Stage Industriel

Rapport de stage

Julien CHEVRIER

Table des matières

1 Remerciements **4**

2 Introduction **5**

3 Le laboratoire de recherche en robotique spatiale de l'université de Tohoku **5**
3.1 L'université de Tohoku 5
3.2 Le laboratoire de recherche en robotique spatiale 5
3.2.1 Présentation du laboratoire 5
3.2.2 Recherches en cours au sein du laboratoire 6

4 Objectifs de la mission technique **7**
4.1 Sujet de stage 7
4.2 Cahier des charges fonctionnel 12
4.3 Critères de validation du projet 13
4.4 Moyens mis a disposition 13
4.5 Planification du projet 14

5 Réalisation du projet **14**
5.1 Analyse du cahier des charges 14
5.2 Conception générale 15
5.3 Conception détaillée 17
5.3.1 Mesure de la force en bout des pattes 18
5.3.2 Module de contrôle des pattes hybrides 20
5.3.3 Module de contrôle principal 25
5.3.4 Module de contrôle en balance 30
5.4 Plan de test de conformité 31
5.5 Validation 32
5.6 Résultats 32

6 Conclusion **32**

7 Glossaire **34**

8 Bibliographie **34**

9 Annexes **35**
9.1 Main et Interruptions 35
9.2 ADCs 46
9.2.1 adc.h 46

9.2.2 adc.c 47
9.3 UARTs 48
9.3.1 uart_def.h 48
9.3.2 uart_def.c 48
9.4 Utilitaires pour le contrôle des servomoteurs Dynamixel 49
9.4.1 dynamixel.h 49
9.4.2 dynamixel.c 50

1 Remerciements

Je tiens à remercier messieurs P. Gloess et S. Azzopardi pour avoir permis l'échange avec l'université de Tohoku en établissant une convention d'échange.

Je remercie également le professeur K. Yoshida pour avoir également participé à la signature de la convention d'échange du côté japonais, et pour m'avoir accepté dans son laboratoire.

Je remercie également le docteur E. Rohmer, pour m'avoir pris comme stagiaire sur son robot, LEON, et pour m'avoir aidé pour toutes les démarches administratives sur place.

2 Introduction

Mes objectifs pour ce stage de deuxième année étaient à la fois techniques, culturels et linguistiques. Le premier d'eux était d'observer les méthodes de travail au Japon, ainsi que de découvrir les technologies utilisées. Ce stage me permettait également de découvrir le Japon, terre de contrastes, et assez méconnu. J'avais également pour but d'améliorer ma pratique de l'anglais et m'améliorer également en japonais.

J'ai choisi l'université de Tohoku pour son laboratoire de recherche en robotique. J'ai choisi le sujet proposé portant sur un robot hexapode, car cela se rapprochait d'un de mes sujets développements personnels en cours.

Dans un premier temps, l'université de Tohoku ainsi que le laboratoire de recherche en robotique spatiale seront présentés. Ensuite, nous présenterons les objectifs de la mission et nous terminerons par détailler la réalisation du projet.

3 Le laboratoire de recherche en robotique spatiale de l'université de Tohoku

3.1 L'université de Tohoku

L'université de Tohoku est la plus grande université de la région nord-est du Japon. Celle-ci est située à Sendai.

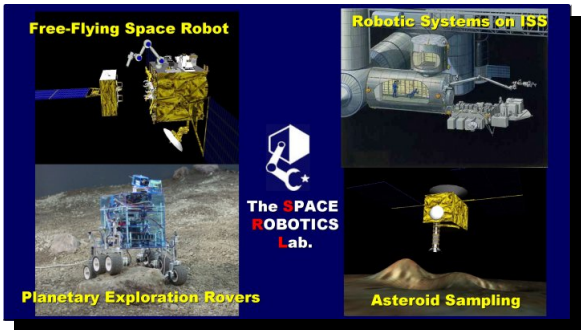
L'université est divisée en plusieurs campus, chacun ayant sa spécialité. Un de ces campus est dédié à l'ingénierie. Dans ce campus, chaque étudiant est affilié à un laboratoire, où il exerce des recherches pour celui-ci. Les étudiants ont peu de cours : environ 10 heures par semaine. Le travail en laboratoire est en quelque sorte des travaux pratiques, où ils appliquent leurs connaissances.

3.2 Le laboratoire de recherche en robotique spatiale

3.2.1 Présentation du laboratoire

Le laboratoire de recherche en robotique spatiale se situe dans le bâtiment d'ingénierie mécanique de l'université.

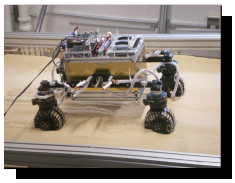
Ce laboratoire est dirigé par le professeur Kazuya Yoshida et assisté par le professeur Keiji Nagatani.



Le laboratoire a déjà participé à plusieurs projets liés avec la JAXA (agence aérospatiale japonaise), notamment sur la mission Hayabusa, mission visant à récupérer de la roche provenant d'un astéroïde. Pour cette mission, plusieurs systèmes de récupération d'échantillons ont été imaginés et testés dans ce laboratoire.

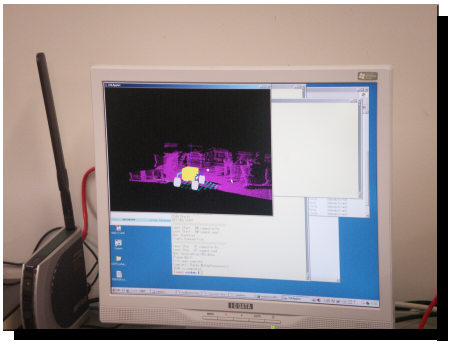
3.2.2 Recherches en cours au sein du laboratoire

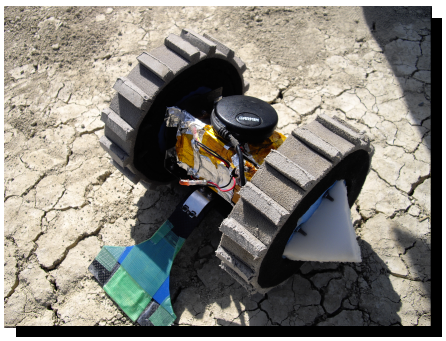
Dans le laboratoire, plusieurs systèmes appliqués à des robots sont en cours d'étude.



Il y a, par exemple, le robot El Dorado (photo ci-contre), sur lequel sont effectués des tests sur du sable pour tester la friction des roues sur celui-ci, et effectuer des corrections de trajectoire en cas de glissement.

Le laboratoire possède également une version plus grande de ce robot, équipé de télémètres LASER, servant à réaliser un plan en trois dimensions de son environnement.





Une équipe d'étudiants du laboratoire participe à la compétition ARLISS (A Rocket Launch for International Student Satellites). Le but de cette compétition est d'envoyer un robot à 4000 mètres et faire en sorte que ce robot rejoigne un point défini. Simon Desfarges, étudiant à l'ENSEIRB, a participé à cette compétition.

4 Objectifs de la mission technique

4.1 Sujet de stage

Le stage a été effectué sur un robot hexapode hybride nommé LEON, acronyme de Lunar Exploration Omnidirectional Netbot (Fig. 1). Ce robot est la réalisation et le sujet de recherche du Docteur E. Rohmer.
Ce robot hybride est composé de six pattes à trois degrés de liberté dont deux pouvant se replier sur elles-mêmes pour former des roues. Ce robot possède ainsi deux modes de déplacement :

- Un mode "pattes", lui permettant de marcher et de franchir divers obstacles
- Un mode "roues", lui permettant d'aller plus vite et d'économiser de l'énergie.

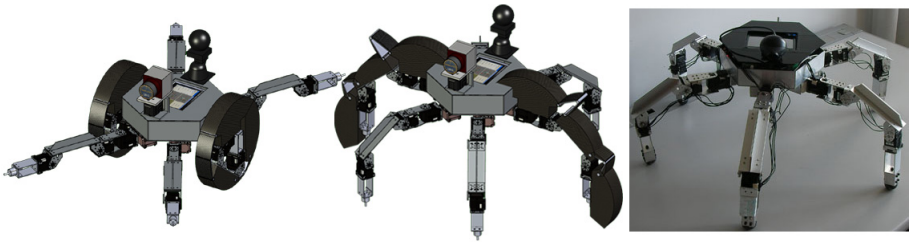


FIG. 1 – A gauche : mode "roues", au centre : mode "pattes", à droite : mécanique réalisée à ce jour

Les actionneurs utilisés (Fig. 2) sont des servomoteurs Dynamixel. Ce sont des servomoteurs à commande numérique via un bus de données RS485.

Fig. 2 – Servomoteur Dynamixel DX117

Ces servomoteurs peuvent fonctionner de deux manières :
- En consigne d'angle, comme un servomoteur classique. Le servomoteur cherche à atteindre la consigne d'angle donnée.
- En rotation continue. La vitesse est le paramètre de contrôle de ce mode.

Ces servomoteurs ont plusieurs paramètres qui peuvent être définis :
- Vitesse de rotation
- Couple
- Écart maximum avec la consigne (en mode consigne d'angle)
- Vitesse de la communication RS485
- etc...

