Serializer que nous branchons respectivement sur les ports Tx (P0.0) et GND du 8051F020. Nous utiliserons à terme le connecteur J24 de la carte d'évaluation du 8051F020.

5.3.3 Elaboration des fonctions

En mode manuel, les commandes implémentées sur le 8051F020 destinées au mouvement envoyées au Serializer sont :

```
void Avancer (void)
        {
          EA=0; // Interdit les interruptions.
          printf ("mogo 1:30 2:30\r"); //Avance le RTP avec une vitesse de 30% de la vitesse maximale.
          EA=1;  // Autorise les interruptions.
        }


void Reculer (void)
        {
          EA=0; // Interdit les interruptions
          printf ("mogo 1:-30 2:-30\r");   //Recule le RTP avec une vitesse de 30% de la vitesse maximale.
          EA=1;  // Autorise les interruptions.
        }


void TournerG (void)
        {
          EA=0; // Interdit les interruptions
          printf ("mogo 1:30 2:-30\r");   //Tourne le RTP à gauche avec une vitesse de 30% de la vitesse maximale.
          EA=1;  // Autorise les interruptions
        }


void TournerD (void)
        {
          EA=0; // Interdit les interruptions
          printf ("mogo 1:30 2:-30\r"); //Tourne le RTP à droite avec une vitesse de 30% de la vitesse maximale.
          EA=1;  // Autorise les interruptions
        }


void Stop (void)
        {
          EA=0; // Interdit les interruptions
          printf ("stop\r"); //Arrête tout mouvement du RTP.
          EA=1;  // Autorise les interruptions
        }
```

L'instruction « mogo » permet la mise en route des deux moteurs du RTP. Elle prend en argument la roue à faire tourner et la vitesse à laquelle elle doit tourner.

Le principe de notre programme est l'utilisation d'une interruption qui scrute l'état des boutons poussoir dès qu'elle rentre en jeu. Cela implique l'utilisation d'un timer. Nous avons choisi le timer 3 pour l'UART 0 (19200 bauds).

En effet, le timer 3, à auto-rechargement, amène une interruption toutes les 35ms. Durant cette dernière, si l'état d'un des boutons poussoir a changé, nous exécutons la fonction correspondante.

La chaine de caractère la plus longue dans notre code compte 18 caractères soit 9,4ms (1 bit est transmis en 52 microsecondes à travers l'UART 0). Afin de réellement prendre en compte l'appui sur un bouton de la télécommande, nous devons laisser un espace temps d'interruption suffisant. Pour cela, on utilise un « countertime » qui nous laisse un espace temps d'environ 350ms.

Voici notre programme :

```
void main (void)
{
  WDTCN = 0xde;              // disable watchdog timer
  WDTCN = 0xad;
  SYSCLK_Init ();            // initialize oscillator
  PORT_Init ();              // initialize crossbar and GPIO
  UART0_Init ();             // initialize UART0
  Timer3_Init ();            // initialize Timer3 to overflow sample rate
  EA = 1;                    // Enable global interrupts
  countertime=10;            // countertime for interrupts 10*35ms=350ms
  while (1)
  {}
}
```

```
void Use (void) interrupt 14
{
   TMR3CN &=0x7F ;           //Registre non adressable bit à bit on fait un masquage pour réinitialiser le timer.
        if(countertime==0) //Laisser le temps d'appuyer sur le bouton poussoir (environ 400ms).
        {
          if(UP==0) //Si l'état du bouton poussoir P3.0 a changé, on appel la fonction Avancer().
          {
                    Avancer();
                    countertime=10;
          }
          else if (DW==0) //Si l'état du bouton poussoir P3.1 a changé, on appel la fonction Reculer().
          {
                    Reculer();
                    countertime=10;
          }
          else if (L==0) //Si l'état du bouton poussoir P3.2 a changé, on appel la fonction TournerG().
          {
                    TournerG();
                    countertime=10;
          }
          else if (R==0) //Si l'état du bouton poussoir P3.3 a changé, on appel la fonction TournerD().
          {
                    TournerD();
                    countertime=10;
          }
          else //Sinon on immobilise le robot.
          {
                    Stop();
```

```
            }
        }
        else countertime--;
}
```

5.4 Tests logiciels et matériels

- **Test de commandes :**
  - **mogo 1 :10 2 :-10** on note bien que les roues tournent a la même vitesse mais de sens opposé.
  - **Stop** : arrêt du fonctionnement du moteur
- **Test logiciel :**
  - Nous avons testé le bon fonctionnement de notre programme grâce à l'allumage d'une LED sur le 8051F020. Nous éteignons la led à chaque début de programme et nous la rallumons si on rentre bien dans la boucle après une interruption.
- **Test Hyperterminal :**
  - Nous affichons grâce à un hyperterminal les commandes envoyées sur l'UART0. Si on appuie sur aucun bouton, aucune commande envoyée et stop.
- **Test Matériel :**
  - La communication se fait entre le 8051F020 où les fonctions sont implémentées et le Serializer par la mise en mouvement du robot par une télécommande

5.5 Difficultés rencontrées

La première difficulté rencontrée a été l'élaboration du programme de base. En effet, nous n'arrivions pas à empêcher l'émission de la commande stop. L'interruption du timer 3 ne captait pas forcément le changement d'état après l'appui sur un bouton poussoir car le temps d'appui était trop long par rapport à l'espace temps entre deux interruptions.

Ce problème a été résolu après la mise en place du countertime qui autorise de scruter l'état du bouton poussoir quand 350 ms se sont écoulées, temps suffisant pour appuyer sur une touche de la télécommande.

5.6 Perspectives d'évolution

Nous avons pu réaliser un mode manuel opérationnel. Il aurait été intéressant de mettre en place un système d'acquittement des fonctions. De plus, les difficultés rencontrées ne nous ont pas permis de mettre en place le mode automatique avec la commande « digo ».

6 Emission et réception sonore

(Partie rédigée par Florian Girodet)

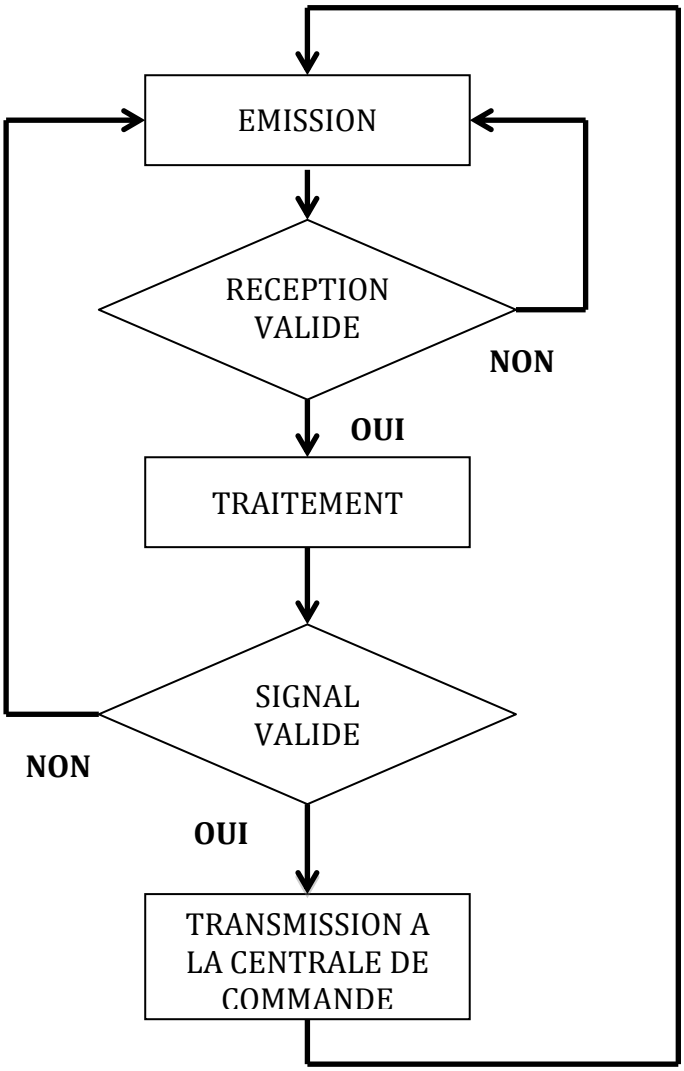
6.1 Communication acoustique : la problématique

Le but de cette étude était de mettre en place et de réaliser un système acoustique embarqué sur le Robot Track Pingers et capable d'identifier la présence d'éventuel Pingers situés à l'intérieur d'obstacles.

Pour ceci on a distingué deux parties à traiter :

- L'**Interrogation** : émission par le système d'un signal sonore bien précis en fréquence et en durée afin qu'il puisse déclencher un transpondeur acoustique
- La **Réponse** : réception par le système d'un signal sonore émis par un transpondeur acoustique situé sur un obstacle, traitement de ce signal et transmission des informations au microcontrôleur

Le fonctionnement du système peut être représenté par l'organigramme de fonctionnement ci-dessous :



Principe :

1) Emission d'un signal sonore par le haut-parleur du robot

2) Réception ou non du signal émis par l'obstacle en réponse au signal sonore du robot

3) Traitement du signal afin de le caractériser, on récupère sa fréquence ainsi que sa durée

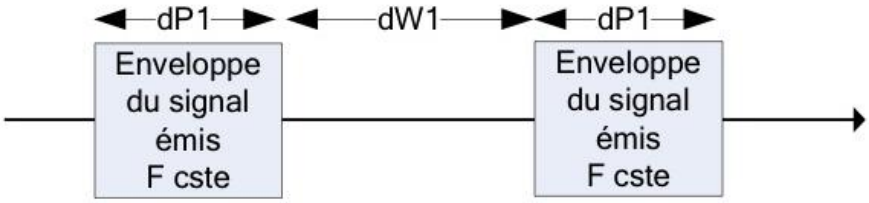
4) Si le signal traité correspond à un signal dont les caractéristiques sont valides alors on transmet ses informations à la centrale de commande, puis on revient à la phase d'émission afin de pouvoir interroger un nouvel obstacle

5) Dans le cas où le signal n'est pas valide on revient directement à la phase d'émission.

6.2 Communication acoustique : émission sonore

6.2.1 Reformulation

La forme du signal à émettre est la suivante, avec dP1, dW1 et F, respectivement la durée d'émission, de silence et la fréquence du fondamental :



Les caractéristiques du signal doivent respecter certaines conditions :

➔ dP1 et dW1, durées réglables par le microcontrôleur variant de 100ms à 1s par pas de 50ms

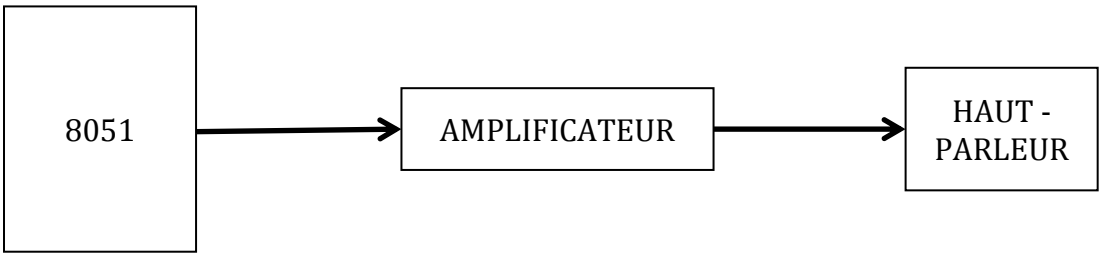
➔ La durée totale d'émission du signal par le robot sera d'au moins 3 s

➔ F, fréquence variable pouvant prendre les valeurs :

	Fréquence en Hertz		
	Octave 3	Octave 4	Octave 5
DO	**261,6**	**523,2**	**1046,4**
RE	293,7	587,4	1174,8
MI	**329,7**	**659,4**	**1318,8**
FA	349,2	698,4	1396,8
SOL	**392**	**784**	**1568**
LA	440	880	1760
SI	**493,9**	**987,8**	**1975,6**

6.2.2 Solution Hardware

Synoptique de la solution retenue :



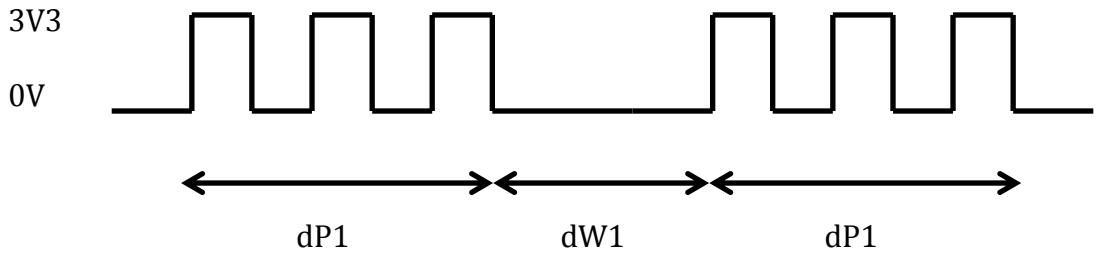
Matériel utilisé :

- Microcontrôleur 8051 (commun au servomoteur)
- Haut-parleur électromagnétique de 8Ω - 50 mm de diamètre
- Connecteurs HE14/4 contacts, pré-câblés sur le Haut-parleur
- Amplificateurs audio LM4871

Choix :

➔ On choisit d'utiliser le microcontrôleur 8051 comme générateur du signal à émettre afin qu'il délivre des signaux dont les caractéristiques répondent au cahier des charges (voir Solution Software **a)**)

Les signaux envoyés sont sous la forme :



Avec dP1 = dW1 = 500 ms.

