



Rapport projet intégré

Antoine AUPÉE – Joachim DRAPS – Nathan DWEK

14 mai 2015

Table des matières

1	Introduction	1
1.1	Analyse du cahier des charges	1
2	Communication	2
2.1	Codage de l'ordre	2
2.2	Notre chaîne d'acquisition	3
3	Régulation	5
4	Transmission des ordres entre les deux microcontrôleurs	6
4.1	Première analyse	6
4.2	Configuration des modules UART	7
4.3	Programmation des émetteurs et récepteurs	9
5	Interprétation des ordres	12
5.1	Variables d'état externes de la régulation du robot	12
	Table des figures	a
	Table des codes source	b

Chapitre 1

Introduction

Le but de ce projet intégré est de doter un robot d'un système de contrôle à distance. La base mécanique du robot est fournie, et celui-ci est déjà muni d'une première carte à microcontrôleur complète. Cette dernière permet d'une part de s'interfacer avec les moteurs et les encodeurs à partir du premier microcontrôleur, et d'autre part d'accéder à certaines pattes de celui-ci ainsi que des alimentations. Les autres blocs nécessaires doivent être conçus et implémentés par les étudiants et le code des microcontrôleurs doit être produit.

Ce rapport décrit la démarche adoptée, ainsi que les solutions techniques adoptées pour mener à bien ce projet. Dans un premier temps le cahier des charges doit être analysé pour diviser l'objectif final en différent sous-problème bien définis.

1.1 Analyse du cahier des charges

Le robot doit

Chapitre 2

Communication

Pour ce projet, notre robot doit être capable de répondre à trois ordres différents : avance tout droit, tourne à droite, tourne à gauche. Le signal audio que nous lui envoyons doit donc contenir cet ordre ainsi qu'une quantité associée : le nombre de centimètres à parcourir ou le nombre de degré qu'il doit tourner. Dans ce chapitre, nous allons développer le chemin que parcourt l'information entre le moment où elle est envoyée sous forme sonore par l'émetteur qui nous a été fourni et la réception par le microcontrôleur.

2.1 Codage de l'ordre

Le message envoyé au robot est codé sur 13 bits. Le premier et le dernier sont tout deux mis dans l'état bas 0, ils font office de start bit et de stop bit, c'est-à-dire qu'ils délimitent la commande. L'avant dernier bit est un bit de parité, il permet de détecter les cas où la commande est erronée parce qu'elle contient un nombre impair de bits. Les 10 bits restant contiennent l'information utile, les deux premiers donnent l'ordre suivant la convention suivante :

- 0b00 : avance tout droit
- 0b01 : tourne à droite
- 0b10 : tourne à gauche

Les 8 bits suivants représentent simplement le nombre de degrés ou de centimètres à parcourir. Par exemple, la trame 010111111110 fera tourner notre robot de 255° dans le sens anti-horlogique.

Pour ce qui est de mettre en forme le signal audio, une fonction Matlab se charge de générer le signal audio modulé en FSK correspondant à l'information que l'on désire envoyer. Le principe de la modulation FSK est simple : la fréquence du signal correspond soit à l'état haut soit à l'état bas. Dans notre cas, chaque bit de la trame d'information sera émis sur une période

de 1 ms, un bit 1 sera représenté par une fréquence de 1100 Hz et un bit 0 par une fréquence de 900 Hz.

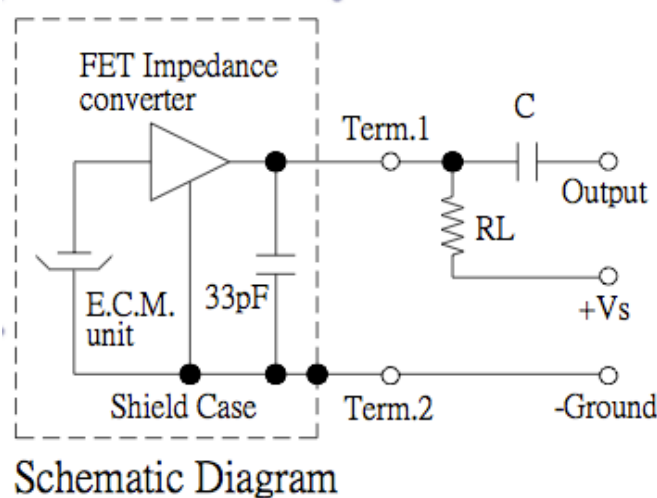
2.2 Notre chaîne d'acquisition

Notre chaîne d'acquisition est constituée de plusieurs étages :