Les ordres donnés à ces servomoteurs se font par un protocole particulier (Fig. 3). Ce protocole est nommé "protocole Dynamixel" dans la suite du rapport. Un paquet Dynamixel comporte les éléments suivants :
- 2 octets ayant pour valeur 0xFF, indiquant le début d'une trame
- l'ID de l'élément concerné
- la longueur du paquet
- l'instruction envoyée
- divers paramètres liés à l'instruction
- un checksum de contrôle

Fig. 3 – Exemple de paquet Dynamixel

LEON est un robot téléopéré. Il embarque un ordinateur Vaio pour le contrôle bas niveau du robot (contrôle des actionneurs, et synchronisation des phases de mouvement des pattes). Le Vaio contrôle donc les mouvements du robot. L'ordinateur Vaio est connecté par liaison sans-fil à une plateforme de contrôle haut niveau du robot (direction et vitesse du robot). Cette plateforme est basée sur la librairie ODE (Open Dynamics Engine), permettant d'avoir une simulation dynamique en temps réel du robot en plus du contrôle de celui-ci. Cette plateforme a pour nom ERODE (ER, étant les initiales de l'auteur).
Une architecture simple (Fig. 4) a été déjà réalisée permettant un fonctionnement en mode "pattes" : le Vaio communique directement par liaison RS485 par l'intermédiaire d'un module "USB2Dynamixel". Cependant, cette architecture électronique ne permet pas de faire fonctionner ce robot en mode roue : en effet, si les pattes hybrides se mettaient à tourner, cela enroulerait les câbles autour des axes de ces pattes. L'architecture décrite en Fig. 4 se compose de quatre nivaux :

- Navigation level : Son objectif est de générer un modèle 3D de l'environnement du robot par l intermédiaires de laser range finder, et d'un retour visuel par une caméra pour aider l'opérateur à naviguer.
- Body level : Il est la colonne vertébrale et le cerveau du robot. L'ordinateur principal est un Vaio UX91 qui fait office d interface de communication entre les capteurs/actionneurs et la plate-forme de télérobotique ERode. Il s'occupe aussi du contrôle moyen et bas niveau des actionneurs , et rapporte du status des capteurs embarqués (accéléromètres, capteurs de force, configuration angulaire des servomoteurs...). On y trouve aussi le module "USB2Dynamixel" pour la communication entre le Vaio et les servomoteurs.
- Leg level : C'est le niveau dans lequel se trouvent les servomoteurs. Un micro-contrôleur SH4 est utilisé pour lire les capteurs de force au bout des pattes.
- Contact sensor level : On trouve dans ce niveau les capteurs de force au bout des pattes.

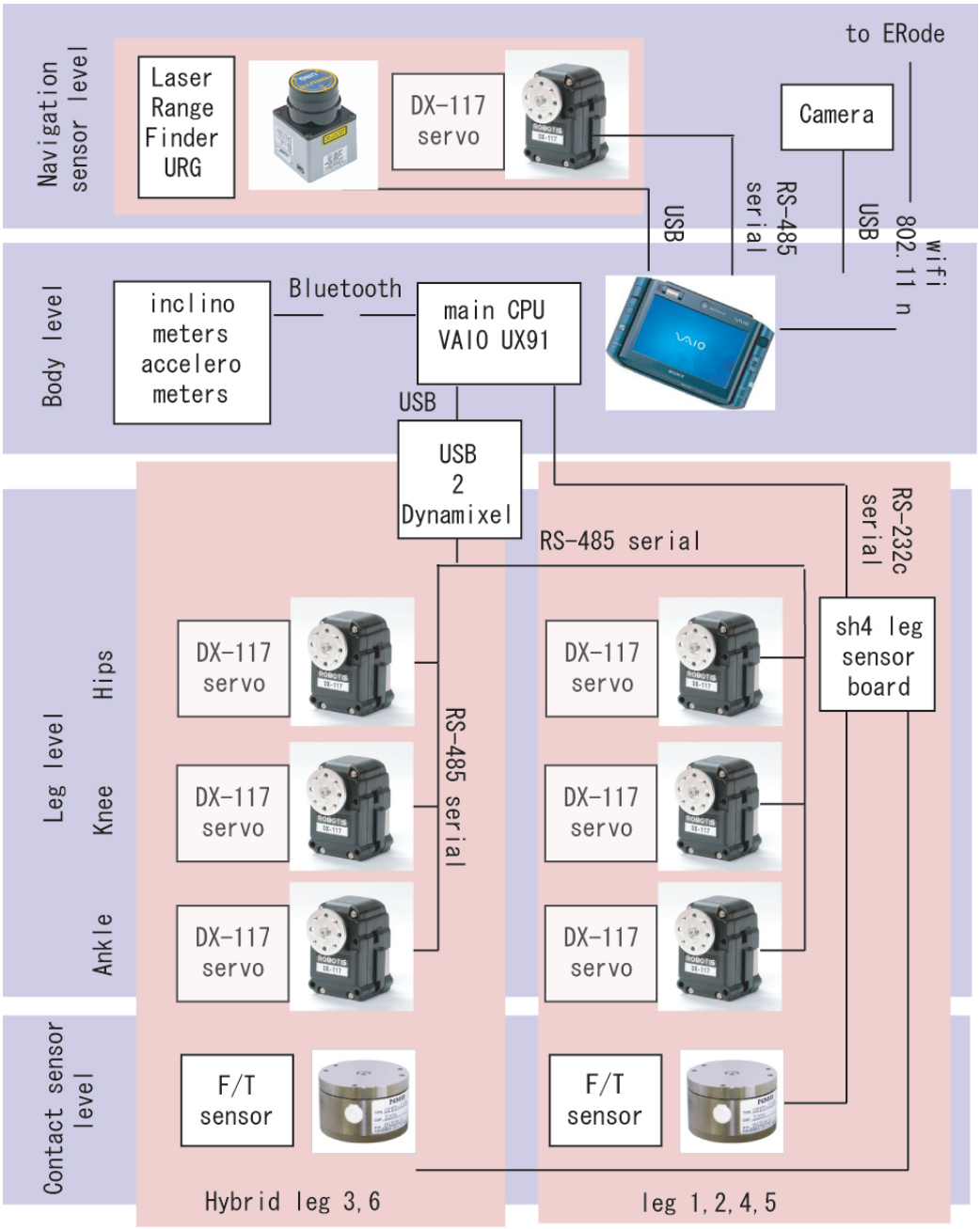


FIG. 4 – Architecture de base

Plusieurs solutions avaient déjà été envisagées pour pouvoir avoir un mode “roues” fonctionnel :

– La première consistait en l’utilisation de contacts tournants (“Slip rings”)

– La deuxième imposait une séparation complète du corps et des pattes hybrides au niveau électrique, faisant ainsi des pattes hybrides des système autonomes communicant avec le corps par des moyens de communication sans fil.

La deuxième solution avait été retenue pour des raisons de budget. L'architecture globale change peu par rapport à l'architecture existante. Des cartes électroniques embarquées dans les pattes hybrides ont été rajoutées pour pouvoir répéter les paquets Dynamixel envoyé par le Vaio sur les liaisons RS485 et pour pouvoir gérer des capteurs embarqués dans ces pattes. Tout cela a conduit à l'idée d'architecture de la Fig. 5. L'architecture décrite en Fig. 5 est basée sur celle présentée en Fig. 4. Les différences concernent le contenu des niveaux :

– Body level : Le module USB2Dynamixel ne relie plus le Vaio qu'aux servomoteurs des pattes non hybrides et aux deux servomoteurs actionnant les roues. Pour les roues hybrides, une connexion Bluetooth permet de retirer les fils qui s'enrouleraient autour de l'arbre de transmission.
– Leg level : Le micro-contrôleur SH4 est utilisé seulement pour lire les capteurs de force au bout des pattes non hybrides. Pour les pattes hybrides, c'est un micro-contrôleur H8 embarqué dans chaque pattes, qui s'occupe de cette tâche et qui assure la communication sans fil via Bluetooth avec le Vaio.