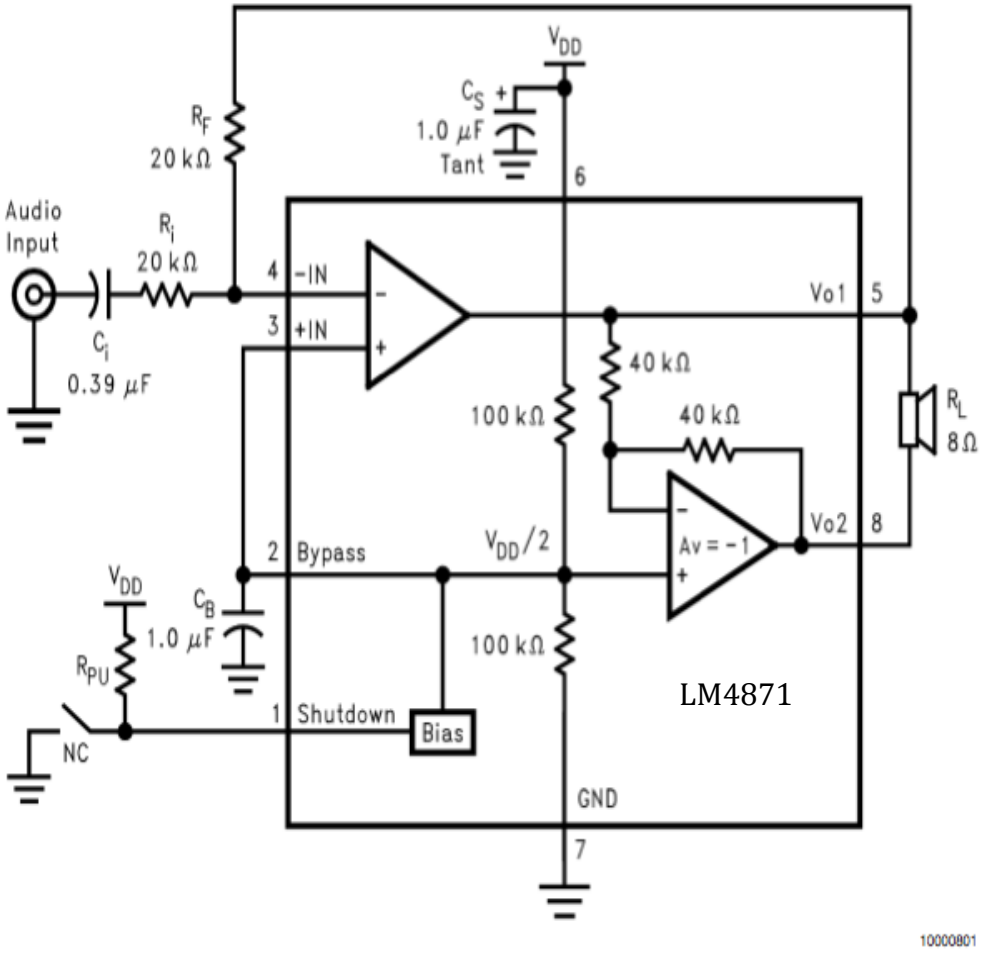
Les signaux délivrés par le 8051 sont compris entre 0 et 3,3V (tension d'alimentation). Ainsi la puissance mise en jeu à la sortie du 8051 est **P = U x I**.

➔ Afin de transformer ce signal électrique en signal acoustique on met en œuvre un Haut-Parleur électromagnétique de 8Ω - 50mm de diamètre dont les caractéristiques sont :

Impédance	8 Ohm
Puissance max	2 W
Sensibilité	88dB

On voit d'après ces caractéristiques que pour exploiter au mieux toutes les capacités du Haut-parleur, il est primordial d'amplifier le signal issu du microcontrôleur.

➔ On choisira donc également de placer un amplificateur audio LM4871 (représenté ci-dessous) entre le 8051 et le Haut-parleur :



Celui-ci est configuré dans son application typique avec une résistance de Pull-Up **RPU** de 5,5 MOhm. On connecte le Haut-parleur en amont de ce module. Il est représenté par l'impédance **RL** de 8 Ohm.

6.2.3 Solution Software

Afin de répondre à notre problématique, nous avons besoin de construire des temps d'émission (Tdp1) et de silence (Tdw1) configurables par l'utilisateur. Pour cela, nous avons décidé de fabriquer une horloge temps réelle pour obtenir le temps et fixer ces durées. Cette RTC a été réalisée grâce au Timer 2 qui produit une interruption toutes les 5 ms. Une fois ces durées acquises, nous comparons dans void emission () l'horloge temps réelle avec les temps d'émission et de silence souhaités pour gérer ces durées. Le DAC du microcontrôleur qui est utilisé pour la génération de son est le DAC1. Pour cette génération sonore, on utilisera un Timer 4 qui sera réservé à l'échantillonnage du sinus. L'interruption du Timer 4 vient mettre à jour le DAC et calcule la prochaine valeur de sortie en se basant sur la forme du signal attendue. Nous avons testé le bon fonctionnement de notre programme par la génération d'un signal réglable par la forme, la fréquence et l'amplitude.

```
void temps(void)
{
   Tdp1=horloge_reel+dp1;
   Tdw1=Tdp1+dw1;
   Tfin=Tdw1+dp1;
}
void emission(void)
{
   int emission=1;
      if(horloge_reel<Tdp1)
      {
         sortie=1; // sinus
      }
}

void RTC(void)
{
   centieme=centieme+5 //appele toutes les 0.0125ms, interruption toute les 5ms
   if(centieme=1000)
   {
      seconde=seconde+1;
      centieme=0;
      if(seconde=60)
      {
         minute=minute+1;
         seconde=0;
      }
   }
   horloge_reel= centieme + seconde*1000 + minute*60000; //valeur en ms.
}
```

6.2.4 Résultats

Nous avons commencé par câbler le module sur une plaquette LAB, puis grâce à des visualisations sur l'oscilloscope nous l'avons validé matériellement.
Cette version permet d'obtenir un signal dont la fréquence est celle souhaitée.
Notre système d'émission sonore est conforme au cahier des charges dans le sens où il est capable d'émettre des signaux sonores dont les fréquences sont comprises entre 261,6 Hz et 1975,6 Hz, avec des durées d'émission configurables.

6.2.5 Difficultés rencontrées

Les parties Hardware et Software ont été traitées assez rapidement. Nous n'avons pas eu de difficultés particulières pour la conception et la mise en œuvre du système dans son ensemble.
Le système a donc été opérationnel rapidement.

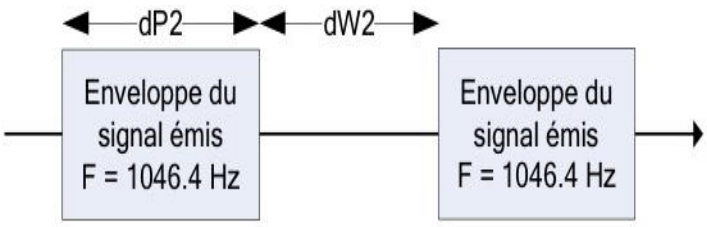
6.2.6 Perspectives d'amélioration

L'une des fonctionnalités intéressantes de l'amplificateur LM4871 est le « Shutdown » (broche n°1) qui permet de réduire significativement la consommation du système. En effet lorsque le module d'émission n'est pas utilisé il est possible de passer l'amplificateur en mode « IDLE ». Pour ce faire il suffit d'appliquer un niveau logique haut sur la pin SHUTDOWN (on utiliserait une des sorties du 8051 pour contrôler cette fonction).

6.3 Communication acoustique : réception sonore

6.3.1 Reformulation

La forme du signal sonore envoyé par le transpondeur acoustique est la suivante :

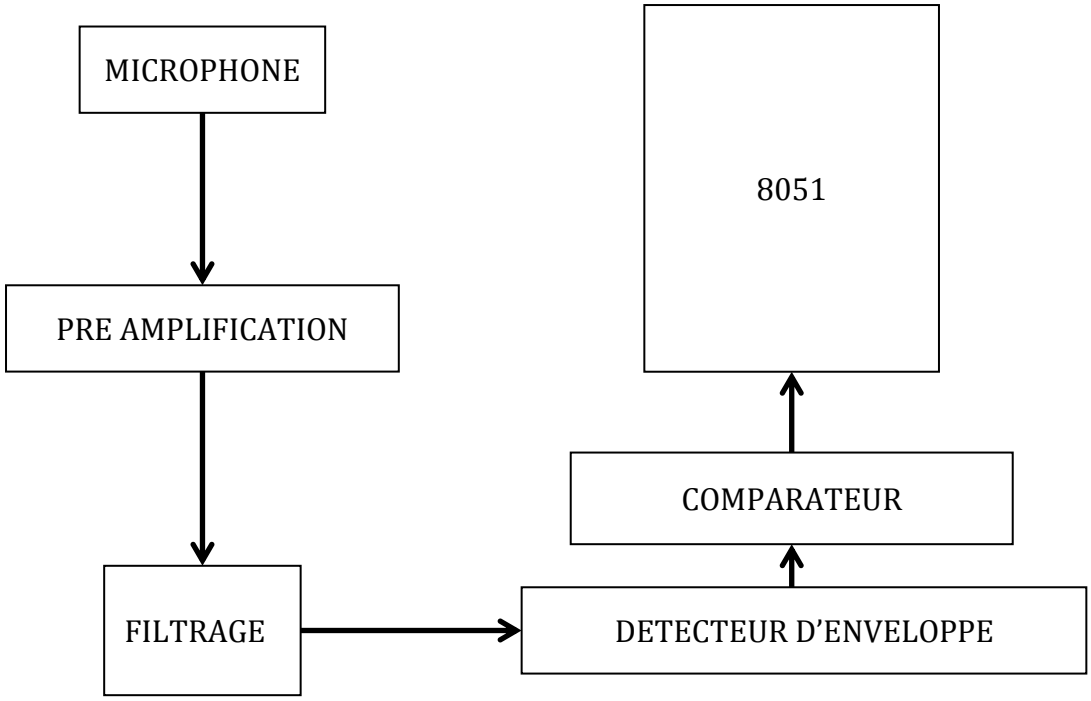


Les caractéristiques du signal respectent les conditions suivantes :

- ➔ dP2 et dW2, durées variables de 100ms à 1s par pas de 100ms, détectables par le système afin de les communiquer à la centrale de commande
- ➔ la durée d'émission du signal par le pinger sera d'au moins 8s
- ➔ F = 1046,4 Hz (DO de l'octave 5)
- ➔ le pinger émettra si le transpondeur capte le signal adéquat émis par le robot pendant au moins 2s

6.3.2 Solution Hardware

Synoptique de la solution retenue :

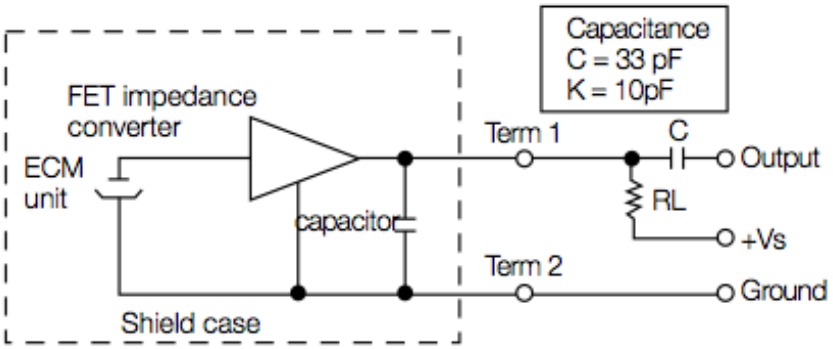


Matériel utilisé :

- Microcontrôleur 8051 (commun au servomoteur)
- Microphone à capsule électret de référence Panasonic WM-62A
- Connecteurs HE14/4 contacts, pré-câblé sur le Microphone
- Amplificateur opérationnel MCP602

Analyse :

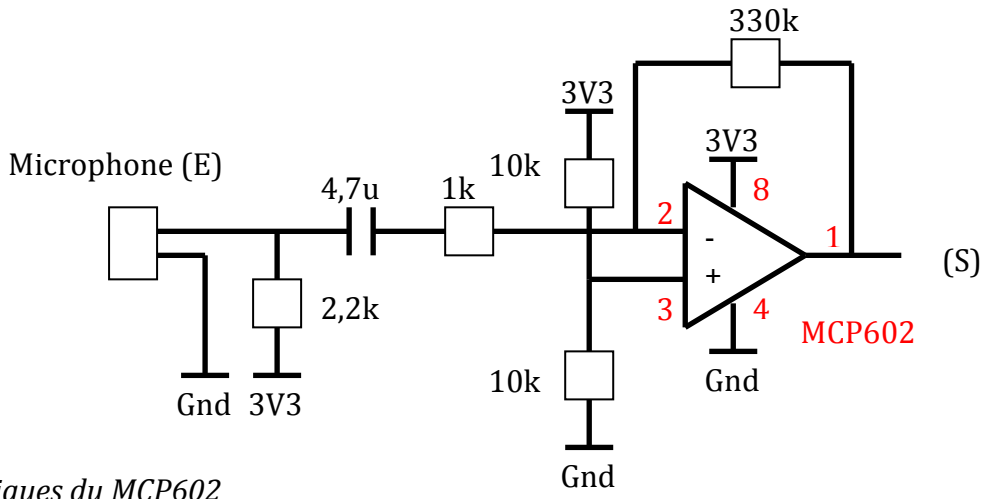
Afin de réceptionner le son émis par le transpondeur acoustique d'un Pinger, on choisit d'utiliser un microphone Panasonic WM-62A à capsule électret dont le schéma interne électrique est le suivant :



Ce microphone permet bien de récupérer un signal électrique mais non exploitable du fait de sa trop faible amplitude de sortie (typiquement quelques mV). Ainsi il a fallu trouver un moyen d'amplifier le signal significativement.