- Le micro
- L'amplification
- Le filtre de garde
- Le convertisseur analogique-numérique
- Les filtres passe-bande

2.2.1 Le micro

Notre microphone fournit un signal d'une amplitude maximale de 1 mV. Pour le faire fonctionner il a fallu le polariser avec une résistance de 2.2 k Ω suivant le schéma de la figure 2.1.



$$RL=2.2K\Omega$$

FIG. 2.1 : Circuit du microphone

Le principe de fonctionnement du micro est que sa membrane extérieure mobile constitue l'armature d'un condensateur chargé en permanence, la

deuxième armature étant fixe. Le mouvement de cette membrane, dû aux variations de pression, va modifier la capacité du condensateur, et donc la tension à ses bornes. Le microphone est dit piézoélectrique. Dans notre cas, le microphone est aussi dit "à électret", ce qui signifie qu'il ne nécessite pas d'alimentation pour maintenir le condensateur chargé : le matériau constituant la membrane présente la propriété de conserver une charge électrostatique. Cependant, une alimentation est nécessaire pour polariser le transistor de l'étage de sortie du microphone, au travers de la résistance R_L .

2.2.2 Montage amplificateur

Comme nous l'avons dit précédemment, le signal en sortie de notre micro a une amplitude maximale de 1 mV. La plage des tensions d'entrée de l'ADC va de 0 V à 3.3 V, afin d'occuper cette plage le plus largement possible tout en gardant une marge de sécurité par rapport à la saturation de l'ADC il nous était demandé d'amplifier notre signal de façon à atteindre une valeur crête à crête de 3 V. Pour cela nous avons utilisé deux étages amplificateurs, respectivement d'un gain de 68 et 22.

La seule source d'alimentation dont nous disposons étant de 3.3 V, il a fallu polariser nos étages amplificateurs autour de 1.65 V pour éviter tout problème de saturation. Nous avons donc utilisé le montage représenté à la figure ???. Pour calculer

Chapitre 3

Régulation

Dans cette partie

Chapitre 4

Transmission des ordres entre les deux microcontrôleurs

Ce chapitre couvre la transmission des trames démodulées par le premier microcontrôleur vers le deuxième microcontrôleur. Dans la section suivante, le format des entrées et sorties de ce bloc vont être détaillées.

4.1 Première analyse

Puisqu'on désire limiter le nombre d'opérations effectuées par le microcontrôleur effectuant le traitement du signal audio, les trames de 10 bits renvoyées par la fonction `fskDetector` sont envoyées telles quelles au deuxième microcontrôleur. L'entrée du bloc transmission est donc une trame de 10 bits. L'uart, implémenté en hardware des deux côtés est utilisé pour effectuer la transmission. Celui-ci utilise des trames de 8 ou 9 bits, et il faudra donc 2 trames d'UART pour transmettre une trame de FSK. L'émetteur et le récepteur sera donc logiquement des machines à état séquentielles. Plutôt que de simplement reconstituer la trame de 10 bits originale, on choisit que le récepteur renvoie directement d'une part les 2 bits de commande et d'autre part les 8 bits d'arguments contenus dans une transmission.

Les parties récepteur et émetteur vont être abordées en parallèle dans la suite, puisqu'elles sont fortement liées. Tout d'abord, la configuration des modules UART est examinée¹, ensuite, l'émetteur et le récepteur vont être construits.

¹La plupart des paramètres doivent forcément être les mêmes des deux côtés de la transmission pour que celle-ci soit possible.

4.2 Configuration des modules UART

4.2.1 Paramètres communs

Pour que la communication soit possible, les deux UART doivent être en accord sur quatre paramètres : le Baud Rate, la polarité des ports TX et RX, le format de trame, et enfin le protocole d'envoi de trame. Ces paramètres sont fixés dans la fonction `initUart`, qui est donc en grande partie commune aux deux microcontrôleurs.

Le Baud Rate est fixé à 9600. Cette valeur rend la transmission d'une instruction (deux trames) pratiquement instantanée par rapport aux autres constantes de temps en présence ($f_{regul} = 100\text{ Hz}$, $f_{symbol} = 10\text{ Hz}$), tout en étant suffisamment basse pour permettre d'utiliser l'horloge de l'UART en 16X speed mode, ce qui diminue la probabilité d'erreur car chaque bit est alors échantillonné trois fois au lieu d'une.