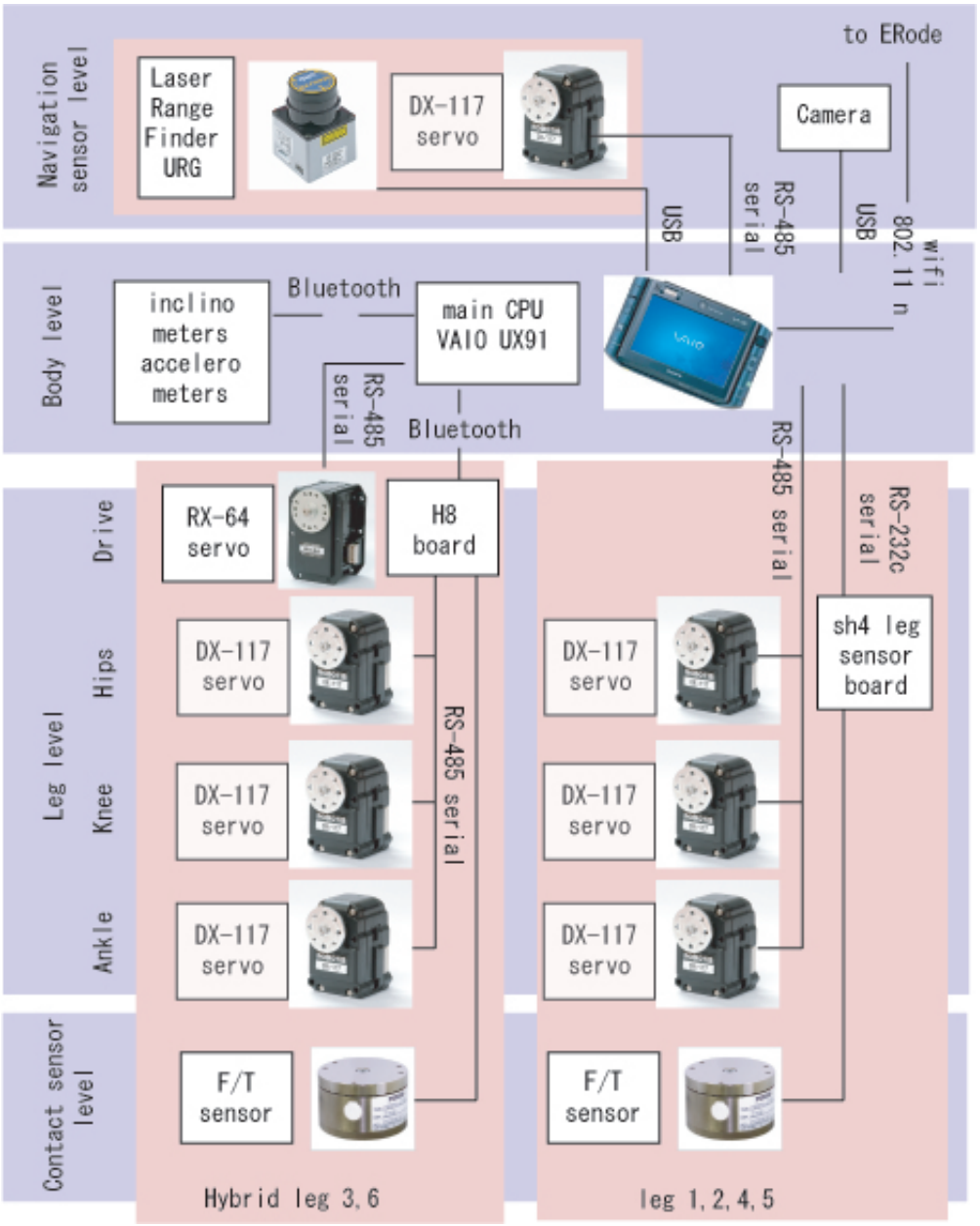


FIG. 5 – Architecture proposée

.

4.2 Cahier des charges fonctionnel

Le but du stage est la réalisation/amélioration du système présenté précédemment. Pour cela, il faudra tester divers types de connexions sans-fil et

choisir la plus adaptée. Il faudra ensuite réaliser un système embarqué dans chacune des pattes hybrides permettant le contrôle de celles-ci par liaison sans-fil et pouvant gérer l'acquisition de données de capteurs intégrés à ces pattes. Il faudra ensuite réaliser l'architecture électronique présentée précédemment.
Dans un second temps, selon le temps disponible, une solution de contrôle en balance devra être imaginée permettant au robot de se maintenir en équilibre sur ses roues. Ce module est optionnel.
Les cartes électroniques réalisées devront être les plus petites possibles. Le système réalisé devra être cohérent avec l'existant. Seuls des micro-contrôleurs H8 ou SH2 de Renesas pourront être utilisés pour permettre une maintenabilité du projet au sein du laboratoire.
La connectique disponible sur l'ordinateur Vaio étant très limitée, il faut utiliser le moins possible de connexions physique entre le Vaio et les cartes électroniques créées.
La boucle d'asservissement du système étant un peu lente ayant des saut à des temps de réponse élevés, le système créé devra résoudre ce problème.

4.3 Critères de validation du projet

La validation du projet sera faite sur les critères suivants :
- Succès de communication avec les servomoteurs
- Temps de réponse du système minimal
- Accès aux capteurs fonctionnel

4.4 Moyens mis a disposition

Un box du laboratoire m'a été réservé, je disposais ainsi d'un bureau personnel pendant toute la durée du stage.
Au niveau logiciel, je disposais de l'environnement de programmation YellowIDE de Yellowsoft, préconisé par Renesas Technology. Seul bémol, ce logiciel était uniquement disponible en japonais, il m'a fallu donc apprendre certains caractères japonais pour avoir plus de facilité pour l'utiliser.
Pour ce qui est du matériel, le laboratoire disposait de postes de soudure ainsi que des réserves de composants électroniques pour le développement. Cependant, le matériel de soudure était assez minimaliste. Quelques platines de test de micro-contrôleurs étaient également à ma disposition. De plus, je pouvais également demander à commander tout composant/module électronique nécessaire à la réalisation du projet.

4.5 Planification du projet

Le projet n'avait pas de planning particulier au début du stage, il a été définit au fur et à mesure (Fig. 6).

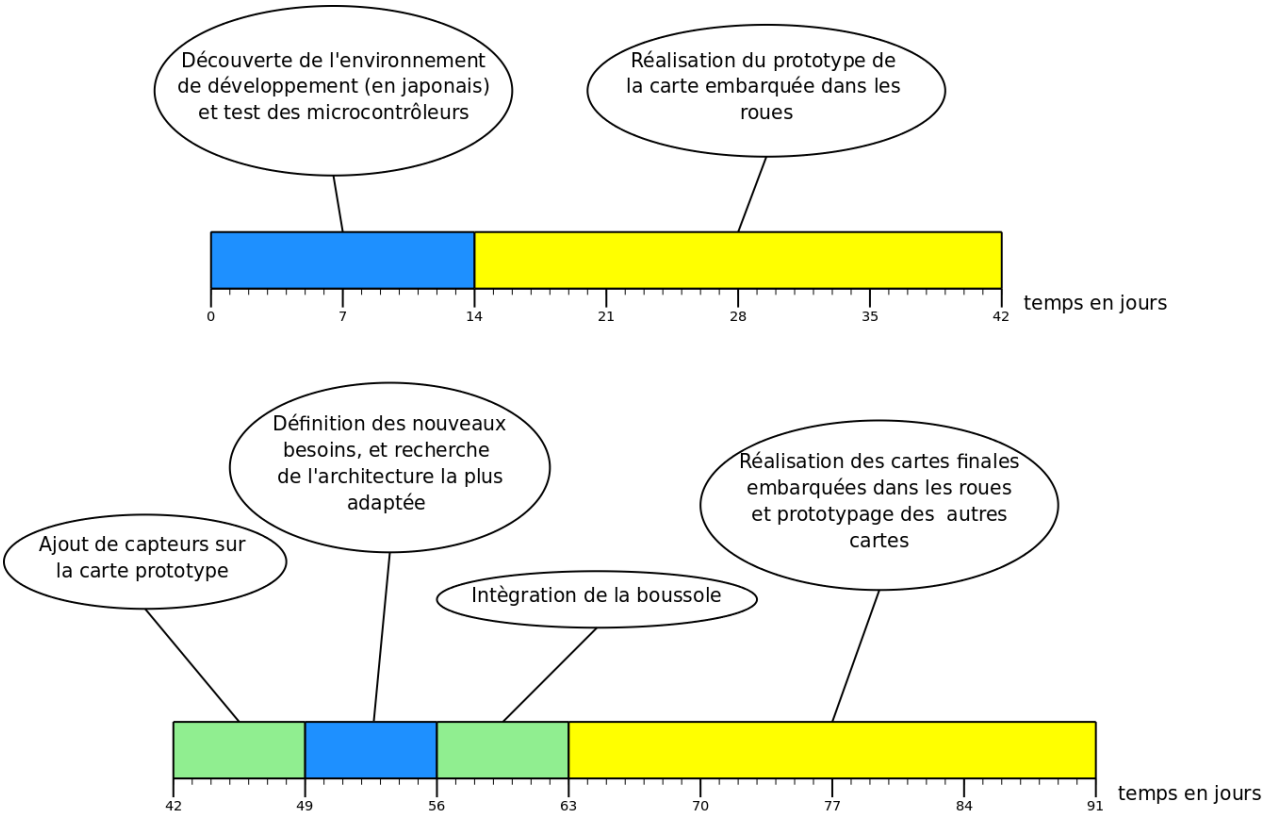


Fig. 6 – Étapes chronologiques du projet

5 Réalisation du projet

5.1 Analyse du cahier des charges

La conception mécanique de LEON implique de faire des cartes électroniques aussi petites que possible, particulièrement dans les roues, où la place pour de l'électronique embarquée est très limitée. De plus, ces dernières doivent être aussi légères que possible pour que les pattes hybrides en mode roue ne soient pas déséquilibrées.
Le module de contrôle en balance étant optionnel, il faut que le système ne dépende pas de lui.
Afin de rester cohérent avec l'existant et de faire une utilisation simple des

cartes, le protocole de communication Dynamixel sera utilisé pour accéder à tous les éléments : servomoteurs, capteurs, contrôle en balance. Cela évite d'avoir à réinventer un protocole de communication entre le Vaio et les différents modules, et cela permet également d'avoir un protocole de communication unique. Chaque élément, qu'il soit module de contrôle en balance, capteur ou servomoteur, aura donc une ID fixe et les cartes créées devront être transparentes du point de vue de la communication du Vaio vers chaque élément.

5.2 Conception générale

L'acquisition de données de capteurs de force se faisant par l'intermédiaire d'une carte électronique, une carte principale gérant des capteurs analogiques et des servomoteurs sera donc créée. De plus cette carte devra gérer une boussole communiquant par bus I2C.

Le contrôle en balance demandé impose la conception d'un carte dédiée à ce système, peu interrompue par des événements extérieur limitant ainsi les communications avec d'autres éléments.