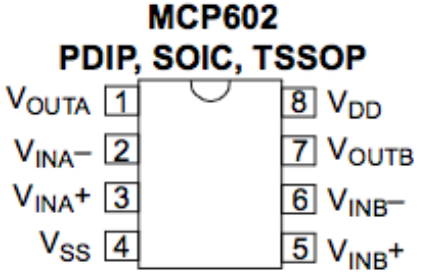
On a donc choisi de mettre en œuvre un étage d'amplification, notamment à l'aide d'un amplificateur opérationnel MCP602, dont l'ossature électrique est la suivante :

Schéma électrique



Caractéristiques du MCP602

Alimentation	2,7 à 5,5V
Produit Gain-Bande	2,8 MHz
Courant de repos	230 uA

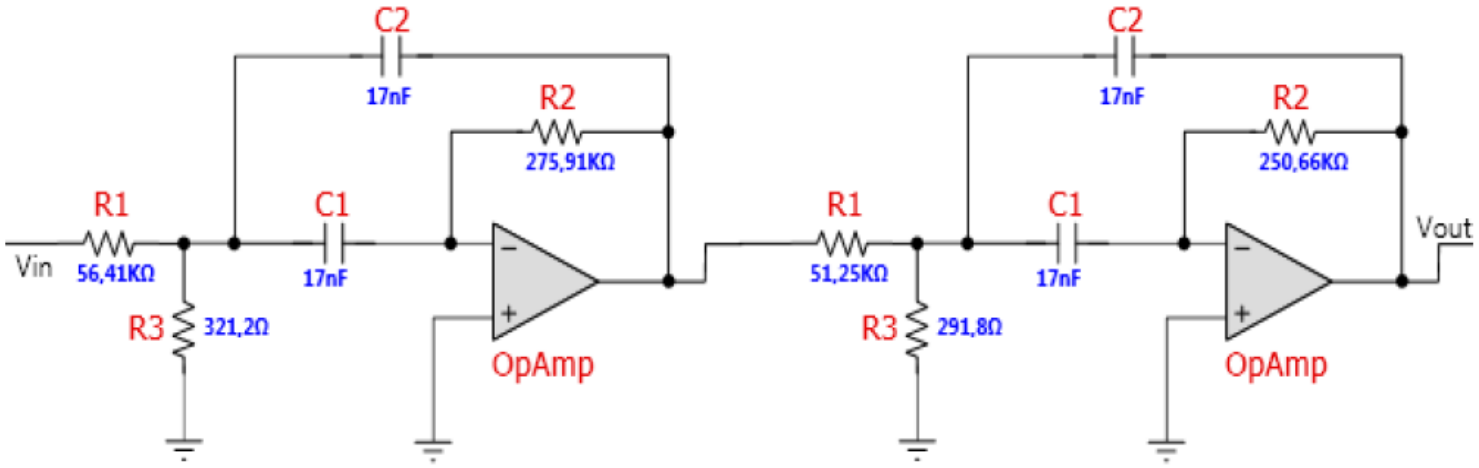


On a choisit R1 = 330 kOhm et R2 = 1kOhm afin d'obtenir un facteur d'amplification de 330. En effet, d'après les caractéristiques du MCP602 (exposé ci-dessus) un tel facteur est suffisant pour obtenir un signal exploitable. Après quelques tests nous obtenons un facteur de 270, ce qui est suffisant.

De plus on doit prendre en compte la nature bruyante de l'environnement dans lequel le robot évoluera, c'est-à-dire un milieu source de parasites. On procédera donc à un filtrage du signal afin de réduire l'impact négatif des bruits parasites.

Cet étage permet d'éliminer toutes les fréquences autres que celle désirée (1046,4 Hz). En réalité une certaine bande de fréquence Δf est tout de même à considérer car un filtrage ne peut être absolu. En effet la tension de sortie voit son amplitude maximale pour une fréquence d'entrée au centre de la bande passante.

Nous avons utilisé un filtre actif passe-bande d'ordre 4 ayant une structure de Sallen&Key (celles étudiées en TP FAC).



Son schéma électrique est le suivant : (partie rédigée par Romain Ringwald)

La valeur des différents composants et l'étude fréquentielle a été obtenue grâce au logiciel FilterPro (téléchargeable sur le site de Microchip) à partir du gain H, du facteur de qualité Q, et de la fréquence de coupure Fc connus.

Fc = 1046,4Hz	Q = Fc / Δf = 10,46
Δf = 100 Hz	H = 2

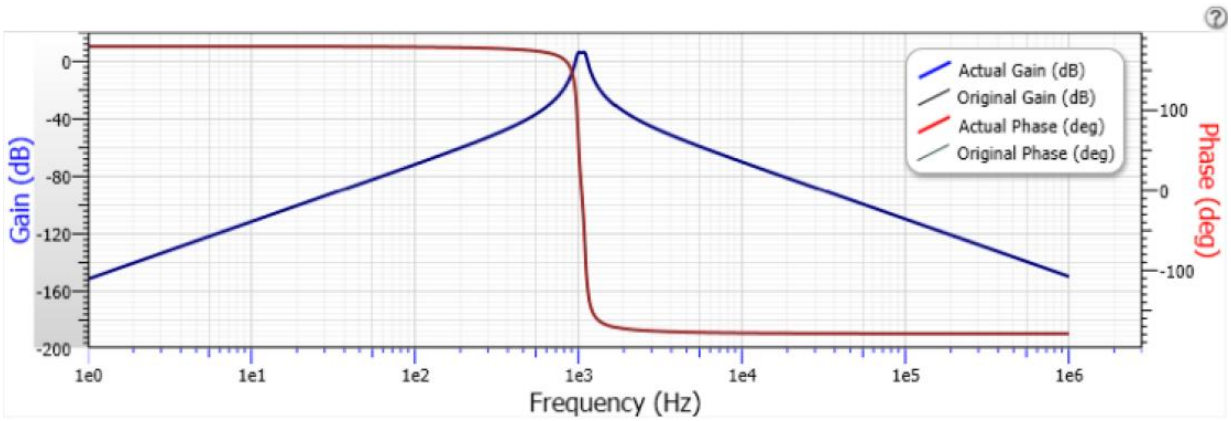
On obtient :

FilterPro Design Report
Bill of Materials

Design Name: Bandpass, Multiple Feedback, Chebyshev 0,5 dB **Part:** Ideal Opamp **Order:** 4 **Stages:** 2
Gain: 1,999 V/V (6,01797637843522 dB) **Allowable PassBand Ripple:** 1 dB **Center Frequency:** 1,0464 kHz
Corner Frequency Attenuation: 6,018 dB **Passband Bandwidth:** 100 Hz

Element ID	Quantity	Part Number	Value	Tolerance	Description	Manufacturer
R1 (Stage 1)	1	Standard	56,41KΩ	Exact: 0%	Resistor	
R2 (Stage 1)	1	Standard	275,91KΩ	Exact: 0%	Resistor	
R3 (Stage 1)	1	Standard	321,2Ω	Exact: 0%	Resistor	
C1 (Stage 1)	1	Standard	17nF	Exact: 0%	Capacitor	
C2 (Stage 1)	1	Standard	17nF	Exact: 0%	Capacitor	
OpAmp (Stage 1)	1	Standard			Ideal OpAmp	
R1 (Stage 2)	1	Standard	51,25KΩ	Exact: 0%	Resistor	
R2 (Stage 2)	1	Standard	250,66KΩ	Exact: 0%	Resistor	
R3 (Stage 2)	1	Standard	291,8Ω	Exact: 0%	Resistor	
C1 (Stage 2)	1	Standard	17nF	Exact: 0%	Capacitor	
C2 (Stage 2)	1	Standard	17nF	Exact: 0%	Capacitor	
OpAmp (Stage 2)	1	Standard			Ideal OpAmp	

Etude fréquentielle (filtre passe-bande dans son ensemble) :



Une fois le montage câblé nous nous sommes aperçu que deux fréquences de coupure sont filtrées, ainsi on a du modifier la valeur des condensateurs C1 et C2 afin de ne filtrer plus qu'une fréquence.

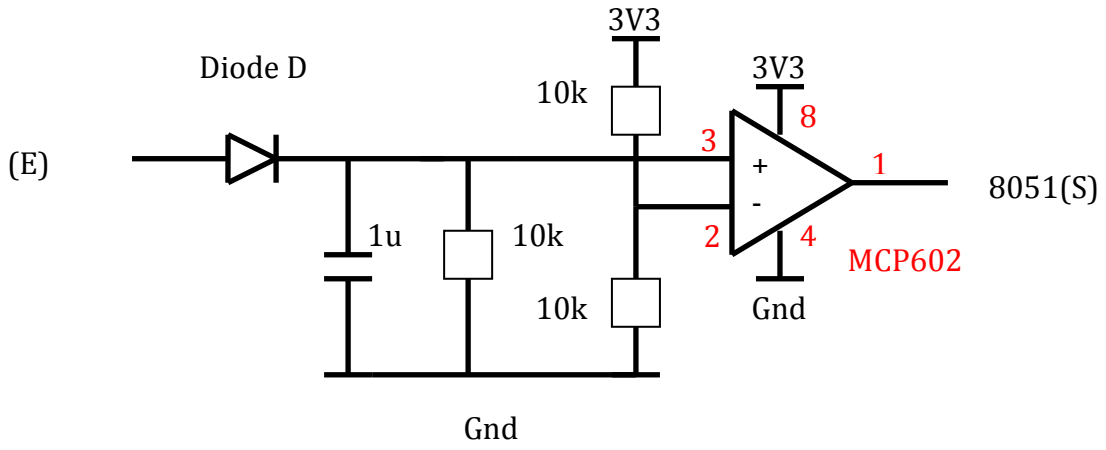
On choisit d'augmenter uniquement la valeur de C1 à 30 nF après quelques tests. Notre filtre passe-bande est maintenant centré sur une fréquence de 1040 Hz avec une bande passante Δf de 120 Hz. On considère que les caractéristiques du filtre satisfont les conditions stipulées dans le cahier des charges.

Afin que le microcontrôleur puisse mesurer la largeur d'impulsion dP2 pendant lequel le système de réception acquiert un signal acoustique (temps d'émission à la fréquence 1046,4 Hz), il faut que le signal soit de type créneau. C'est pour ceci que l'on a choisi de mettre en œuvre un détecteur de crête suivi d'un comparateur, en amont du filtre précédent.

Le comparateur transforme une information analogique en une donnée logique :

- présence du signal souhaitée => tension de sortie à 3V3
- absence du signal souhaitée => tension de sortie à 0V

Le schéma de câblage électrique est le suivant :



6.3.3 Solution Software

Notre signal généré permet de détecter la présence ou non du signal sonore. Il est nécessaire d'étudier le signal afin de récupérer les durées dP2 et dW2. Par manque de temps imparti, nous n'avons pas pu implémenter la partie logicielle concernant la réception. Cependant, de manière théorique, dès qu'on détecte la présence d'un signal, on déclenche un compteur qui va s'incrémenter selon un Timer. Lorsque le signal n'est plus présent, on arrête le compteur et on obtient le temps correspondant à l'émission.

6.3.4 Résultats

Au niveau Hardware, les choix que nous avons effectué ont permis à notre système d'être capable de détecter un signal sonore à la fréquence voulue (Fc = 1046,4 Hz). On a visualisé l'évolution du signal grâce à un oscilloscope, tout au long de la chaîne de traitement.

6.3.5 Difficultés rencontrées

La chaîne de traitement a vite été pensée mais nous avons perdu beaucoup de temps et nous avons eu beaucoup de difficulté à trouver un filtre passe-bande fonctionnel sans l'aide de FilterPro.
Pour la conception du filtre passe bande, le calcul des valeurs des composants par FilterPro n'étaient pas optimal au vu des différences obtenues entre les attentes que nous avions au niveau des caractéristiques du filtre et les résultats à l'oscilloscope.
Nous avons donc dû modifier légèrement certaines valeurs de composants.

La tension de sortie du microphone était trop faible, ainsi nous avons pallié à ce problème en mettant en œuvre un fort gain en amont du filtre.

Lors des essais, les tests ont été concluants rapidement.

Une fois toute la chaîne opérationnelle, nous avons décidé de souder nos modules sur carte LAB, en commençant par le filtre. Il ne fonctionnait plus, ainsi nous avons tout recâblé sur plaquette LAB en abandonnant l'idée de soudures.

De plus nous n'avons malheureusement pas réussis à récupérer le temps d'émission d'un signal émis par un pinger (réceptionné par le robot). En effet, même si au niveau matériel la solution prend en compte cette mesure de temps (comparateur), nous n'avons pas programmé l'interruption sur l'état logique Haut de la broche du microcontrôleur considérée (en sortie du comparateur), capable de déclencher un timer.

6.3.6 Perspectives d'évolution

La sortie du comparateur est envoyée sur une des entrés d'interruption du 8051, ainsi avec une programmation logicielle d'un timer il serait possible de récupérer les durées dP2 et dW2. De plus si cette partie fonctionnait on pourrait utiliser un comparateur à cycle d'hystérésis afin d'éviter les effets de rebonds occasionnés par un simple comparateur.

Il serait nécessaire également de procéder à une soudure propre et fonctionnelle afin d'embarqué notre système sur le Robot Track Pingers.

6.4 Consommation électrique

(Partie rédigée par Florian Girodet)

6.4.1 Reformulation

Pendant la durée de son évolution, le Robot doit être capable de transmettre à l'utilisateur sa consommation d'énergie et ceci en temps réel. Ainsi nous devions équiper notre Robot d'un dispositif électrique capable de mesurer à tout moment la quantité d'énergie utilisée par la globalité du système « Robot Track Pingers ».