Les trames sont choisies contenant 8 bits utiles avec un bit de parité et terminées par 1 stop bit. Passer à 9 bites utiles et sans bit de parité ne présente pas d'intérêt puisqu'il faut toujours envoyer deux trames pour transmettre une instruction complète (10 bits). La détection rudimentaire d'erreur fournie par le bit de parité est donc légèrement préférable. La polarité n'a elle aucune importance, du moment qu'elle est identique de part et d'autre d'une ligne TX(μC_1)-RX(μC_2).

Vu les très faibles contraintes sur l'UART, le risque d'erreur et la probabilité que les FIFO de réception et transmission se remplissent sont pratiquement nuls, on peut donc se contenter du protocole le plus simple avec seulement deux fils RX et TX et pas de flow control physique. Au niveau des pattes, RX est d'une part simplement lié à la patte reprogrammable choisie pour RX dans le registre `RPINR18bits.U1RXR` et celle-ci est mise en input, et d'autre part, la patte reprogrammable choisie pour TX est liée à TX dans le registre `RPORXbits.RPXR`. Enfin, les branchement croisés RX-TX sont effectués. Les paramètres de l'UART et les branchements sont validés dans la section suivante.

4.2.2 Validation de la partie «physique» de l'UART

A partir de cette configuration essentielle, le fonctionnement de l'UART entre les deux microcontrôleurs est vérifié en envoyant une suite de caractères avec un microcontrôleur et en vérifiant que ceux-ci sont bien reçus par l'autre microcontrôleur, et ce à une fréquence suffisamment lente pour pouvoir utiliser le debugger de MPLab. L'émetteur de test utilise l'interruption d'un timer 32 bit¹ pour écrire un caractère dans le registre `U1TXREG` et incrémenter ce caractère. On place un breakpoint dans l'interruption `U1RXREG`

¹Un timer 32 bits est utilisé afin d'envoyer des caractères suffisamment lentement pour qu'une fois le breakpoint déclenché, nous ayons le temps de vérifier la valeur du caractère et de relancer le code côté récepteur avant qu'une nouvelle trame ne soit envoyée.

pour vérifier que la trame est bien reçue et qu'il n'y a pas eu d'erreur de parité ou de formatage.

La communication a été testée dans les deux sens et la partie «physique» de l'UART a ainsi été validée. Il reste maintenant à implémenter le software autour de ce bloc pour transmettre des trames FSK de 10 bits du microcontrôleur communication vers le microcontrôleur propulsion. Tout d'abord, les modes d'interruption pour la réception et l'émission sont adaptés pour chaque microcontrôleur en fonction du rôle que celui-ci joue dans la communication.

4.2.3 Choix des modes d'interruption

La communication est clairement asymétrique : le microcontrôleur communication transmet les ordres au microcontrôleur propulsion, qui ne répond presque jamais, comme il est expliqué dans la suite. Il est donc logique que les modes d'interruptions soient légèrement différents pour les deux microcontrôleurs.

Réception

Le mode d'interruption pour la réception est tout de même identique pour les deux microcontrôleurs. Une interruption est déclenchée dès qu'une nouvelle trame peut être lue. La routine de réception d'une trame d'UART vérifie simplement qu'il n'y a pas d'erreur de formatage ou de parité puis appelle la fonction `handleReceived`, qui déclenche le traitement de la trame reçue si elle est correcte, ou la sort simplement de la FIFO si elle est incorrecte. Bien évidemment, la fonction `handleReceived` contient toute la logique et varie entre les deux microcontrôleurs.

Émission

Les deux microcontrôleurs envoient finalement assez peu de trames. Pour cette raison, l'émetteur UART ne déclenche pas par défaut d'interruption liée au statut de sa FIFO. La différence entre les deux microcontrôleurs réside dans le fait que le microcontrôleur propulsion n'envoie jamais de trames successives mais le microcontrôleur communication bien. Pour ce dernier, nous avons choisi d'exploiter une routine d'interruption pour envoyer les trames.

Du côté propulsion, l'interruption d'émission est donc simplement désactivée. La configuration de l'UART du microcontrôleur communication inhibe au départ l'interruption d'émission, mais comme elle sera activée durant le fonctionnement, il faut tout de même configurer le mode d'interruption. On choisit de déclencher une interruption dès qu'une trame peut-être écrite dans la FIFO, ce qui correspond au mode `0b00`. La routine d'émission contient en

réalité la machine d'état de l'émetteur, et est donc présentée dans la section suivante.

4.3 Programmation des émetteurs et récepteurs

Dans cette section, le format de deux trames d'UART pour représenter une trame de FSK va être présenté, et les récepteurs et émetteurs des deux microcontrôleurs vont être construits à partir de ce format.