Pour une conception simplifiée et une future maintenance de l'architecture électronique, un micro-contrôleur commun à toutes les carte a été utilisé. Celui-ci devait avoir les spécifications suivantes pour pouvoir satisfaire les besoins de toutes les cartes :

- Deux liaisons UART
- Bus de communication I2C
- Au moins six convertisseurs analogiques/numériques
- Carte la plus petite possible
- Plusieurs ports d'entrées/sorties.

Une seule carte disponible disposait de toutes ces fonctionnalités : c'est une carte de base de micro-contrôleur H8/3687 (Fig. 7), disponible chez Yellowsoft.

Fig. 7 – Carte basée sur micro-contrôleur H8/3687

Cela a mené à l'architecture présentée en figure 8. La figure 8 est une évolution l architecture de la figure 5. Les changements se retrouvent dans les parties suivantes :

- Navigation level : On y trouve maintenant une wiimote connectée en Bluetooth sur le Vaio, utilisée pour fournir un signal déjà filtré des angles de roulis et de tanguage du corps du robot.
- Body level : On y trouve maintenant que des micro-contrôleurs H8 pour des raisons d'homogénéité. La connexion avec les pattes hybrides se fait via un module ZigBee. Le "Leg control Module" est l'interface de communication entre le Vaio et les pattes hybrides. Il transmet les paquets au format Dynamixel en transit entre le Vaio et les servomoteurs et empaquette dans ce format les données des capteurs (boussole, accéléromètres, et capteurs de pression) en les rendant accessible par une ID propre comme s'ils étaient des servomoteurs Dynamixel. Ce module communique avec le balance module optionnel et la boussole via un bus I2C. Le Balance module est dédié au contrôle en balance du robot en mode roues. Il rends accessible les données du gyroscope et des accéléromètres au Vaio, en passant par le legs control module.
- Wheel / Leg level : le Sh2 a été remplacé par un H8/3687 toujours dans un souci d'homogénéité matérielle. La communication avec le Vaio se fait via un module ZigBee.
- Contact sensor level : les capteurs de force ont été remplacés par des FlexiForce moins encombrants.

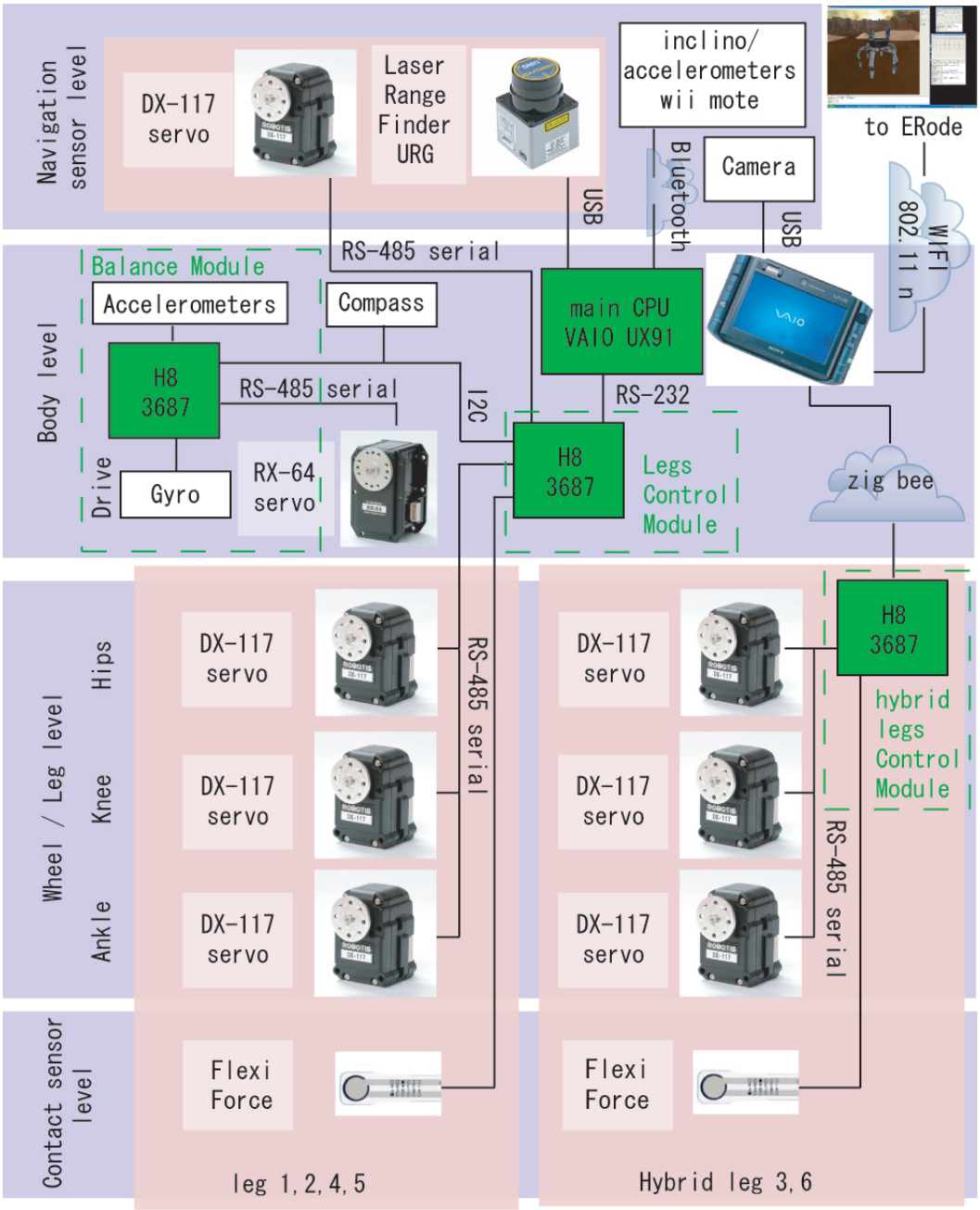


FIG. 8 – Nouvelle architecture

5.3 Conception détaillée

Dans un premier temps, la méthode de mesure en bout des pattes sera présentée, car présente dans deux des trois modules, suivie de la conception

de chacun des modules.

5.3.1 Mesure de la force en bout des pattes

Pour la mesure de la force en bout des pattes de LEON, des capteurs de force ont été utilisés. Ces capteurs sont des Flexiforce de Tekscan (Fig. 9.

Fig. 9 – Capteur de pression

Ces capteurs sont des résistances variables, dont la conductance varie quasiment proportionnellement à la force appliquée sur le capteur comme le montre le graphique de la figure 10).

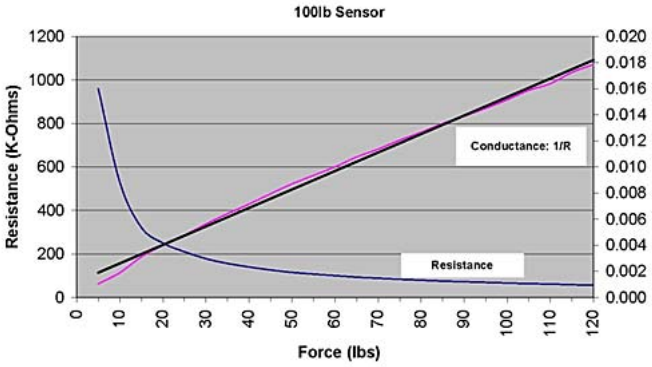


Fig. 10 – Caractéristique du capteur

Sur le site du constructeur, le circuit de la figure 11 est recommandé pour l'utilisation de ces capteurs.

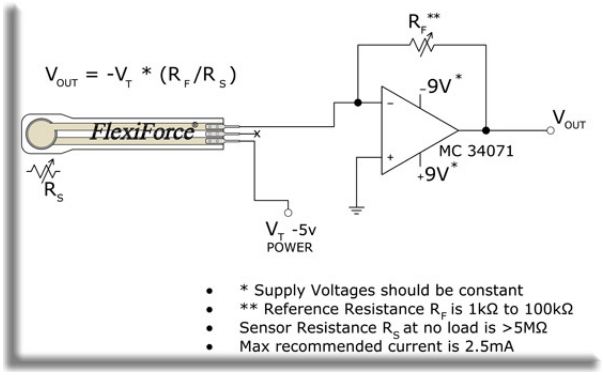


Fig. 11 – Circuit recommandé

Ce circuit est particulièrement simple et à l'avantage de renvoyer un signal proportionnel à la force exercée sur le capteur. Cependant, ne disposant que d'une seule alimentation asymétrique 0V/5V dans le robot, ce circuit a été adapté de façon à être utilisé avec cette alimentation, comme montré sur la figure 12.

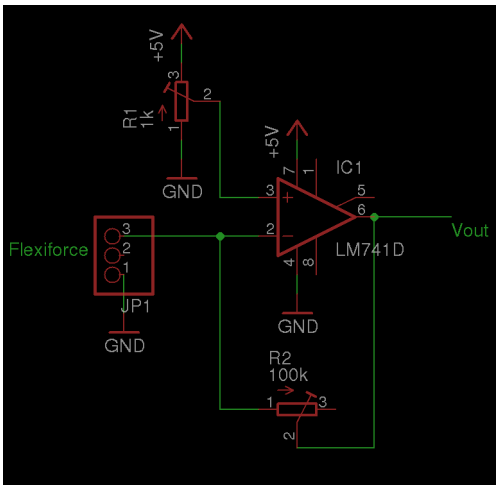


Fig. 12 – Circuit utilisé

Cela permet de plus, de régler les seuils de mesure. Le signal de sortie de ce circuit est lu par les convertisseurs analogiques/numériques des microcontrôleurs H8, permettant ainsi une mesure simple et assez fiable. Ce circuit est utilisé dans le module de contrôle des pattes hybrides et le module de contrôle principal.