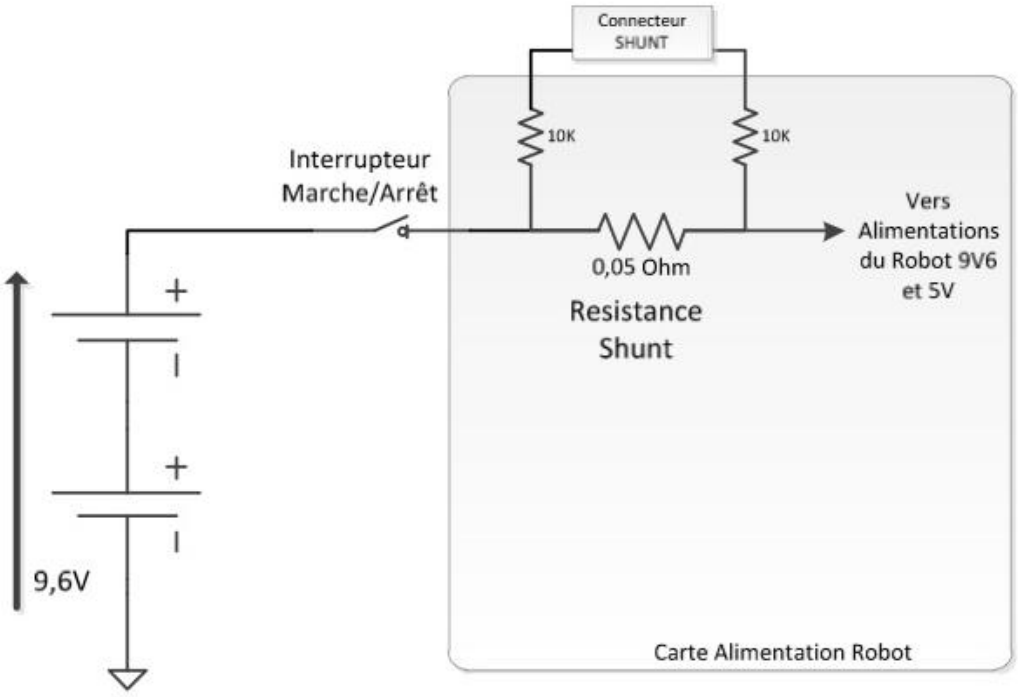
En mettant en place ce dispositif nous aurions pu réduire au mieux sa consommation électrique mais également quantifier l'autonomie du Robot en termes d'énergie.

Le Robot doit être autonome et donc il doit disposer d'une alimentation qui lui est propre, soit deux sources d'alimentation :

- une batterie de 9V6
- une alimentation de 5V régulée et limitée à 1,5A qui alimente les cartes 8051F020, les télémètres et le servomoteur

De plus une résistance « Shunt » de 50 mOhms est placée en série dans l'alimentation. C'est grâce notamment à cette dernière que nous pouvions mesurer la consommation électrique du Robot.

Schéma de l'alimentation :

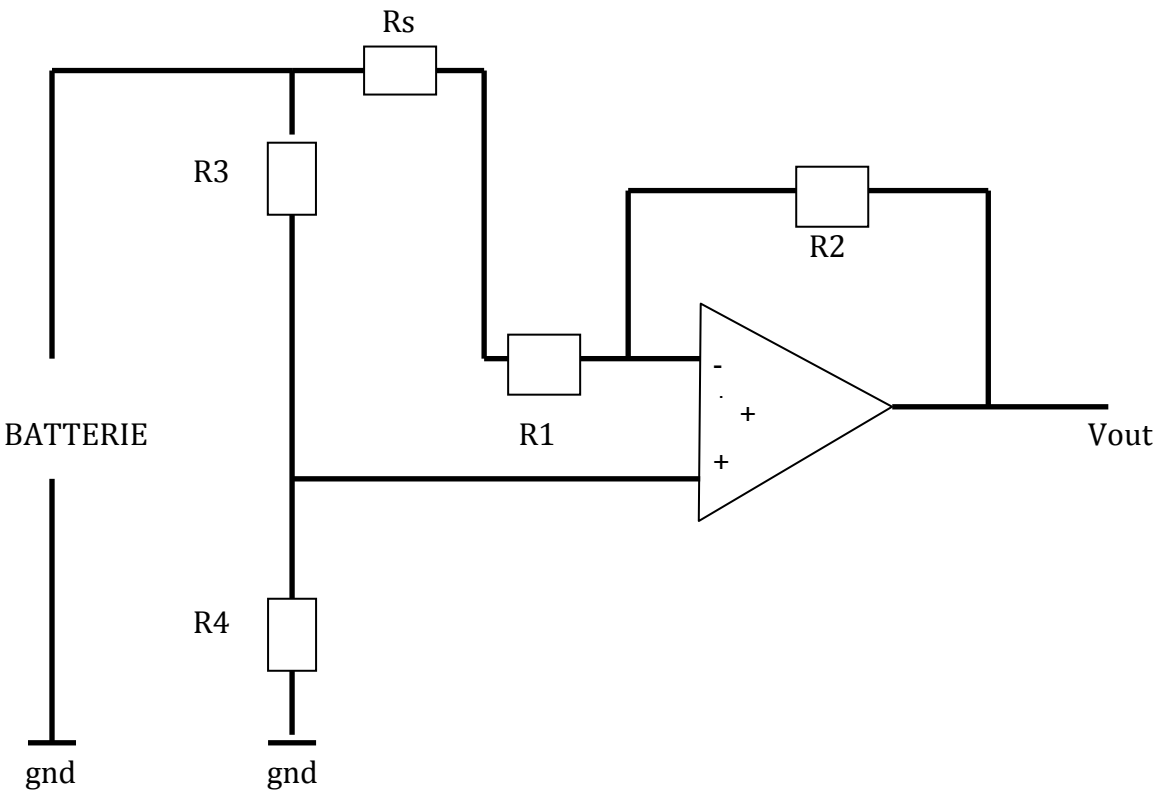


6.4.2 Solution envisagée précédemment

Pour calculer la puissance consommée par le Robot, le plus simple était de mesurer la puissance que débite la batterie via la résistance « Shunt »
Nous devons mesurer les courants afin de pouvoir calculer la puissance consommé grâce à la formule P=Rsh.I^2.

La résistance « Shunt » est de très faible impédance, ainsi on pouvait mesurer une différence de potentiel à ses bornes. On pourra utiliser un Amplificateur Opérationnel Différentiel (AOD) pour calculer celle-ci. Nous pouvions donc connaître la puissance débitée par la source d'alimentation si on impose une relation proportionnelle entre le courant de « Shunt » et la tension de sortie de l'AOD, grâce à plusieurs résistances correctement sélectionnées.

On réalise le montage suivant :



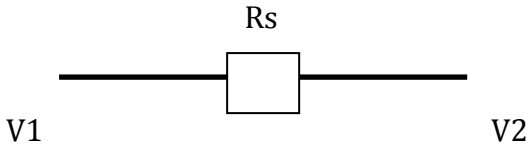
Afin de mesurer la consommation électrique en temps réel et de transmettre les informations correspondantes à l'utilisateur, nous devrons convertir la tension de sortie de l'AOD en grandeur numérique. En effet le Microprocesseur peut traiter uniquement des valeurs numériques et un traitement logiciel est nécessaire pour sommer la consommation électrique instantanée. Nous utiliserons donc le CAN du Microprocesseur 8051.

Les résistances de l'amplificateur opérationnel différentiel doivent également être choisies afin de ne pas perturber le circuit principal. Elles doivent amplifier la tension mesurée mais aussi limiter la tension en sorti de l'amplificateur pour protéger le CAN qui n'accepte que des tensions qui varient de 0 à 2,2V.

Il serait nécessaire de dimensionner correctement les composants du dispositif de mesure de la consommation d'énergie ainsi que la base de temps de conversion du CAN. En effet si la consommation varie dans le temps alors il sera peut-être nécessaire d'avoir une fréquence d'échantillonnage assez élevée. Les composants devront également résister aux pics de tension ou de courant.

Dimensionnement des composants :

`Différence de potentiel aux bornes de la résistance « Shunt » :



Il faut que **Vout = (R2/R1).(V2-V1).**

(V2-V1) est la chute de tension aux bornes de la « Shunt », celle-ci est au maximum de 500 mV.

Pour ne pas dégrader le CAN, il faut que **R2/R1 < Voutm/(V2-V1)** , avec Voutm=2,5V.

Ainsi on choisit R2/R1=4,5.

Nous devrons choisir des résistances avec une impédance élevée pour ne pas perturber le circuit.

On choisirait : R1=R3=1MOhm et R2=R4=4,7MOhms.

7 Mise en place de la télécommande

(Partie rédigée par Selim Boudjaoui)

7.1 Problématique

Le robot Track Pinger est un engin capable de se diriger par ses propres moyens, et évolue à petite vitesse sur un sol plat. Ces fonctionnalités évoluent cependant en fonction des objectifs fixés lors de la pré-étude.

Notre robot doit atteindre la deuxième phase de conception, c'est-à-dire qu'il doit être capable de se déplacer et de détecter les obstacles en relative autonomie. L'utilisateur pourra donc contrôler le robot à partir d'une centrale de commande (liaison sans fil) et accéder via celle-ci aux informations de trajectoire et d'obstacles sur sa route.

7.2 Les ressources matérielles

Nous utilisons ainsi une carte STM32 dotée d'un écran tactile, qui fera office de centrale de commande et permettra à l'utilisateur de visualiser les données précédemment citées.

Les modules radio Xbee utilisés sont automatiquement configurés pour une transmission RF des données et permettent à l'utilisateur de contrôler le robot à distance.

Enfin, 2 microcontrôleurs 8051 seront programmés pour remplir les différentes fonctions du robot.

Caractéristiques de la STM32 :

La carte microcontrôleur STM32 est dotée d'un écran tactile LCD sur lequel le robot affichera l'ensemble des données qu'il enregistre durant son déplacement.

L'utilisateur aura ainsi accès à la consommation en temps réel du robot, des informations sur sa trajectoire, ainsi que la position des différents obstacles présents sur son parcours.

De plus, cette carte possède plusieurs liaisons de types différents permettant de l'interfacer avec plusieurs périphériques externes (I2S-I2C-USART-SPI-CAN), 64KB de mémoire interne SRAM et 256KB de mémoire FLASH.

Nous utilisons ainsi la liaison UART de la carte STM32 pour communiquer avec le module radio qui transmettra les données au microcontrôleur 8051.

Caractéristiques des modules radio Xbee :

Certifiés à la norme Zigbee, les modules radios Xbee sont capables d'assurer la transmission bidirectionnelle RF des signaux de façon transparente.

Véritable modem radio sub-miniature « low cost » en bande 2,4Ghz, ce modèle dispose d'une antenne « chip » intégrée qui libère l'espace autour du module et permet d'obtenir une portée maximale de l'ordre de 30m en intérieur et jusqu'à 100m en extérieur.

Les microcontrôleurs 8051 :

Nous avons choisi d'utiliser 2 microcontrôleurs 8051 communiquant via une liaison SPI, et permettant de remplir les différentes fonctions du 8051.

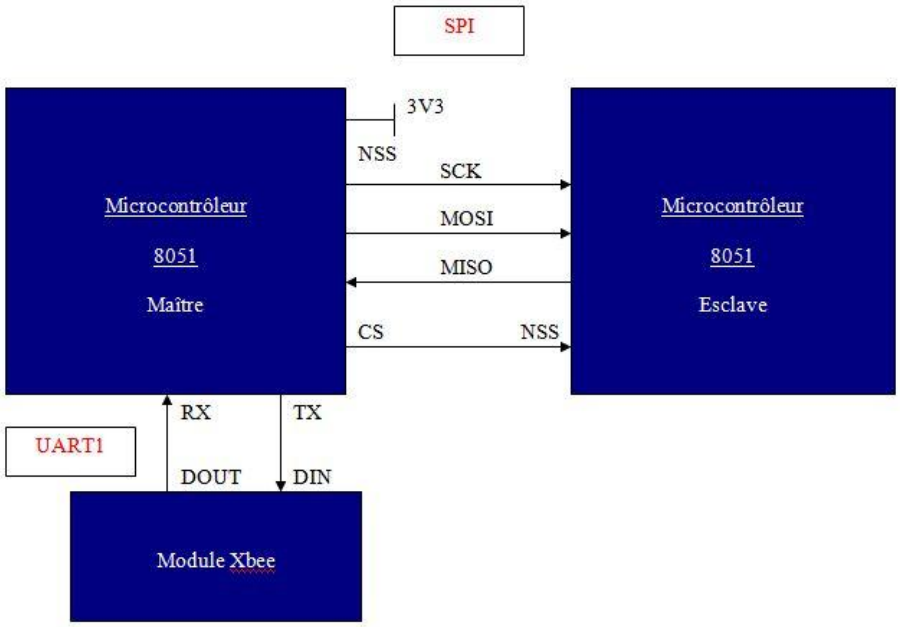
Le maître remplit la fonction de commande du robot grâce à la réception des signaux provenant de la carte STM32 (module Xbee relié à l'UART1), leur interprétation et la transmission des directives de déplacement au Serializer (via l'UART0).

L'esclave accueille le système de détection des obstacles (infrarouge ou sonore) et permet ainsi de calculer les distances correspondantes.

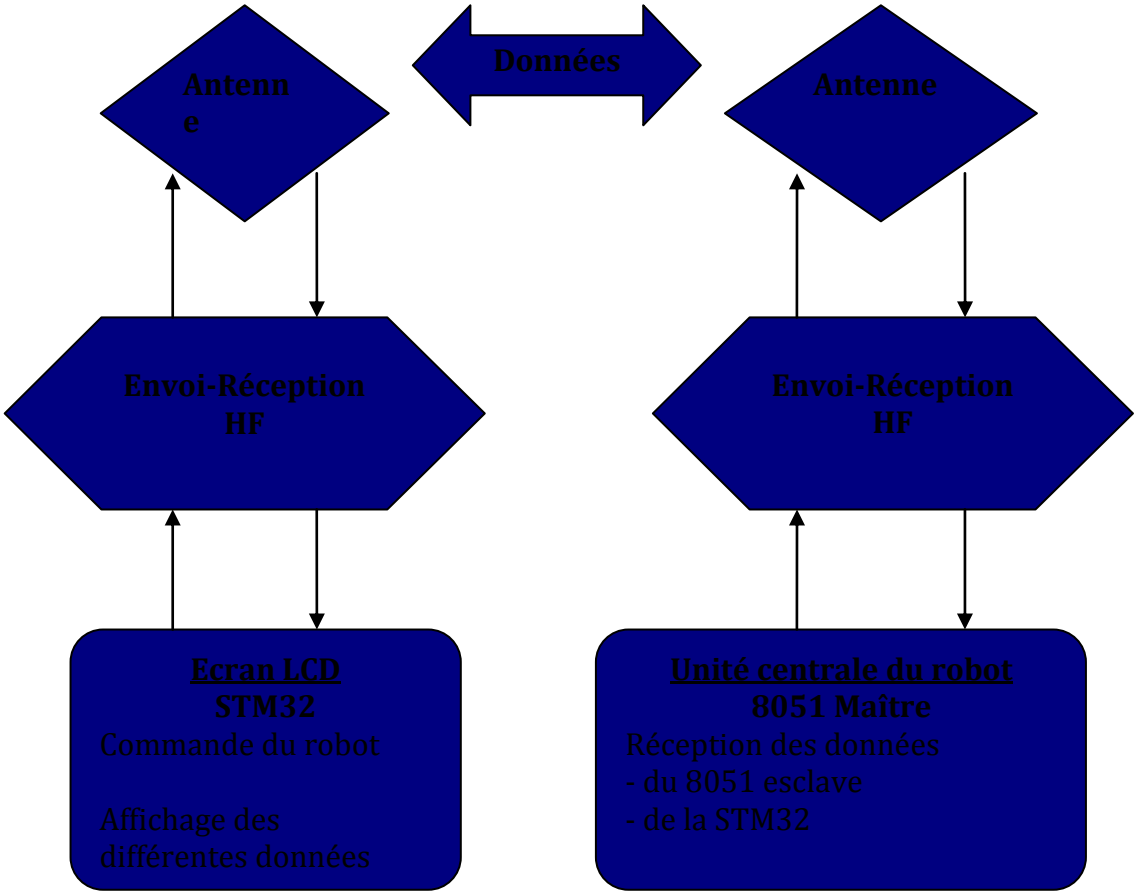
Les données remontent enfin jusqu'au maître via la liaison SPI, et sont affichées sur l'écran tactile grâce au microcontrôleur maître qui communique via le module Xbee.