4.3.1 Division d'une trame de FSK

Pour rappel le format des trames à la sortie du démodulateur FSK est donné à la figure 4.1. C'est cette trame qui doit être divisée en deux trames

Données	
Ordre	Paramètre
2 bits	8 bits

FIG. 4.1 : Format d'une trame à la sortie de `fskDetector`.

de 8 bits. De plus, le récepteur doit pouvoir faire la différence entre les trames «première partie» et les trames «deuxième partie», afin de pouvoir détecter certaines erreurs (partie manquante ou répétition) et pouvoir correctement reconstituer l'instruction reçue. Dans la suite, les trames «première partie» seront notées T1 et les trames «deuxième partie» T2.

L'ordre ne peut prendre que les valeurs `0b00`, `0b10` ou `0b01`. On choisit donc que la quatrième possibilité, `0b11` constitue les deux premiers bits de T1 afin, de distinguer celle-ci de T2, dont les deux premiers bits sont l'ordre¹. Les 8 bits de paramètre restants sont répartis entre T1 et T2. Il reste donc deux bits inutilisés dans chaque trame, qu'on met simplement à 0. Tout ceci est résumé dans la figure 4.2.

A partir de ce format de trame, les émetteurs et récepteurs vont maintenant être construits en suivant l'émission d'une commande et en examinant tous les scénarios possibles.

4.3.2 Émetteur communication

L'entrée du bloc UART dans son entier est la fonction `sendCommand(int newCommand)`. Comme des erreurs sont envisageables, le récepteur peut demander que la commande soit répétée. Pour cette raison, `newCommand` est stockée de manière persistante avant d'être envoyée. Comme annoncé plus haut,

¹Voir la section 4.3.3 sur le récepteur propulsion pour voir pourquoi l'ordre est envoyé dans la deuxième trame.

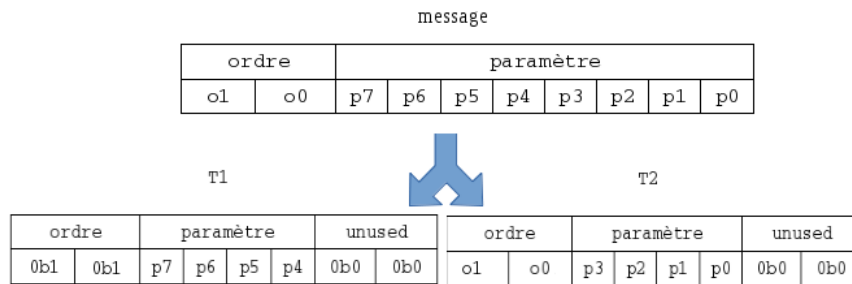


FIG. 4.2 : Division du message en deux trames UART.

les interruptions d'émissions sont utilisées pour envoyer les deux trames, et l'émetteur est une machine d'état séquentielle.

Lorsqu'une commande doit être envoyée, l'interruption d'émission est activée, et l'état de l'émetteur mis à zéro, ce qui correspond à aucune trame déjà envoyée. Ensuite, dans la routine d'émission, T1 ou T2 est construite selon l'état, puis envoyée, et l'état est incrémenté. Après l'envoi de T2, l'interruption est désactivée jusqu'à ce qu'une nouvelle commande doive être envoyée, et ainsi de suite. Ceci est exécuté par la routine d'émission.

Le récepteur propulsion est calqué sur cet émetteur et est présenté dans la section suivante.

4.3.3 Récepteur propulsion

Le récepteur propulsion est donc lui aussi une machine d'état séquentielle à deux états.

A l'état 0, une T1 est attendue. Si c'est bien une T1 qui est reçue, alors l'état passe à 1 et $p<7-4>$ contenus dans la trame sont stockés. Sinon, il y a eu une erreur, l'état reste à 0 et l'UART propulsion demande que le message soit répété.

A l'état 1, une T2 est attendue. Si une T2 est reçue, alors le message a été entièrement reçu, l'état retourne à zéro et le récepteur retourne séparément $o<1-0>$ et $p<7-0>$ à partir de T2¹ et de $p<7-4>$ en mémoire. Sinon, il y a eu une erreur, l'état retourne aussi à zéro et l'UART propulsion demande que le message entier soit répété. Ceci est exécuté par la fonction `handleReceived` qui appelle `handleParam1` ou `handleParam2` selon le type de trame reçue.