5.3.2 Module de contrôle des pattes hybrides

Le module de contrôle des pattes hybrides (Fig. 13) est embarqué dans chacune de celles-ci et est contrôlé par liaison sans-fil. Ce module embarque une connectique RS485 pour communiquer avec les servomoteurs Dynamixel, ainsi qu'un capteur de force.
Il y a au total deux modules : un par patte hybride.

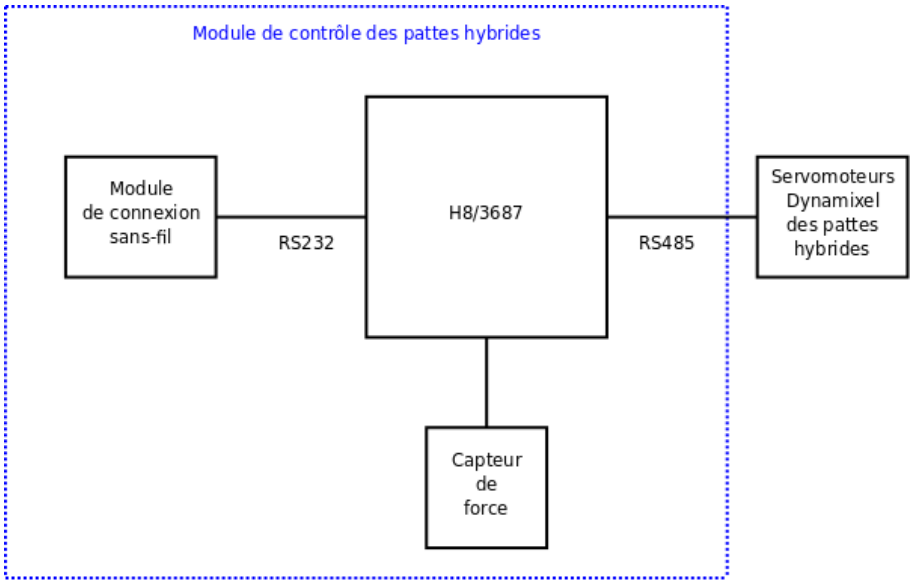


FIG. 13 – Module embarqué dans chaque patte hybride

Choix du module de communication sans-fil

Deux modules de communication sans-fil étaient disponibles :

- un module de liaison RS232 via Bluetooth
- un module de liaison RS232 via ZigBee

Chacun de ces modules a ses avantages et ses inconvénients :

	Module ZigBee	Module Bluetooth
Coût	-	+
Nombre de modules nécessaires	4 (2 connections point-à-point par patte)	2 (1 par patte + utilisation du Bluetooth du Vaio)
Ports RS232 nécessaires sur le Vaio	2	0 (2 virtuels par liaison Bluetooth)
Vitesse de communication RS232	+	-

Bien que le module Bluetooth soit moins cher que le module ZigBee (Fig. 14), la solution par liaison ZigBee a été retenue, car la vitesse de communication maximale pour la partie RS232 est deux fois plus grande que celle du module Bluetooth.

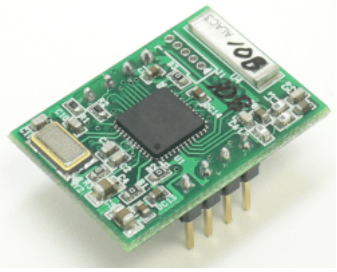


FIG. 14 – Module ZigBee

Réalisation de la communication RS485

Pour réaliser le passage de la communication UART à RS485, un level-shifter MAX485 a été utilisé. Au niveau du micro-contrôleur, les deux pins d'émission et de réception d'un des deux UARTs disponibles étaient nécessaires. Une pin d'entrée/sortie a été également utilisée pour gérer la direction de la liaison RS485 (qui est une liaison half-duplex). Le schéma du montage est présenté sur la figure 15.

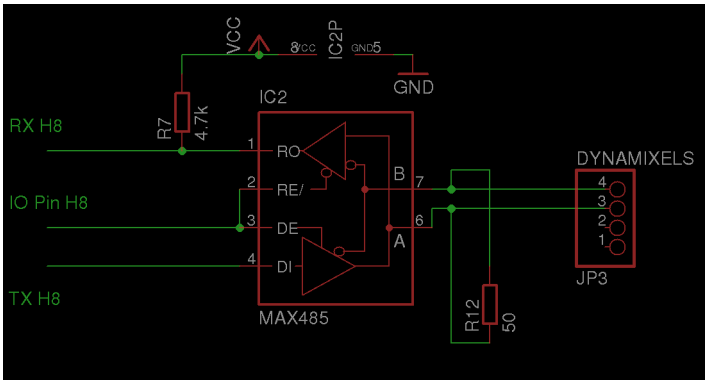


FIG. 15 – Schéma de montage MAX485

Choix de l'alimentation

Les servomoteurs étant déjà alimentés par une batterie, et pour éviter de rajouter une seconde batterie dans chaque patte, l'alimentation du module se fait sur cette batterie, au travers d'un régulateur de tension.

Programmation de la carte

Cette carte étant de susceptible d'avoir à recevoir des données à la fois du Vaio et des servomoteurs, un traitement multi-tâches a été envisagé, permettant ainsi de traiter ces informations en même temps. Une programmation du système avec traitement par interruptions a été réalisé. Le système peut donc traiter en même temps :

- Réception de données en provenance du Vaio
- Émission de données vers le Vaio
- Émission de données vers les servomoteurs Dynamixel ou réception de données en provenance de ceux-ci
- Acquisition de données du capteur de force