7.3 Synoptique matérielle envisagé

Liaison 8051 maître-esclave :



Liaison 8051 maitre- STM32 :



7.4 Solution technique

7.4.1 L'affichage

7.4.1.1 Le principe

Afin que l'utilisateur puisse piloter de manière efficace le robot, l'objectif est ici de réaliser une interface tactile intuitive permettant à l'utilisateur de naviguer entre les différents sous-menus en fonction des opérations qu'il compte effectuer.

Lors du démarrage de la carte STM32, un contrôle du système est effectué afin de vérifier si l'utilisateur peut utiliser correctement l'écran tactile LCD.

Il est alors invité à appuyer sur une zone précise de l'écran afin qu'il accède au menu principal.

Le menu principal se compose de 3 zones que l'utilisateur peut actionner en fonction de son objectif :

La 1ère zone, « pilotage du robot », lui permet de commander le robot. Il accède ainsi à 4 boutons distincts qui définissent la trajectoire du robot, et envoient les instructions correspondantes au sérializer qui pilotera le robot.

Les boutons poussoirs implémentées correspondent aux directions possibles (Forward–Right–Left–Back).

Lorsque l'utilisateur appuie sur la zone concernée, les caractères « F », « R », « L » ou « B » sont alors transmis au microcontrôleur 8051 qui les analyse et commande le serializer. L'utilisateur commande ainsi le déplacement du robot.

La deuxième zone, « Information » permet à l'utilisateur d'afficher les données enregistrées par le robot, telles que la consommation en temps réel ou encore les informations traduisant la position des obstacles sur le parcours du robot.

La troisième zone, « Credits » correspond à une éventuelle option qu'il sera possible d'ajouter au robot.

Enfin, un retour au menu principal a été programmé dans chacune des sous catégories précédentes.

7.4.1.2 Le projet

Nous avons commencé par étudier le projet Mirror_Screen disponible sur l' ECampus, auquel nous avons ajouté les différentes sources qui permettent de mettre en place l'affichage convenu.

Le projet fourni, implémenté sous Uvision 4, contient les fichiers suivants :

-Stm32f10x_it.c ➔ Définition des fonctions d'interruptions

-Main2.c ➔ initialisation de l'écran LCD et affichage du menu

-Menu.h
Menu.c ➔ menu principal implémenté : accès aux différents sous-menus

-Pilotage.h
Pilotage.c ➔ Déplacement du robot

-Infomation.h
Information.c ➔ Affichage des données en provenance du microcontrôleur maître

7.4.1.3 Les fonctions utilisées

Lorsqu'on alimente ma carte STM32, une série de foncions d'initialisation se lancent pour que la configuration initiale de la carte soit mise en place :

Initialisation du système :

```
SystemInit();
STM_EVAL_LEDInit(LED ...);   // Initiation des LED
STM3210C_LCD_Init(); // Initiation de l'écran LCD
```

Initialisation de la liaison UART:

```
USART_InitStructure.USART_BaudRate = 19200;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

Config du port :

STM_EVAL_COMInit(COM1, &USART_InitStructure);
```

Une fois ces différentes instructions lancées, il est possible d'implémenter le menu auquel aura accès l'utilisateur.

Trois fonctions sont ainsi disponibles, permettant d'afficher une page blanche, de choisir la couleur de fond de l'écran ou du texte affiché.

```
LCD_Clear(White);     /* Clear the LCD */
LCD_SetBackColor(Blue);  /* Set the LCD Back Color */
LCD_SetTextColor(White);    /* Set the LCD Text Color */
```

Le type de menu implémenté est classique, et se distingue par :

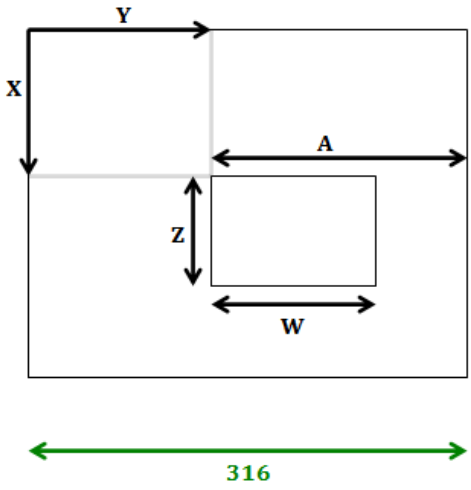
- l'affichage de texte sur des lignes prédéfinies

```
LCD_DisplayString(Line0, "   STM3210C-EVAL     ");
```

- la mise en place de rectangles qui définissent des zones précises selon les fonctions réalisées par notre télécommande.

```
LCD_DrawRect(170, 316-250, 80, 200);   //(X , Y=316 - A, Z , W )
```

L'écran tactile se décompose ainsi en zone de pixels (316 pixels de long) d'après le schéma suivant :



X-Y: Coordonnées de l'origine du rectangle
Z : Largeur du rectangle
W : Longueur du rectangle

Remarque :

Après initialisation de la carte, un test est lancé permettant de contrôler le fonctionnement correct de la carte STM32, rendu possible grâce à cette fonction :

```
if (IOE_Config() == IOE_OK)
 {
  LCD_DisplayStringLine(Line4, "   IO Expander OK   ");
 }
 else
 {
  LCD_DisplayStringLine(Line4, "IO Expander FAILED ");
  LCD_DisplayStringLine(Line5, " Please Reset the  ");
  LCD_DisplayStringLine(Line6, "   board and start ");
  LCD_DisplayStringLine(Line7, "    again         ");
  while(1);
 }
```

Maintenant que nous maîtrisons les fonctions de bases utilisables sur cette carte de développement, nous allons nous intéresser au basculement entre les différents sous-menus disponibles.

7.4.1.4 Les fonctions par interruption

Si la carte STM32 est prête à être utiliser, un rectangle est alors mis à la disposition de l'utilisateur qui lui permettra d'accéder au menu principal.

Le basculement entre les écrans programmés est réalisé à partir d'interruptions.
Il est possible de détecter les zones de l'écran où l'utilisateur va appuyer. Le rectangle présenté à l'écran défini alors une zone précise qui, active (l'utilisateur appuie sur la zone correspondante), lance le programme d'interruption et appelle la fonction d'affichage du menu principal.

```
#ifdef IOE_INTERRUPT_MODE
        static TS_STATE* TS_State;

        /* Check if the interrupt source is the Touch Screen */
          if (IOE_GetGITStatus(IOE_1_ADDR, IOE_TS_IT) & IOE_TS_IT)
                {
                /* Update the structure with the current position */
                TS_State = IOE_TS_GetState();

                if ((TS_State->TouchDetected) && (TS_State->Y < 220) && (TS_State->Y > 170))
                        {

                        if ((TS_State->X > 90) && (TS_State->X < 230))
                                {
                                afficher_menu();
                                }

                         }
                else
                        {
                        STM_EVAL_LEDOff(LED1);
                        STM_EVAL_LEDOff(LED2);
                        STM_EVAL_LEDOff(LED3);
                        STM_EVAL_LEDOff(LED4);
                        }

                IOE_ClearGITPending(IOE_1_ADDR, IOE_TS_IT);
                }
```

Les 3 zones implémentées dans le menu principal fonctionnent selon le même principe : on redéfinit un fonctionnement par interruptions dans le fichier « Menu.c »

Une première zone appelle la fonction « pilotage_fct() » : l'utilisateur bascule sur un nouvel écran où 4 boutons vont permettre d'envoyer des instructions de déplacement.

Le deuxième rectangle appelle la fonction « afficher_infos() » ; l'utilisateur bascule sur un nouvelle page où les données reçues, caractère par caractère via la liaison UART avec le module RF, vont pouvoir défiler.

Le 3e rectangle permet simplement d'allumer une LED, et sera implémenté en cas d'une éventuelle amélioration du robot.

7.4.1.5 Traitement des données

Envoi d'un caractère : (pilotage.c)

Lorsque l'utilisateur pilote le robot à l'aide des 4 boutons tactiles, il envoie en continu des caractères que le microcontrôleur va réceptionner.

Cette transmission est rendue possible grâce à l'implémentation du prototype « Putchar » dans le fichier Main2.c :

```
PUTCHAR_PROTOTYPE
{
USART_SendData(EVAL_COM1, (uint8_t) ch);

/* Loop until the end of transmission */
while (USART_GetFlagStatus(EVAL_COM1, USART_FLAG_TC) == RESET)
{}

return ch;
}
```

Deux possibilités s'offrent alors à nous pour transmettre les informations nécessaires à la commande du robot.

Les caractères définis peuvent être rassemblés dans des variables messages et sont transmis via l'UART grâce à la fonction « printf» qui envoie directement les caractères sur l'UART.

```
#define MESSAGE1 "F" // Forward
#define MESSAGE2 "L" // Left
#define MESSAGE3 "R" // Right
#define MESSAGE4 "B" // Back
```

```
printf("%s", MESSAGE1);
```

Il est également possible d'utiliser la fonction « USART_SendData » qui contrôle le port de l'UART et envoie le caractère rentré en paramètre, codé sur 8bits.

Les interruptions définies pour les 4 boutons tactiles utilisent donc ces fonctions pour envoyer les instructions correspondantes.

Réception d'un caractère : (information.c)

La réception des informations provenant du système est rendu possible grâce au prototype Getchar implémenté dans le Main2.c :

```
GETCHAR_PROTOTYPE
{
uint8_t ch;

/* Loop until the end of transmission */
while (USART_GetFlagStatus(EVAL_COM1, USART_FLAG_RXNE) == RESET)
{}
ch=USART_ReceiveData(EVAL_COM1);

return ch;
}
```

Il est alors possible de récupérer les caractères reçus sur la liaison UART et de les afficher sur l'écran tactile :

```
c=getchar();

                LCD_SetTextColor(Red);
                LCD_DisplayStringLine(Line5,&c);
```

Cependant, la mise en place d'un message défilant n'a pas été réalisée dans notre étude.

7.4.1.6 Tests réalisés sur la STM32

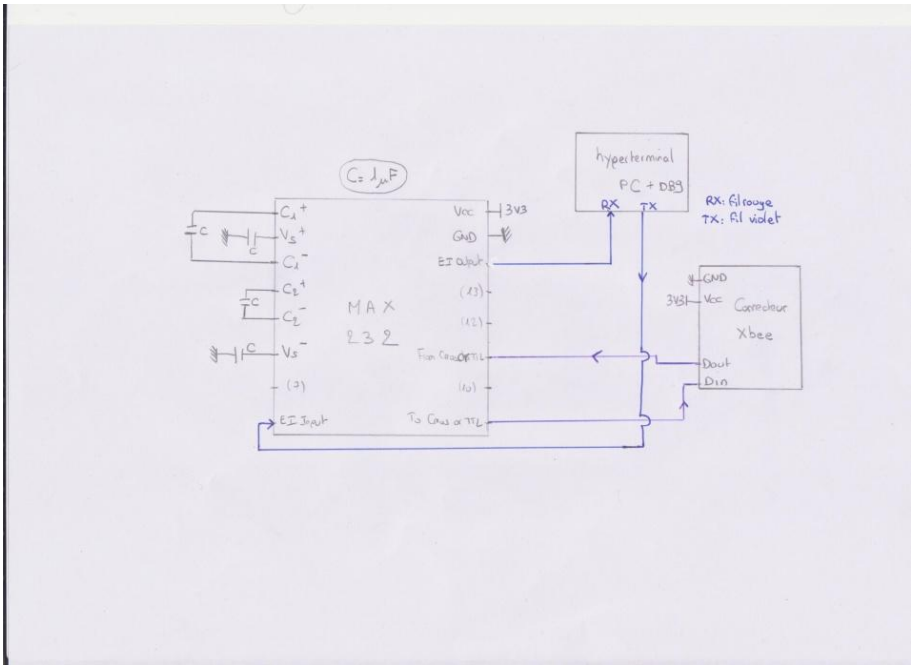
Les tests réalisés ont visé à la validation du jalon 1 : réaliser une interface de commande du robot à partir de l'écran LCD.

Test matériel :

Avant toutes choses il était nécessaire de tester les modules Xbee afin de contrôler la transmission RF des données.

Nous avons ainsi relié les 2 modules Xbee à 2 PC distincts (hyperterminal) puis tenté de transmettre un caractère rentré au clavier via la liaison RF.

Schéma :



Configuration :

-vitesse 19200 Bauds
-8bits -1 bit de stop
-sans parité
-sans contrôle de flux

Le caractère envoyé à partir de l'un des terminaux s'affiche bien sur l'autre terminal.

Tests logiciels :

Test 1 : envoi d'un caractère

Nous avons tout d'abord testé le prototype Putchar en analysant les caractères que nous transmettions après appui sur les boutons contrôlant le déplacement du robot.