D'autres possibilités d'erreurs ont été vues plus haut : mauvaise parité (détecté par l'UART lui-même) ou mauvais formatage (détecté par l'UART ou le récepteur si LSB et LSB+1 d'une trame sont non nuls). Le mécanisme est systématiquement le même : l'état du récepteur retourne à zéro et la répétition du message est demandée par la fonction `askRepeat`.

¹On voit donc ici pourquoi il est plus intéressant d'envoyer l'ordre dans T2 : cela permet de ne pas devoir le stocker à l'état 0.

La demande de répétition du message est la seule chose que l'UART propulsion émet. L'émetteur propulsion est donc très simple. Il est présenté dans la section suivante.

4.3.4 Émetteur propulsion

L'émetteur propulsion n'envoie que des messages de une trame, et ce relativement rarement. On peut donc se contenter de vérifier (pour la forme) que l'on peut écrire dans la FIFO et d'y écrire la trame `0x01` qui demande le renvoi de la commande. Il s'agit donc aussi de la seule trame que le récepteur communication peut doit gérer. Ce récepteur est donc lui aussi très simple. Il est présenté dans la section suivante.

4.3.5 Récepteur communication

Si l'UART communication reçoit une trame `0x01`, il faut donc renvoyer la commande actuelle. Pour cela `sendCommand(int newCommand)` est légèrement modifiée. Si `newCommand` est `0xFFFF` (pas un message valide), alors la commande gardée en mémoire n'est pas écrasée, et l'envoi déclenché ensuite renvoie donc la commande précédente. Ceci est fait par un simple test au début de la fonction `sendCommand`

Le cas où la trame reçue côté communication n'est pas valide n'est pas géré : la transmission de la commande, s'il y avait bien une commande à transmettre au départ, est abandonnée. Ce cas ne s'est jamais présenté. Tous les scénarios possibles ont donc été examinés et le code présenté implémente donc une communication UART complète et parfaitement fonctionnelle. Ceci va être validé dans la section suivante.

Le traitement de l'ordre et du paramètre doit maintenant être abordé. Ce traitement est appelé dans la fonction `handleParam2` du code source du récepteur propulsion par l'instruction `interpretCommand(command, param);`, et va être présenté dans le chapitre suivant. La transmission de la commande entre les deux microcontrôleurs et l'interprétation de celle-ci seront validée en même temps à la fin du chapitre suivant.

Chapitre 5

Interprétation des ordres

Ce chapitre décrit le bloc qui fait le lien entre les instructions reçues par l'UART propulsion et la régulation des moteurs. Dans le chapitre 3, nous avons vu que les consignes sont générées sur base de trois variables accessibles par les autres blocs : l'accélération, la distance ou l'angle total à parcourir, et la distance ou l'angle à parcourir avant décélération. La régulation doit aussi savoir si le robot est en train de tourner ou d'avancer tout droit pour simplifier le test d'arrivée. Dans la section suivante, ces variables d'état sont rapidement détaillées, et nous verrons ensuite comme celles-ci sont modifiées par les instructions reçues par l'UART.

5.1 Variables d'état externes de la régulation du robot

Les variables sont dédoublées pour les mouvements rectilignes et les rotations. Dans la suite, toutes les variables sont citées, mais le rôle ou l'évolution au cours du temps n'est décrit que pour les variables «rectilignes» afin de ne pas alourdir cette section.

L'accélération et l'accélération angulaire sont fixées par `float acceleration` et `float angularAcceleration`. La régulation est déjà responsable, lors de l'exécution d'une commande, de d'abord mettre l'accélération à zéro lorsque la vitesse nominale est atteinte, puis de fixer une décélération lorsque la distance avant décélération est atteinte, et enfin de remettre une dernière fois l'accélération à zéro lorsque la distance totale est atteinte. Le rôle du bloc traité ici se borne donc à fixer une valeur non-nulle lors de la réception d'une commande pour démarrer.

La distance ou l'angle total à parcourir sont fixés par `char goalDistance` et `float goalTheta`. Seul le bloc couvert dans ce chapitre-ci modifie cette valeur. La distance ou l'angle à parcourir avant décélération sont fixés par `char decelerationDistance` et `float decelerationTheta`. Seul ce bloc modifie cette valeur, qui est soit une valeur par défaut si la dis-

tance totale à parcourir est suffisante, soit un tiers de la distance totale si $\frac{\text{goalDistance}}{3} < \text{DFLT_DECELERATION_DST}$.

Table des figures

2.1	Circuit du microphone	3
4.1	Format d'une trame à la sortie de <code>fskDetector</code>	9
4.2	Division du message en deux trames UART.	10

Table des codes source