Algorithmes du programme de la carte

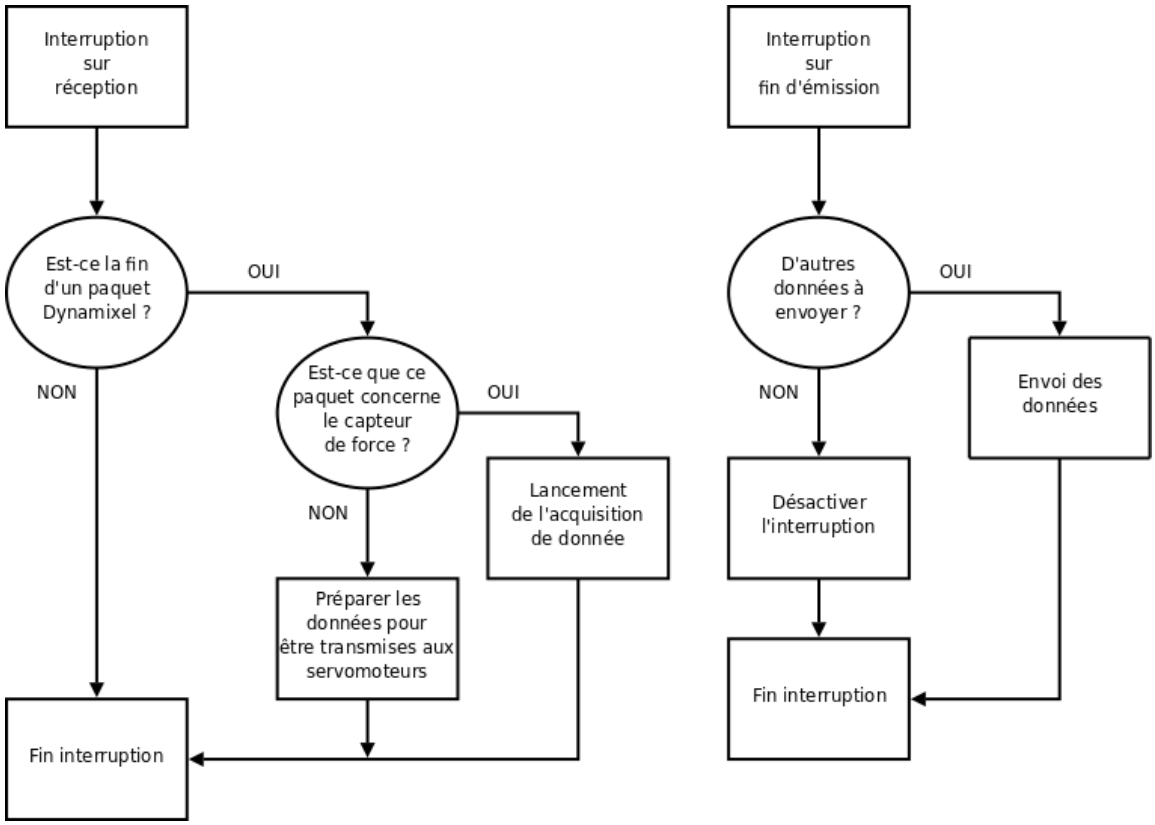


Fig. 16 – Différentes interruptions liée à la communication avec le Vaio

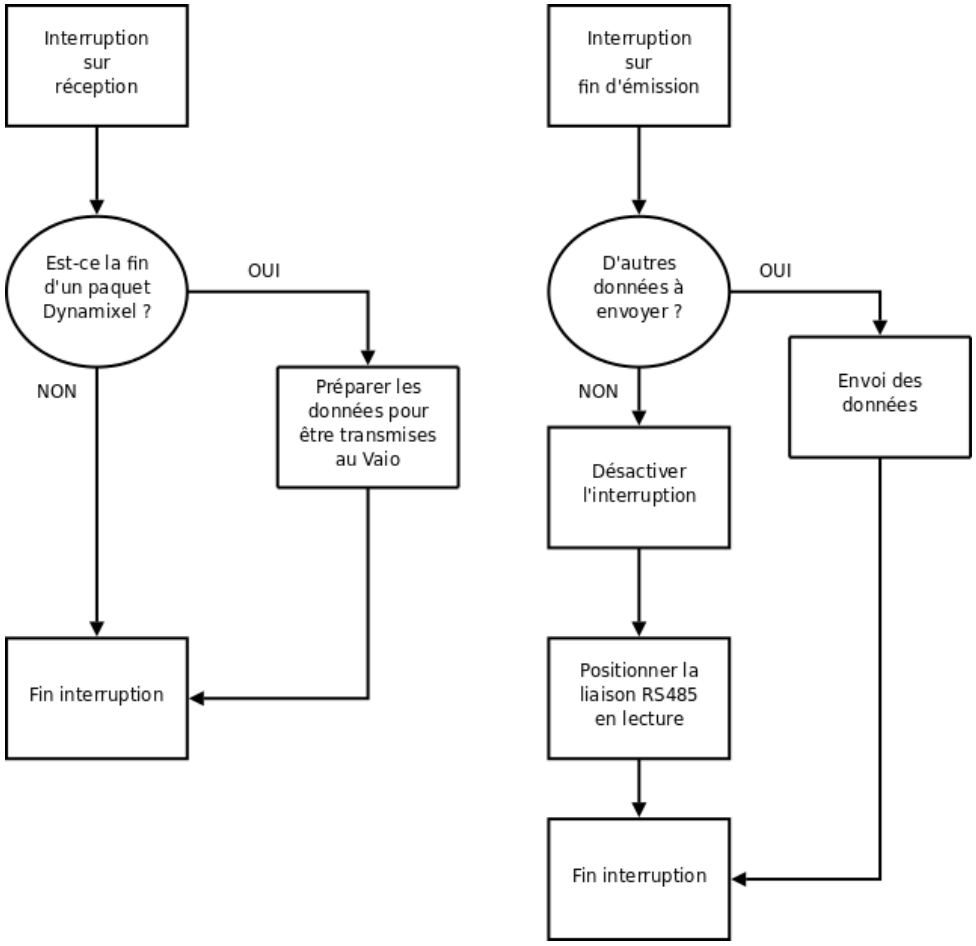


FIG. 17 – Différentes interruptions liée à la communication avec les servomoteurs

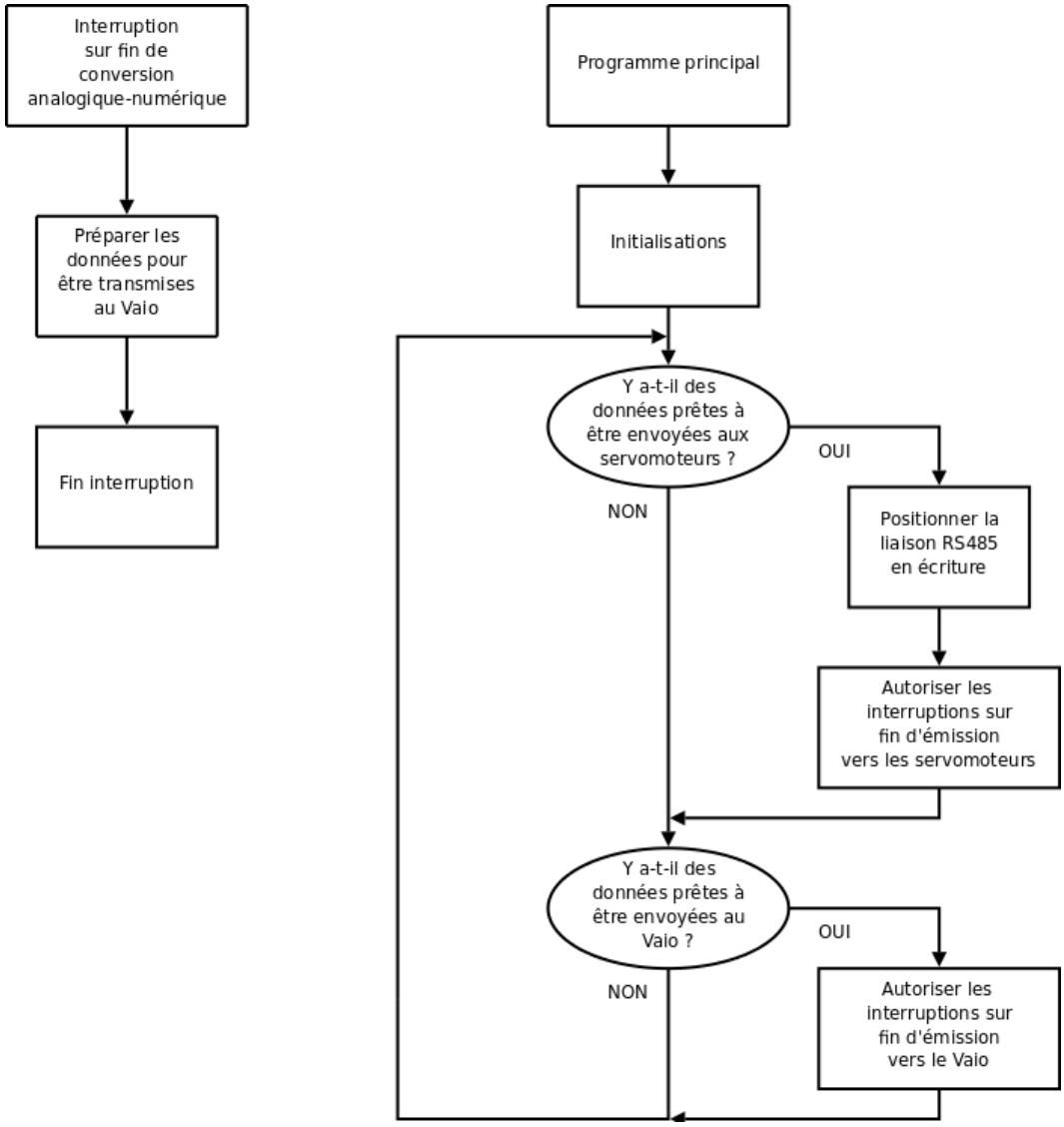


FIG. 18 – Interruption de fin de conversion analogique-numérique et programme principal

Pour faciliter la reconnaissances de paquets Dynamixel, l'envoi et la réception de ces paquets se fait sous la forme de machines d'états. Cela simplifie la programmation et la rend plus claire.
Des structures de vérification des communication non représentés dans les algorithmes précédents ont été également implémentés, permettant d'éviter certaines erreurs de réception. De plus, un "timeout" sur la réception de données en provenance des servomoteurs a été implémenté, évitant ainsi un bloquage de la liaison RS485 dans le cas où le module ne recevrait pas cor-

rectement le paquet d'erreur Dynamixel.

5.3.3 Module de contrôle principal

Le module de contrôle principal (Fig. 19) est embarqué dans le corps de LEON. Ce module embarque une connectique RS485 pour communiquer avec les servomoteurs Dynamixel, ainsi que 4 capteurs de force et une boussole. De plus ce module communique également avec le module de contrôle en balance via le bus I2C.

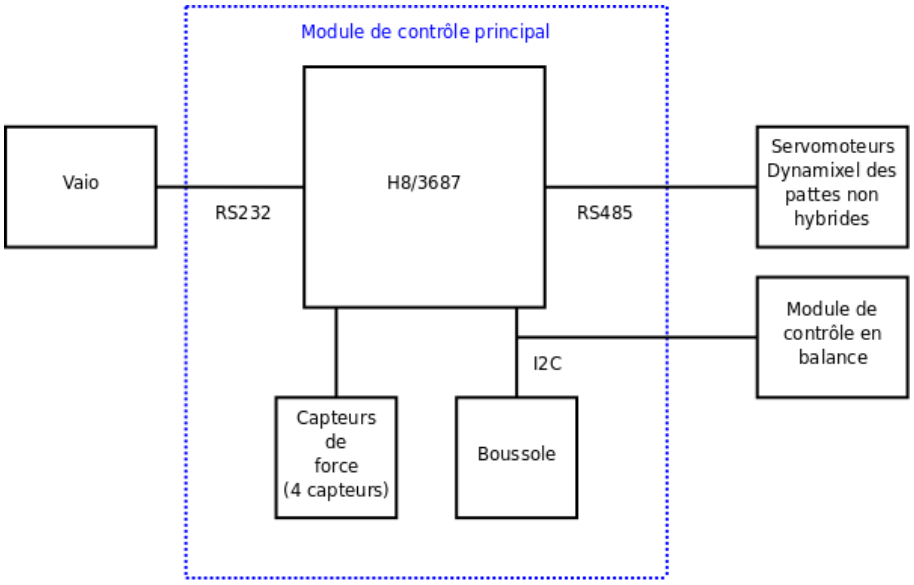


Fig. 19 – Module de contrôle principal

Ce module implémente le même système de passage RS232 à RS485 que les modules de contrôle des pattes hybrides.

I2C

Étant donné que ce module doit communiquer avec le module de contrôle en balance, le bus en maître doit être implémenté sur cette carte.

Boussole

La boussole utilisée est un module acheté chez SparkFun Electronics (Fig. 20). Celle-ci a une précision de 0,1 degré et communique par I2C.