Les caractères définis dans les messages s'affichent bien à l'écran à l'aide de la fonction « printf »: le flux est continu lorsque l'utilisateur reste appuyé sur le bouton.

Test 2 : réception d'un caractère

L'objectif était ici de visualiser un caractère transmis sur la liaison UART grâce au clavier sur l'écran LCD.

La fonction getchar permet bien de récupérer le caractère rentré au clavier et la fonction LCD_DisplayStringLine affiche le caractère sur l'écran tactile.

Ces tests ont ainsi conduit à la validation du Jalon1.

7.4.1.7 Aléas et perspectives d'évolution

La fonction permettant de recevoir des informations sur l'écran tactile (getchar) nous a posé un certain nombre de problèmes. Pendant plusieurs séances, nous n'arrivions pas à afficher un caractère sur l'écran tactile (problèmes de variable et de vitesse de transmission). Néanmoins, nous avons finalement réussi à l'implémenté et la STM32 affiche bien des caractères à partir du microcontrôleur 8051.

Si plus de temps nous avait été accordé, il aurait été possible d'améliorer l'affichage des informations reçues (message défilant).
De plus, l'utilisateur doit pouvoir choisir une fréquence à partir d'une gamme prédéfinie par le constructeur. Notre onglet « crédits » peut ainsi être implémenté pour remplir cette fonction et permettre à l'utilisateur de choisir la bonne fréquence de coupure parmi celles qui s'afficheront à l'écran.

7.4.2 L'UART1

(Partie rédigée par Arthur Simon)

7.4.2.1 Problématique

Le 8051 maître a plusieurs fonctionnalités à gérer :

- la commande du Serializer, et donc des moteurs, via liaison UART.
- la réception et l'envoi des données au 8051 esclave via la liaison SPI
- la réception et l'envoi des données à la carte STM32 via la liaison RF.

Ainsi, après avoir mis en place la commande du Serializer à l'aide de boutons poussoir (cf "Déplacement du robot") et après avoir programmé le menu et les fonctions de transmission - réception de données de la carte STM32, il nous fallait adapter le code développé sur le 8051 maître et le compléter afin de gérer la transmission des ordres au Serializer. L'UART0 étant déjà utilisée pour la liaison 8051<->Serializer, nous avons alors décidé d'établir la liaison RF STM32<->8051 en utilisant l'UART1.

7.4.2.2 Introduction à l'UART

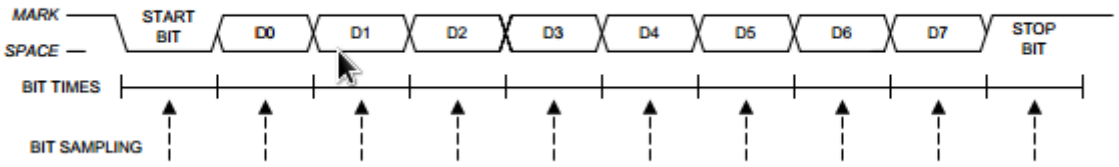
La liaison UART est une liaison série très classique sur ce type de microcontroleur. Elle peut être configurée en full-duplex ou en half-duplex, et pour plusieurs modes de transmission. Elle est associée à ces registres SFR : SCON pour la configuration de la liaison, et SBUF qui fait référence aux registres d'envoi ET de réception de données. Une lecture de SBUF permet d'accéder au registre de réception, une écriture au registre d'envoi, et ces attributions sont faites automatiquement.

L'UART1 peut être configurée en mode "polling" ou en mode interruption. C'est ce dernier qui nous intéressera par la suite. Il y a deux sources d'interruption : les flags d'interruption de transmission (TI1) et de réception (RI1). Ceux-ci sont accessibles dans le registre SCON1. Ils doivent être remis à zéro manuellement.

7.4.2.3 Configuration de l'UART1

Dans le cadre de ce projet, nous avons choisi le "mode 1" pour le fonctionnement de l'UART. Celui-ci permet une transmission asynchrone full-duplex sur 10 bits : 1 bit de Start, 8 bits de données et 1 bit de stop. Les bits de données sont stockés dans le registre SBUF1 (avant transmission ou après réception). Un mot de 8 bits nous permet ainsi d'envoyer et de recevoir des caractères, ce qui est idéal pour le contrôle du moteur. Cela devrait également permettre l'affichage des informations du télémètre et de la réception sonore sur la STM32.

Figure 21.4. UART1 Mode 1 Timing Diagram



L'outil *Configuration Wizard* permet d'obtenir la configuration des ports avec les critères suivants :

- enable UART0
- enable UART1
- enable SPI0
- enable Crossbar
- sorties en push-pull
- entrées en drain ouvert

On obtient la fonction d'initialisation des ports comme suit :

```
void PORT_Init (void)
{  P0MDOUT   = 0x55;
   P1MDOUT |= 0x40;
   P3MDOUT = 0x00;
   P3 |= 0x0F;
   XBR0     = 0x06;
   XBR1    = 0x00;
 XBR2     = 0x44;
}
```

Il faut également initialiser les registres propres à l'UART1 :

```
void UART1_Init (void)
{   SCON1   = 0x50;                 // SCON1: mode 1, 8-bit UART, enable RX
   TMOD    = 0x20;                 // TMOD: timer 1, mode 2, 8-bit reload
   TH1    = -(SYSCLK/BAUDRATE/16);    // set Timer1 reload value for baudrate
   TR1    = 1;                                  // start Timer1
   CKCON |= 0x10;                  // Timer1 uses SYSCLK as time base
   PCON  |= 0x90;                 // SMOD01 = 1
  EIE2 |=0x40;
}
```

Ayant choisi le même mode de fonctionnement et la même vitesse de transmission pour l'UART1 que pour l'UART0, les fonctions d'initialisation sont identiques. (Seule différence : SCON1 n'est pas adressable bit à bit).

SCON1=0x50=0101000

- SM01=0 et SM11=1 //choix du mode 1
- SM21=0 // bit de stop ignoré
- REN1=1 //reception enabled
- RI1=0 et TI1=0 // les flags sont abaissés, ils passeront à "1" lors qu'une transmission (TI1) ou une réception de donnée (RI1) sera effectuée sur SBUF1. L'interruption correspondante (priorité 20) sera alors engagée.

TMOD=0x20

- GATE1=0 // avec TR1=1 : Timer 1 enabled
- C/T1=0 //Timer 1 incrémenté par la clock définie par T1M (registre CKCON)
- T1M1=1 et T1M0=0 //Timer1 en mode 2

TH1 : registre 8 bits de rechargement du Timer1 en mode 2.

CKCON=0x10

- T1M=1 // utilisation classique de la clock (pas de division par 12)

PCON=0x90 (commun à UART0 et UART1)

- SSTAT0=0 et SSTAT1=0 // mode de fonctionnement configurable dans les registres SCONs.
- SMOD0=1 et SMOD1=1 // "baud rate divided-by-two disabled"

EIE2 |=0x40

- ES1=1 // active l'interruption de l'UART1.

C'est la mauvaise configuration du registre PCON qui nous a empêchés d'utiliser la liaison UART1 et nous a fait stagner pendant 5 séances. Le bit SMOD1 étant à "0" au lieu d'être à 1, la vitesse de transmission de l'UART était donc mauvaise.

7.4.2.4 Commande du robot

Comme on l'a dit précédemment, la réception des informations sera gérée grâce à l'interruption liée à l'UART1 selon la déclaration suivante :

```
void UART1_isr (void) interrupt 20
{
        if(SCON1 && 0x01 == 0x01)
        {
                recu_stm32=SBUF1;
        }

        //SBUF1='b';
        SCON1&=0xfc;                    //remise à zéro de RI et TI

}
```

La première condition "*if*" vérifie que l'on est bien dans le cadre d'une réception : le flag RI1 correspondant à la réception d'un mot dans SBUF1 est alors mis à 1. Dans ce cas, on récupère le caractère. Dans tous les cas, on s'assure que les bits TI1 et RI1 sont remis à zéro avant de quitter l'interruption.

NB : *//SBUF1='b'* correspondant à l'envoie du caractère 'b' sur la STM32.

Une fois que le caractère de commande envoyé par la STM32 est récupéré, le 8051 analyse l'information et le compare au cas de pilotage du moteur :

```
void Use (void) interrupt 14
{
	 TMR3CN &=0x7F;                    //masquage
	if(countertime==0)
	{
		if(recu_stm32=='f')
		{
			Avancer();
			countertime=10;
		}
		else if (recu_stm32=='b')
		{
			Reculer();
			countertime=10;
		}
		else if (recu_stm32=='l')
		{
			TournerG();
			countertime=10;
		}
		else if (recu_stm32=='r')
		{
			TournerD();
			countertime=10;
		}
		else
		{
			Stop();
		}
	 }
	else countertime--;

	recu_stm32='a';
}
```

La STM32 envoie les caractères '*f*', '*l*', '*r*' ou '*b*' selon la direction de déplacement que l'on veut donner au robot (respectivement vers l'avant, la gauche, la droite ou l'arrière). Le 8051 compare donc le caractère contenu dans ***recu_stm32*** à l'un de ces 4 caractères et envoie l'ordre correspondant au Serializer.

Si la STM32 n'envoie plus d'informations, l'interruption de l'UART1 ne se déclenche pas. En revanche, l'interruption du Timer3 se déclenche toutes les 35ms. On force donc la condition de STOP en y mettant un caractère autre (ici le caractère 'a'). Il n'y a pas de conflit lorsque la STM32 envoie des caractères puisque l'interruption de l'UART1 est beaucoup plus rapide.

7.4.2.5 Envoi d'informations

L'envoi de caractères à la STM32 est tout à fait fonctionnel. En plaçant un caractère dans SBUF1 comme on l'a vu un peu plus haut (*//SBUF1='b'),* on envoie le caractère '*b*' à la STM32 qui peut alors s'afficher dans la fenêtre *Informations*. Sur la STM32 on a d'ailleurs traité l'affichage de chaines de caractères sur une même ligne, qui s'efface lorsque la chaine arrive en bout de ligne et continue au début de cette même ligne. On n'est cependant pas allé plus loin, la liaison SPI n'étant pas fonctionnelle.

7.4.2.6 Tests logiciels

Afin de vérifier le fonctionnement de chaque partie à chaque étape de la conception, on a systématiquement fait appel à l'HyperTerminal pour vérifier la réception et la bonne gestion des données.

On vérifiait donc à chaque début de séance le fonctionnement de la STM32 en la branchant directement à l'ordinateur par une liaison RS232. Pendant un moment, nous vérifiions également la bonne transmission du signal par le module XBee en branchant celui-ci à l'ordinateur via un circuit MAX232. Cette solution a rapidement été abandonnée, et nous vérifiions ensuite la bonne transmission du signal par des tests matériels.

Enfin, nous regardions si l'information envoyée par la STM32 sur l'UART1 du 8051 était bien traitée en connectant l'UART0 à l'ordinateur : l'HyperTerminal devait alors afficher les messages de commandes envoyés au Serializer.

L'HyperTerminal était configuré à un baud rate 19200 avec 1 bit de stop et sans contrôle de flux matériel.

7.4.2.7 Tests matériels

Nous avons longtemps eu beaucoup de mal à configurer correctement la liaison UART1. Cependant, nous savions que le problème résidait dans la programmation du 8051. Il nous fallait donc, à chaque début de séance, valider le fonctionnement des parties déjà traitées. Finalement, nous vérifiions que le caractère, codé sur 8 bits, arrivait en sortie du XBee, c'est-à-dire à l'entrée du 8051, en regardant l'allure du signal à l'oscilloscope. Cette technique est très simple à mettre en œuvre et relativement sûre pour notre cas. Il nous arrivait également de valider cette étape grâce aux LEDs des supports XBee qui indiquent si des informations transitent par les modules RF.

7.4.3 La liaison SPI

(Partie rédigée par Sélim Boudjaoui)

7.4.3.1 Principe

Le fonctionnement autonome du robot se base sur la transmission de données entre les 2 microcontrôleurs 8051 qui gèrent les différents périphériques.

Le 8051 maître sélectionne ainsi le 8051 esclave et demande le rapatriement des données provenant des systèmes de détection. Il est alors en mesure de transmettre ces données à la carte STM32 par l'intermédiaire du Xbee.

L'utilisateur y accède grâce à l'écran tactile.

7.4.3.2 Réalisation

Quatre ports présentés dans le synoptique matériel de la liaison sont responsables du fonctionnement de la liaison SPI.

Les données transitent en « full duplex », c'est-à-dire qu'un bus est réservé pour l'envoi du maître vers l'esclave, et inversement. Tout caractère envoyé sur le registre SPI0DAT sera ainsi automatiquement transféré sur la liaison. Les ports MOSI (Master Output / Slave Input) et MISO (Master Input / Slave Output) des microcontrôleurs sont ainsi reliés deux à deux pour former la liaison.

Nous avons choisi que le microcontrôleur associé au serializer et au module Xbee sera maître de la liaison, et fournira donc le signal d'horloge CLK permettant de synchroniser les transferts. Un des ces ports (P1.0), configuré en sortie Push-Pull, permettra de sélectionner le microcontrôleur maître pour la transmission.

Configuration du microcontrôleur maître :

Cinq ports sont utilisés sur ce microcontrôleur pour implémenter la liaison. Grâce au logiciel « configuration Wizard », il est possible de déterminer leur configuration :

SCK :	Port P0.2
MISO :	Port P0.3
MOSI :	Port P0.4
NSS :	Port P0.5
CS :	Port P1.0

Les ports fonctionnant en temps que sortie doivent être configurés en sortie Push-Pull (SCK-MOSI-CS).

Voici la fonction implémentée permettant de configurer ces ports, l'UART0/1 étant déjà utilisés :

```
Void Port_Init()
{
P0MDOUT = 0x55; // P0 en push-pull
P1MDOUT = 0x41; // P1.0 en push pull
XBR0= 0x06 ; // positionnement des ports
XBR2= 0x44 ;
```

Configuration du microcontrôleur esclave :

On détermine de même la configuration des 4 ports de l'esclave gérant la liaison SPI. Cette fois ci, seule la sortie de l'esclave (MISO) sera configurée en Push-Pull.