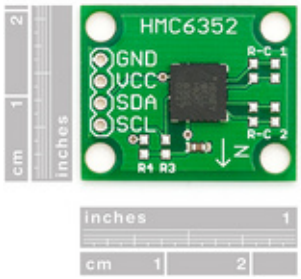


FIG. 20 – Module boussole

Algorithmes du programme de la carte

Le programme de ce module ressemble à celui du module embarqué dans les roues hybrides. Seule l'implémentation de la communication par I2C a été ajoutée, ainsi que le contrôle de la boussole.

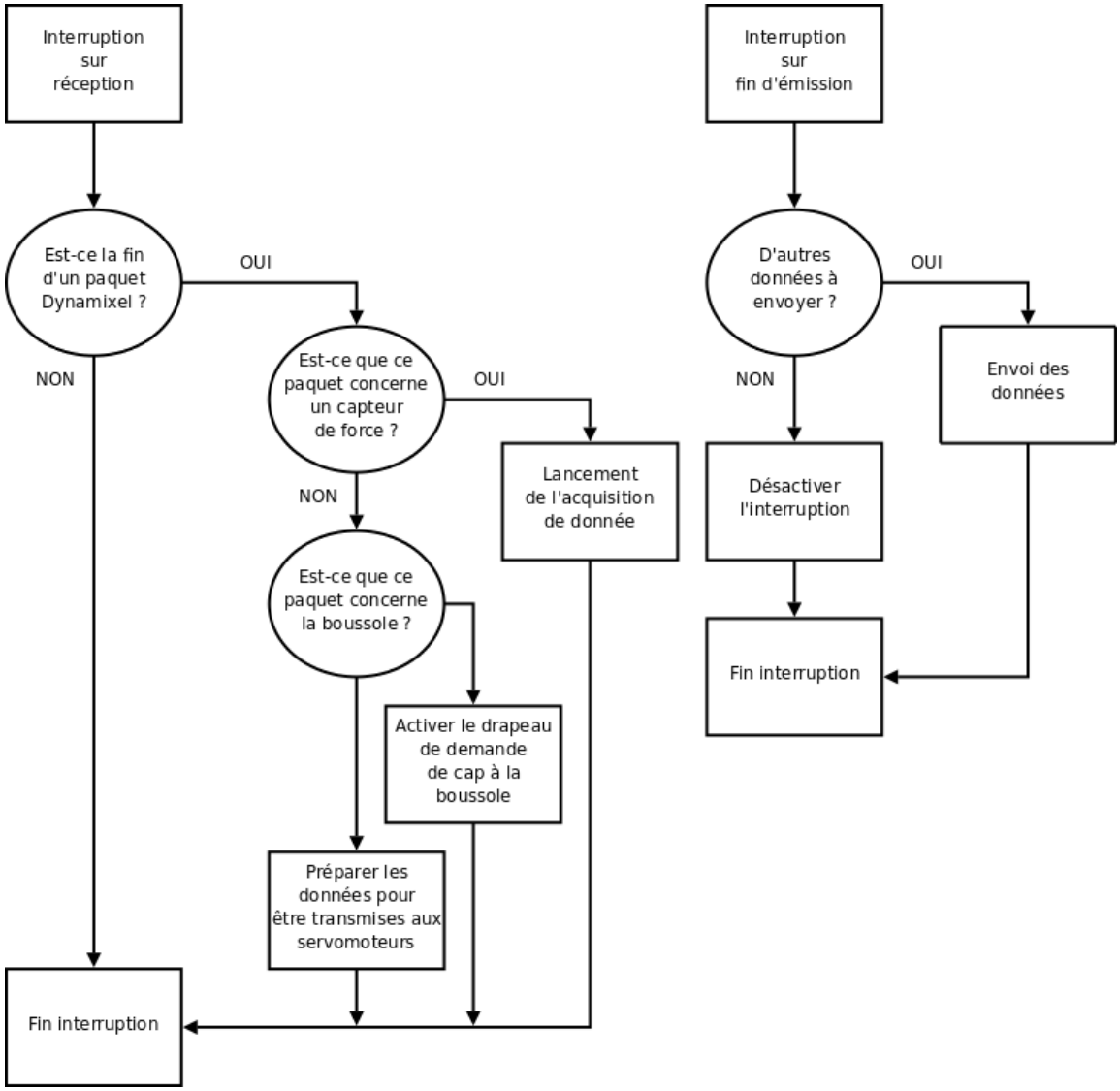


FIG. 21 – Différentes interruptions liée à la communication avec le Vaio

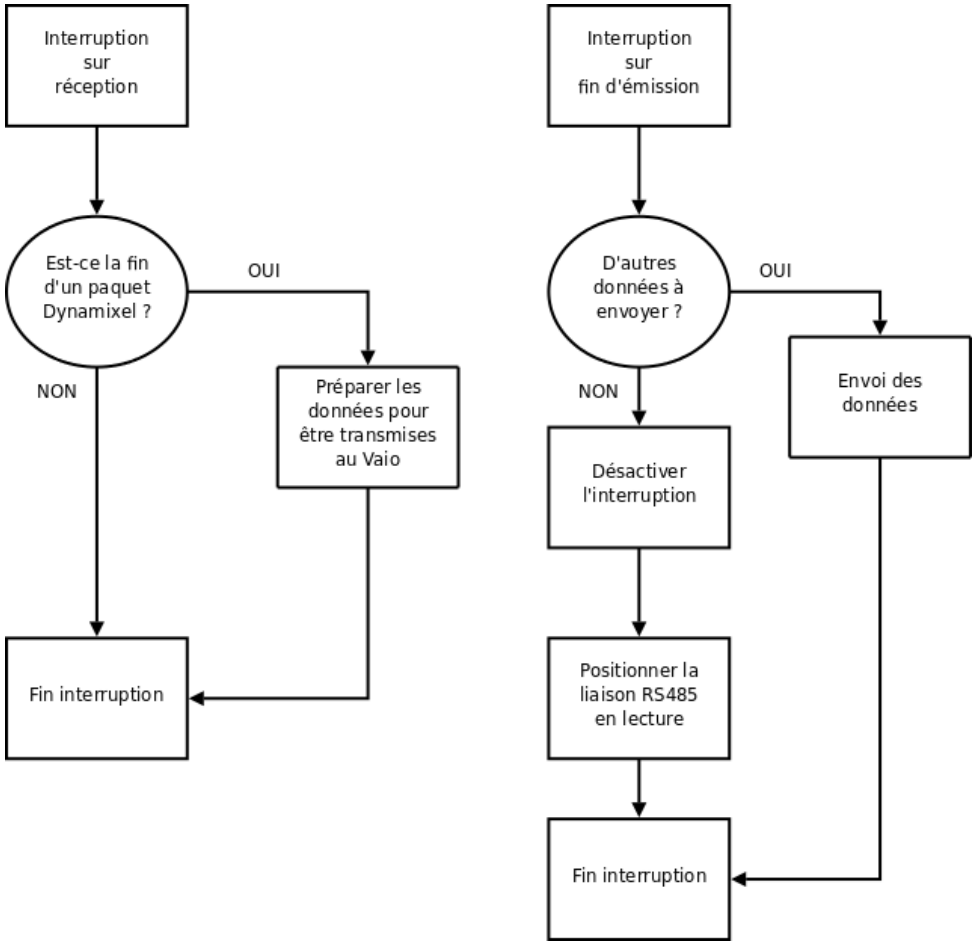


FIG. 22 – Différentes interruptions liée à la communication avec les servomoteurs