SCK :	Port P0.2	
MISO :	Port P0.3	
MOSI :	Port P0.4	Push-Pull
NSS :	Port P0.5	

Voici la fonction implémentée permettant de configurer ces ports :

```
Void Port_Init()
{
P0MDOUT = 0x49;
P1MDOUT = 0x40;
XBR0= 0x06 ;
XBR2= 0x44 ;
}
```

Génération de l'horloge :

L'horloge fournie par le maître aura une fréquence de 200khz. La configuration du registre SPI0CKR permet de déterminer cette fréquence, en s'appuyant sur la fréquence d'oscillation du quartz externe de notre système (20 MHZ).

On a ainsi : SPI0CKR=0x36 ;

Initiation de la liaison :

Les 2 registres restant permettent d'initialiser la liaison et de configurer les bits qui seront envoyés via la SPI.
Le premier registre SPI0CN, par l'intermédiaire de ces 2 premiers bits, permet d'activer la liaison et de choisir les microcontrôleurs en jeu :

```
SPIEN = 1;    // Enable SPI

MSTEN= 1 ; // Master

MSTEN= 0 ; // Slave
```

Le second registre SPI0CFG permet de déterminer le nombre de bits qui seront transférés lors de l'envoi, selon l'horloge fournie par le maître.
Nous transférerons ainsi 8bits à la fois à chaque front montant de l'horloge.

```
SPIOCFG=0x07 ;

//SPIOCFG[2:0] =1 ;   // 8bits transférés.

// CPOL=0 ;   // 1er front montant
// CPHA=0 ;
```

7.4.3.3 Tests

Par manque de temps, les tests de cette liaison n'ont pas pu être concluants.
J'ai tout d'abord tenté d'envoyer un caractère en continu à partir du maître.
L'horloge de fréquence 200khz s'établit bien lorsqu'on pointe la sonde sur la broche CLK.
J'essaie par la suite de récupérer le caractère sur un HyperTerminal relié à l'esclave. Cependant ce caractère ne s'affiche pas, et l'HyperTerminal présente plusieurs caractères qui s'inscrivent en continu sur l'écran. Il est possible que le caractère soit mal analysé du fait de la vitesse configurée pour la liaison.

7.4.3.4 Perspectives d'évolution

La majeure partie du travail reste à faire puisque le but est de transférer les informations provenant du système de détection, et de les communiquer au microcontrôleur maître. Il est important de parcourir globalement le registre SPI0DAT afin de vérifier que l'ensemble de la chaîne de caractère est bien transféré.

L'objectif primaire reste bien évidemment de pouvoir transférer et afficher un simple caractère via la liaison SPI entre les 2 microcontrôleurs.

8 Avancement du robot

8.1 Fonctionnement du robot

Au terme des séances de projet, le comportement de notre robot possède les caractéristiques suivantes : il est capable de se déplacer dans l'espace dans toutes les directions et il est piloté par la télécommande de la carte STM32. La transmission des informations de déplacement sont transmises au robot via la liaison RF mise en place grâce aux modules radio Xbee.

Concernant la détection de l'environnement, l'ensemble servomoteur/télémètre effectue un balayage complet des 360° et réalise une mesure de distance avant et arrière tous les 10°. Il renvoie ensuite la distance minimale environnante avec l'angle de mesure ainsi qu'une indication permettant de savoir si l'obstacle le plus proche se trouve devant ou derrière le robot.

Nous pouvons également interroger les pingers des obstacles rencontrés. Le robot est capable d'émettre un signal sonore en direction des pingers et de recevoir et traiter le signal reçu.

8.2 Les perspectives d'évolution

A l'heure de ce rapport, le robot n'est pas entièrement fonctionnel. En fait, les principales parties du robot sont fonctionnelles indépendamment. Le principal élément manquant à l'établissement d'un robot entièrement fonctionnel est la mise en place de la liaison SPI. Si celle-ci était bien établie,

il n'y aurait alors plus qu'à mettre en place les protocoles d'envoi et de réception d'informations à la STM32 (position des obstacles, réception sonore et réponse à un pinger), et au vu de l'état actuel du projet, ceci ne devrait pas être si difficile à mettre en place si la liaison SPI était fonctionnelle. On atteindrait alors la phase 1 de développement du robot. L'étape suivante consisterait donc à programmer un comportement autonome au robot qui, tenant compte des informations recueillies par les télémètres et par les échanges sonores effectués avec les pingers, pourrait se déplacer sans pilotage manuel.

9 Bilan des membres de l'équipe

9.1 Organisation de l'équipe

Tout au long de ces séances, nous avons pu expérimenter la construction et l'évolution d'une équipe travaillant de concours et visant un même objectif. N'ayant jamais travaillé ensemble, les débuts ont été quelque peu hésitants. En effet, la coordination d'une équipe de 6 personnes ne se fait pas en un jour. Mais peu à peu, chacun a trouvé ses marques et une sorte de routine s'est installée. Il n'était plus réellement nécessaire de se réunir chaque début de séance pour connaître les avancements et les objectifs de chaque partie ou de chaque membre. En fait, les membres de l'équipe se sont peu à peu faits confiance et, en raison de la proximité, on se tenait au courant du travail de chacun en temps réel.

Cependant, il est évident que cet état d'esprit et cette ambiance de travail ne peuvent pas être les seules clés de l'avancement d'un projet plus long dans la durée. On a pu l'entrevoir au moment du jalon 1 : en effet, une étape clé dans l'avancée d'un projet nécessite la mise à plat de ce qui a été fait, voire d'une définition nouvelle des objectifs à suivre ensuite.

9.2 Bilan des membres de l'équipe

Responsable planificateur :
Boudjaoui Selim

Conduire un projet n'est pas une tâche facile puisqu'il est nécessaire de respecter les délais convenus. Lors de la pré-étude du système, les différentes étapes du projet ont été planifiées et synthétisées dans le Diagramme de Gantt fourni. En temps que responsable planificateur, mon rôle a donc été de veiller au bon déroulement de notre étude en confrontant objectifs fixés au préalable et avancement de l'équipe au cours du temps.

Je me suis ainsi appuyé sur les comptes-rendus de séance, qui résumaient objectifs atteints et objectifs attendus pour la prochaine séance.

Le premier axe du projet, après avoir rendu le dossier de pré-étude, consistait à valider le Jalon 1, c'est-à-dire présenter une solution matérielle et logicielle indépendante pour chacun des modules du système.

Prévu pour les séances PE6 ou PE7 (mi-avril), nous n'avons malheureusement pas réussi à respecter les délais prévus, et le jalon 1 a été validé à la séance suivante PE8 datant du 11 mai.

PE6 a en effet été supprimée, ce qui nous a restreints à préparer un code théorique des modules entre les séances PE5 et PE7 (3 semaines).

Grâce à ce travail personnel, l'équipe a réussi à implémenter les fonctions de base de notre robot, conduisant à la validation du jalon.

Le second axe du projet a alors consisté à apporter les dernières modifications sur les modules, puis de créer le code permettant de les assembler sur le robot. Cette étape nous a causé un certain nombre de difficultés car les codes théoriques implémentés n'ont pas conduit au bon fonctionnement au robot. Plus précisément, les codes gérant les liaisons entre les microcontrôleurs ou les informations provenant de la télécommande n'étaient pas au point, entraînant un retard sur le planning convenu. La validation du jalon 2 a ainsi eu lieu lors de la dernière séance PE11 datant du 01 juin.

En conclusion, le planning a été respecté, même si l'équipe a pris du retard sur l'avancement des tâches. Il est intéressant de revenir sur l'analyse temporelle faite lors de la pré-étude qui après coup parait moins proche de la réalité.

Planifier différentes opérations en parallèles n'est donc pas chose facile, et le responsable se doit de contrôler et relancer plusieurs fois son équipe afin que tout se passe comme convenu. Enfin, les délais critiques de réalisation des opérations se doivent d'être bien connus de tous, permettant à l'équipe de travailler en amont pour la prévention d'un ralentissement du projet.

Responsable qualité générale - gestionnaire des rendus : Hélène VOLOT

Mon rôle organisationnel dans ce projet a été de superviser la qualité des rendus, de vérifier le travail effectué par rapport aux consignes et de mettre en place des outils de travail.

Donc il s'agissait surtout au début du projet de mettre en place des outils de communication comme dropbox. Ensuite, comme il fallait que je supervise la qualité des rendus j'ai passé beaucoup de temps sur le rapport de pré-étude et plus récemment dans la rédaction du rapport final afin d'en vérifier la qualité et le rendu. C'est une responsabilité qui m'a demandée beaucoup de temps et d'implication, mais qui m'a aussi beaucoup appris sur le plan managérial. En effet j'ai du manager le reste de l'équipe pour pouvoir rendre le rapport à temps.

Responsable chef de projet : Arthur Simon

Si ce projet fut passionnant par l'objectif qui était visé et par les compétences techniques à mettre en œuvre au sein d'une équipe qui se devait d'être coordonnée, il fut assez frustrant de passer tant de temps sur la configuration de l'UART1 pour ce qui s'est avéré être en fait une erreur de configuration sur un seul bit. Ceci nous a empêchés de poursuivre plus loin et d'obtenir un robot fonctionnel : il semble qu'à l'heure de ce rendu, l'établissement de la liaison SPI soit la seule étape manquante pour obtenir un robot complètement fonctionnel.

Enfin, si tous les membres de l'équipe peinaient quelque peu à trouver leurs marques en tout début de projet, chacun est peu à peu devenu autonome et confiant dans le travail qu'effectuaient les autres membres.

En tant qu'animateur de projet, si j'ai dû au tout début aller voir chaque membre un par un à chaque début de séance pour connaître son avancement et sa motivation et tenter d'établir une certaine cohésion dans l'équipe, mon rôle s'est peu à peu révélé inutile en raison de l'excellente ambiance de travail qui a fini par s'établir.

Responsable développement durable :
Florian Girodet

Le développement durable vise à créer de la richesse économique tout en adoptant une démarche sociale et environnementale responsable.

En tant que responsable de ce secteur mon rôle était d'évaluer l'impact des activités réalisées au cours du projet sous les trois aspects suivants : environnemental, social et économique.

Le rôle de communicant est primordial pour ce type de responsabilité car la démarche de développement durable devait être diffusée en interne. Même si pour ce type de projet nous n'étions pas en situation de création d'entreprise il a fallu s'assurer que la plupart des composants fournis respectent des conditions environnementales, et économiques.

Dans le cas d'une création d'entreprise Il aurait fallu engager des actions d'information, de sensibilisation et de conseil auprès de l'ensemble des membres du groupe afin que les termes sur le développement durable soient respectés, plus précisément ceux concernant les certifications environnementales.

Responsable communication :
Axel Laroche

Mon rôle organisationnel dans ce projet a été de superviser la qualité des rendus, de vérifier le travail effectué par rapport aux consignes et de mettre en place des outils de travail.

Donc il s'agissait surtout au début du projet de mettre en place des outils de communication comme dropbox. Ensuite, comme il fallait que je supervise la qualité des rendus j'ai passé beaucoup de temps sur le rapport de pré-étude et plus récemment dans la rédaction du rapport final afin d'en vérifier la qualité et le rendu.

C'est une responsabilité qui m'a demandée beaucoup de temps et d'implication, mais qui m'a aussi beaucoup appris sur le plan managérial. En effet j'ai du manager le reste de l'équipe pour pouvoir rendre le rapport à temps.

Responsable Qualité – Tests – Contrôle technique :
Romain Ringwald

Mon objectif primordial a été l'évaluation de la conformité des réalisations par rapport aux jalons annoncés. Cette mission a été dans la majorité remplie car nous avons fait valider la plupart des objectifs mis dans nos jalons. Seuls quelques retards dus à des difficultés rencontrées nous ont empêchés d'atteindre ces objectifs.
Ma deuxième mission a été de veiller à la bonne gestion des ressources matérielles et à l'élaboration de la documentation technique. Je n'ai rencontré aucun problème au cours de ma mission ; toute l'équipe a pris soin du matériel.

9.3 Bilan final du projet

Ce projet a été très intéressant car il fut pour nous la première occasion d'acquérir et de mettre en œuvre des compétences sur un projet dont le scénario semble relativement réel et dont l'objectif est absolument passionnant. Nous avons tous pu nous spécialiser techniquement sur des parties totalement différentes du projet et travailler tout de même ensemble afin de pouvoir réunir des parties développées séparément et faire fonctionner le robot (bien que le robot ne soit pas entièrement fonctionnel). Ce fut donc une expérience enrichissante tant du point de vue technique que du point de vue humain, puisqu'on a très bien pu observer que les hauts comme les bats dans la motivation d'un membre pouvaient influer sur l'équipe toute entière. Les individualités étaient à prendre en compte, car la motivation est un facteur primordial dans la bonne conduite d'un tel projet.

10 Annexes

Concaténation des comptes – rendus et jalon

PE4 :
Compte rendu de séance 28.03.2012 équipe A4

Objectifs de la séance : réalisation des premiers tests matériels et logiciels.

Nous avons eu accès pour la première fois au matériel que nous utiliserons pour le projet.

Hélène Volot et Axel Laroche :
-Nous avons réalisé des tests sur le servomoteur.
-Nous avons appliqué un signal TTL de largeur d'impulsion définie (entre 600 us et 2400us) : en fonction de l'impulsion appliquée, le servomoteur a tourné d'un angle qui correspondait bien à l'angle voulu.
-Nous avons ensuite réalisé des tests sur le télémètre : nous avons mesuré différentes distances (comprises dans l'intervalle de mesure du télémètre), la télémètre renvoie une tension qui correspond bien à la distance mesurée.
-Nous avons ensuite commencé à étudier le fonctionnement du CAN intégré au 8051 pour l'utiliser avec le télémètre.

Selim Boudjaoui et Arthur Simon :
-Test communication entre 2 PC (HyperTerminal) via 2 modules XBee interfacés par un max232.
-Prise en main de la carte STM 32, notamment implémentation de l'écran LCD, allumage des LEDs et rafraîchissement de l'écran.

Romain Ringwald :
- Test de la commande : mogo 1 :10 2 :-10 => on voit bien que les roues tournent a la même vitesse mais de sens opposé.
- Commande Stop : arrêt du fonctionnement du moteur
- 1 tick -> 0.020 inch et 1inch ->2.54cm => 1tick->0.0508 cm
- Ticks per unit distance: 51.02 ticks/inch of linear travel
- 51.02 ticks -> 1 inch
- 51.02 ticks -> 2.54cm
- Pour deplacer de 20cm par ex on rentre comme param 201 et on constate que ca avance bien de 10 cm digo 1 :201 :10 2 :201 :10
- Tourner de 90°
  >digo 1:1000:20 2:1000:-20
  ACK.

En conclusion : vérification du fonctionnement correct du robot et de ses commandes avec un module Xbee et un HyperTerminal : Avancer, s'arreter, tourner à gauche, à droite.

Florian Girodet :

Test de la phase 1 de la communication acoustique : mise en place sur plaquette lab le microphone à capsule électret de référence Panasonic WM-62A ainsi que l' amplificateur de puissance que l'on a décidé de lui associer, à savoir un LM4871 avec notamment une résistance de Pull-Up de 5,5 MOhm (tous les autres composants de cet amplificateur sont quantifiés dans la doc technique fournie). Les différents modules qui composent le système d'émission du signal ont donc été raccordés. On a ensuite procédé à des tests sonores en envoyant un signal sinusoïdal de fréquence variable en entrée de l'amplificateur afin d'émettre un son à sa sortie, c'est-à-dire via le microphone. On a ensuite pu analyser les formes des signaux aux différents points du circuit à l'aide de l'oscilloscope. On peut facilement grâce à ceci obtenir le signal sonore que l'on veut, c'est-à-dire imposer la fréquence que l'on veut au signal afin d'être en phase avec les fréquences particulières données dans le cahier des charges.

Objectifs pour la prochaine séance :

-Configuration du PI pour qu'il avance d'une distance plus précise (en effet pour un paramètre d'avancement de 30 cm il avance que de 25cm) et mise en place la liaison série avec le microcontrôleur.
-Etude du CAN et choix d'une solution d'utilisation.
-Gestion de la synthèse du signal à envoyer dans l'amplificateur grâce au microprocesseur.

PE5 :
Compte rendu de séance 18.04.2012 équipe A4

Objectifs de la séance : finalisation des différents éléments à l'approche du premier jalon.

Hélène Volot et Axel Laroche : élaboration des premières fonctions utilisées pour le fonctionnement du servomoteur : initialisation des ports et du PCA, génération d'une impulsion de largeur temporelle variable.

Nous avons effectué un choix de méthode pour la génération des impulsions temporelles (utilisation de l'interruption PCA).

Romain Ringwald : réalisation du programme utilisé pour le sérialiseur et de tests avec la télécommande avec boutons poussoir : tests OK

Florian Girodet : théorie sur l'étage de pré-amplification et de filtrage du signal reçu par le robot et choix des composants. réalisation de code pour l'émission du signal.

Selim Boudjaoui & Arthur Simon : affichage sur carte STM32 et liaison UART de la carte STM32 vers le PC via modules Xbee.

L'objectif de la prochaine séance est le premier jalon (prévu pour la 6ème séance) c'est à dire le fonctionnement indépendant de chaque élément.

PE7 :
Compte rendu de séance 04.05.2012

Selim Boudjaoui - Arthur Simon :

- Communication STM32 -> Hyperterminal : affichage d'un message prédéfini dans le programme sur l'Hyperterminal du PC.
- Communication PC -> STM32 : on tape un caractère au clavier, on récupère ce caractère et on l'afficher sur l'écran tactile.
- Conception du menu de base : mise en place de différents sous-menus et clear de l'écran, fonctionnement par interruption

Hélène Volot :

application du code pour le télémètre :
création du main(),
test de la liaison entre le PC et le 8051 OK.
Cependant la communication ne fonctionne pas avec le télémètre.

Florian Girodet :

Câblage de plusieurs types de filtres pour la détection du signal sonore à la fréquence de 1046 Hz. Tous les filtres n'ont pas été retenus comme étant une solution au problème, faute d'un signal en sortie du filtre inexploitable (incompréhension du problème par les responsables sonores de plusieurs groupes).Ainsi en fin de séance j'ai retenu pour ce filtrage la mise en place d'un filtre passe-bande de Sallen and key d'ordre 2. Les éléments qui le compose ont été calculés via le site internet de Microchip.
Pendant la séance j'ai également procédé au câblage optimal du module d'émission sonore.

La prochaine séance sera consacré au câblage du filtre de S&K et surtout à la pré-amplification du signal reçu avant filtrage. Les codes concernés seront réalisés par un autre membre de l'équipe, également pendant la séance. Peut-être auront nous le temps de souder les composants des différents modules.

Romain Ringwald :

- Configuration du sérialiseur et réalisation du programme pour que :
-> appui sur un bouton poussoir = avance, recule, tourne ...
-> lache = stop- test OK

Axel Laroche :

Test du programme pour le servomoteur :

- réalisation du main : le servomoteur balaye les 180° avec un pas de 10° (le pas est paramétrable) et réalise une pause de 1 seconde grâce à une fonction sleep.

- chaque seconde la LED du port P1.6 change d'état (pour bien visualiser le pas)

Objectif de la prochaine séance : préparation de chaque module pour la présentation du jalon 1 en fin de séance.

PE8: Jalon 1

11/05/12

Ci-après la liste des modules et des fonctions soumise à l'évaluation :

Module Déplacement (Romain Ringwald).

-Test entre le PC et le 8051 : Exécution de fonctions implémentées sur le 8051. Allumage ou non de la LED verte incorporée du 8051 pour vérifier qu'on rentre bien dans les fonctions dédiées à l'appui précis d'un Bouton Poussoir.

-Test entre le PC et l'HyperTerminal : On observe la fonction exécutée en par le 8051 grâce à l'HyperTerminal. On observe la commande sur l'HyperTerminal, du type « mogo 1 :30 »

-Test entre le 8051 et serialiser, espionné par l'HyperTerminal. On exécute les fonctions implémentées sur le 8051 et on observe le résultat observé : déplacement du robot selon l'appel de la fonction grâce à un BP précis et affichage de la commande appelée sur l'HyperTerminal.

-Test de la phase 1 du projet : Mise en mouvement du robot grâce à la télécommande (BP) et aux fonctions implémentées sur le 8051. **Test OK**

Configuration de la STM32 (Selim Boudjaoui et Arthur Simon).

-Communication STM32 – HyperTerminal : Affichage d'un message prédéfini dans le programme sur l'HyperTerminal du PC.

-Communication PC – STM32 : On tape un caractère au clavier et on récupère ce caractère et on l'affiche sur l'écran tactile.

-Conception du menu de base : Mise en place de différents sous-menu et « Clear » de l'écran.

Détection d'obstacle (Laroche Axel, Hélène Volot)

-Test rotation servomoteur (Axel Lroche) : Balayage du servomoteur avec un pas variable (tests avec 10°, 20°, etc…)

-Pour observer les pas, mise en place d'une fonction « sleep » qui réalise une temporisation en secondes. Pour observer ce pas, changement de l'état de la LED (P1.6)

-Application du code pour le télémètre : création du main(), test de la liaison entre le PC et le 8051. La communication ne marche pas encore avec le télémètre.

Emission et réception sonore (Florian Girodet)

-Test émission sonore : ok.

-Réception sonore : En cours de développement.

PE8 :
Compte rendu de séance 11.05.2012

Objectif de la séance : Jalon 1. Chaque partie indépendante est à un stade d'avancement précisé dans la fiche jalon.

Romain Ringwald :

Présentation de la partie moteur : Mise en mouvement du robot grâce à la télécommande (BP) et aux fonctions implémentées sur le 8051. Test OK

Objectifs prochaine séance : modification du code pour permettre le fonctionnement avec la crate STM32 pour le pilotage et travail sur la réalisation du filtre avec Florian.

Florian Girodet :

Présentation de la partie emission-réception sonore : test émission sonore OK.

Objectifs prochaine séance : développement du filtre pour la réception sonore

Selim Boudjaoui et Arthur Simon :

Présentation de la partie STM32 : menus de navigation et affichage à l'écran d'une commande

Objectifs de la prochaine séance : programmation du 8051 pour piloter l'ensemble moteur-serializer depuis la STM32.

Hélène Volot :

Présentation de la partie télémètre : la communication ne fonctionne pas encore entre le télémètre et le PC.

Objectifs prochaine séance : travail sur la partie télémètre, résolution du problème.

Axel Laroche :

Présentation de la partie télémètre : tests ok. balayage de 180° degrés avec un pas variable et une temporisation.

Objectif prochaine séance : travailler sur la partie télémètre avec Hèlène.

PNE1 :
Compte rendu de séance 15.05.2012

Selim Boudjaoui : définition des registres pour la liaison SPI.

Arthur Simon : réception via Xbee des informations envoyées par la STM32 sur le 8051 (commande moteur)
Objectifs prochaine séance : test sur le matériel (différents registres implémentés)

Romain Ringwald : réalisation du filtre pour la réception sonore. simulation correcte sous PSpice et cablâge sur plaquette.
Objectifs prochaine séance : test du filtre avec le matériel

Florian Girodet : développement du code émission sonore.
Objectifs prochaine séance : test du code sur le matériel

Hélène Volot et Axel Laroche : tente toujours de résoudre le même problème sur la fonction de début de conversion. A essayé une nouvelle solution de conversion, qui ne fonctionne toujours pas.
Objectifs prochaine séance : débuggage et essai sur matériel pour tenter de trouver une solution

PNE 2-PE9 :
Compte rendu de séance lundi 21/05/2012 et mardi 22/05/2012

Hélène Volot et Axel Laroche

Résolution du problème concernant le télémètre, utilisation de l'ADC0 au lieu de ADC1.

Le télémètre est donc fonctionnel et mesure des distances entre 25 et 150cm.

Arthur Simon

travail sur la réception du microprocesseur a partir de la STM32 pour controler le moteur. Problème avec l'étape réception pour l'UART1.

Selim Boudjaoui

travail sur la liaison SPI pour faire communiquer les 8051 maitre et esclave

Florian Girodet et Romain Ringwald

travail sur le filtre pour la réception des signaux (montage et code) : synthèse de filtre avec Filterpro et synthèse du filtre.

Objectifs prochaine séance : séance non encadrée donc rédaction des parties du rapport, optimisation du code (commentaires etc), montage du filtre de réception et test du fonctionnement.

PNE3 :
Compte rendu de séance 25/05/2012

Hélène Volot & Axel Laroche

Mis en commun des codes pour le télémètre et le servomoteur et résolution de conflits. Rédaction des parties du rapport.

Objectifs prochaine séance : résolution des conflits et problèmes entre les deux codes (utilisation des mêmes ressources par exemple).

Selim Boudjaoui & Arthur Simon

Communication entre le 8051 et la STM32.

Objectifs prochaine séance : toujours travail sur la communication 8051-STM32.

Florian Girodet & Romain Ringwald

Finalisation de la partie analogique : réalisation du filtre, pré amplification, détection de crête.

Objectifs prochaine séance : soudure du filtre, et test du programme.

PE10 :
Compte rendu de séance 30/05/12

Romain Ringwald & Florian Girodet

Test partie analogique ok : émission, réception, préamplification, filtrage, détecteur de crête.
Finalisation de la soudure du montage.

Objectifs prochaine séance : tests partie logiciel.

Axel Laroche & Hélène Volot

Mise en commun des codes télémètre et servomoteur et test sur le robot.
Problème rencontré : le 8051 du robot ne fonctionne pas, signalé à M. Daviot qui a récupéré la plaque pour tests. De plus, un télémètre est HS (le télémètrer avant), il affiche toujours la même distance, alors que le télémètre arrière est fonctionnel (problème signalé également).
Mise en place du multiplexeur pour l'utilisation alternative du télémètre avant et arrière.
Mise en place d'un tableau stockant toutes les valeurs des distances mesurées et des angles correspondant puis comparaison des valeurs et renvoi de la distance minimale autour du robot.

Objectifs prochaine séance : test du code modifié (ajout du multiplexeur et de la fonction de comparaison) sur le robot si le télémètre avant est fonctionnel.

Selim Boudjaoui & Arthur Simon

Gestion de l'UART1 du 8051 maitre pour la communication avec la STM32 (commande du robot).
Objectifs prochaine séance : finalisation de la gestion de la communication. Si ok, mise en place de la liaison SPI entre les 8051 maitre et esclave.c

Demande de jalon 2 :

Partie de Romain et Florian :
Démonstration de la partie analogie soudée. (Avec la possibilité de la présenter avec le logiciel).

Partie de Sélim et Arthur :
Présentation de la communication STM32-8051, c'est-à-dire le pilotage du robot.
Cette partie ne fonctionne pas encore correctement.

Partie d'Axel et Hélène :
Démonstration de la partie télémètre/servomoteur : le télémètre avant fonctionnent avec le servomoteur mais un des télémètres étant cassé, on ne pourra tester notre multiplexeur et donc les 2 télémètres ensemble que s'il a été réparé.

PE11 :
Compte rendu de séance 01/06/2012 équipe A4

Hélène Volot & Axel Laroche
Passage du jalon 2 : l'ensemble télémètre+servomoteur effectue un balayage des 180° avec un pas de 10°. Il affiche à chaque 10° la distance à l'avant, à l'arrière et l'angle actuel sur l'Hyperterminal. Il stocke également chaque valeur dans un tableau qui contient les angles et les distances correspondantes (avant et arrière). A la fin des 180°, il retourne la distance minimale autour du robot avec l'angle correspondant et le coté de l'angle (avant ou arrière).

Selim Boudjaoui & Arthur Simon
Passage du jalon 2 : Commande du robot grâce a la STM32 via l'UART0 (UART1 non fonctionnel).

Romain Ringwald & Florian Girodet
Passage du jalon 2 : partie analogique.
Soudure d'un module qui ne fonctionne pas donc recablage sur plaquette pour la démo.