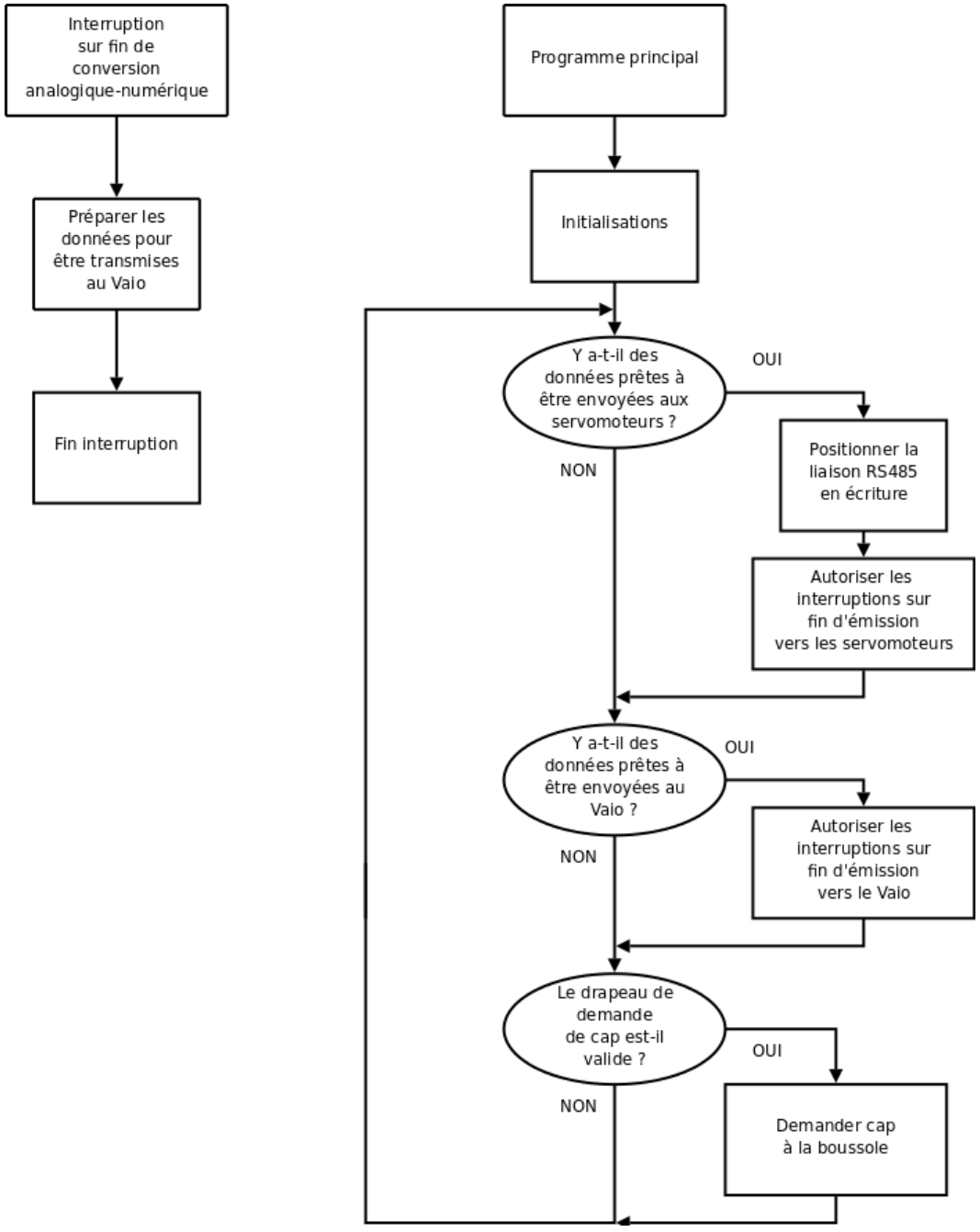


FIG. 23 – Interruption de fin de conversion analogique-numérique et programme principal

5.3.4 Module de contrôle en balance

Le module optionnel de contrôle en balance (Fig. 24) permet un contrôle en balance du robot lorsqu'il est en mode "roues". Pour cela, il est necessaire de connaître la vitesse de rotation du robot autour de ses roues ainsi que son inclinaison.

I2C

Étant donné que ce module communique par I2C avec le module de contrôle principal, l'I2C en mode esclave doit être implémenté.

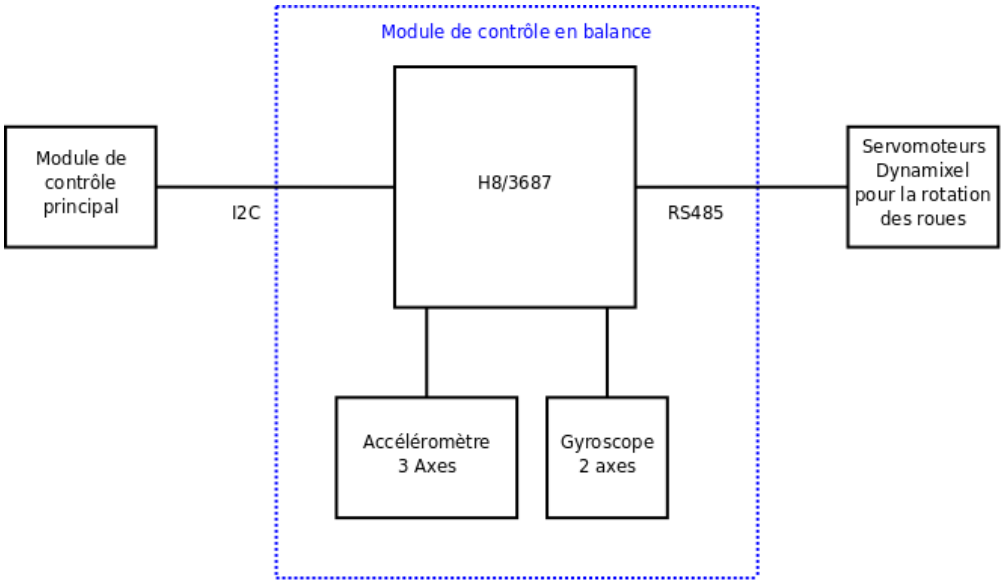


Fig. 24 – Module de contrôle en balance

Mesure de la vitesse de rotation du robot autour de ses roues

Pour mesurer la vitesse de rotation autour de l'axe des roues, on peut utiliser un gyroscope. Le gyroscope utilisé est un IDG300 (Fig. 25). Celui-ci est connecté aux ADCs du micro-contrôleur.

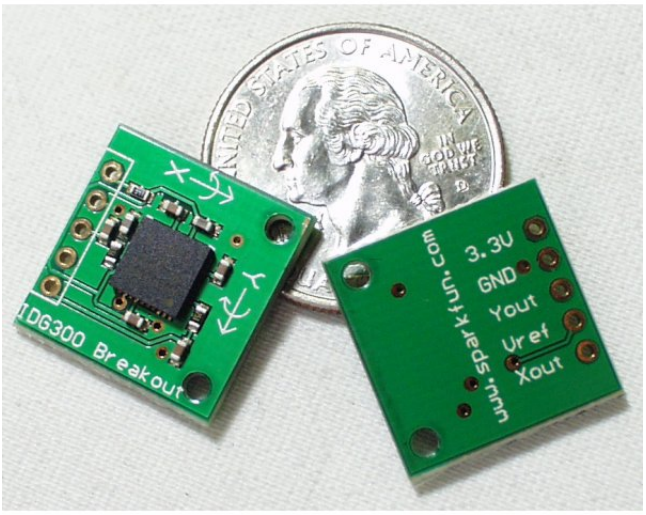


FIG. 25 – Gyroscope utilisé

Mesure de l'inclinaison du robot

Pour mesurer l'inclinaison du robot , on peut utiliser un accéléromètre et mesurer une composante de l'accélération gravitationnelle. L'accéléromètre est un ADXL330 (Fig. 26).

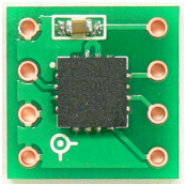


FIG. 26 – Accéléromètre utilisé

5.4 Plan de test de conformité

Pour tester les différentes cartes électroniques, les cartes seront connectées au Vaio et aux servomoteurs Dynamixel. Dans un premier temps, la validité des données transmises par la carte seront vérifiées à l'aide de points de test. Les différentes trames envoyées par chacune des cartes seront analysées pour vérifier qu'elles correspondent bien à ce qu'elles doivent renvoyer.
Dans un second temps, les capteurs seront connectés et nous vérifierons

que les trames Dynamixel renvoyées lors d'une demande de mesure soient conformes au protocole Dynamixel.
Pour terminer, les cartes seront insérées dans le robot et nous testerons la boucle de réponse d'asservissement de celui-ci pour vérifier si celle-ci a été améliorée.

5.5 Validation

A l'aide d'un logiciel de test, sur un ordinateur quelconque, envoyant des commandes Dynamixel, et analysant les réponses renvoyées par ceux-ci, les deux premiers points précédemment cités ont été vérifiés : les servomoteurs répondent correctement aux ordres envoyés, et les trames envoyées par chaque éléments sont valides.
Lors du test de la boucle d'asservissement, aucun saut à des temps de réponse élevé n'a été observé lors d'aucun des tests. De plus, le temps de réponse du système à été réduit par rapport au temps de réponse moyen du système d'origine.

5.6 Résultats

Le système, tel que réalisé, permet maintenant au robot de fonctionner en mode "roue". Le système réalisé est modulaire et transparent depuis le Vaio. Malheureusement, le système de contrôle en balance a à peine été commencé. Mais celui-ci étant optionnel et modulaire, son insertion ou son retrait du système ne gène pas ce dernier.

6 Conclusion

La majorité du travail demandé a été réalisé avec succès : une architecture électronique modulaire pour le robot a été réalisée, permettant ainsi au Docteur E. Rohmer de poursuivre son sujet de recherche sur LEON. Malheureusement, le module de contrôle en balance n'a pas été terminé, partie du projet proposant le plus grand challenge technique.

Ce stage a conforté mon envie de travailler dans la robotique, ou du moins, dans le domaine des systèmes embarqués (domaine que j'avais choisi pour ma troisième année à l'ENSEIRB).
Ce stage au Japon m'a permis d'observer la manière de travailler des japonais, ce qui change beaucoup de la manière française. Cependant, comme cela

se passait dans un laboratoire d'université avec des étudiants, cela doit également être un peu différents de méthodes de travail en entreprise au Japon. La barrière de la langue a été ma plus grande difficulté durant ce stage. Peu de japonais parlent anglais. De plus, l'environnement de développement pour micro-contrôleur H8 était entièrement en japonais, ce qui a quelque peu ralenti le développement lorsqu'il fallait corriger des erreurs/warnings. Cependant, cela m'a forcé à apprendre plus vite certains mots/caractères, améliorant ma pratique du japonais.

7 Glossaire

- DX117 : servomoteur numérique de Dynamixel
- Dynamixel : compagnie fabriquant des servomoteurs numérique ; par association, nom également donnés aux servomoteurs de cette marque
- ERODE : plateforme de simulation et de contrôle haut niveau de LEON basée sur ODE, “ER” étant les initiales de l’auteur de cette plateforme : Eric Rohmer
- H8 : Micro-contrôleur de Renesas Technology
- LEON : acronyme de Lunar Exploration Omnidirectional Netbot, nom du robot hexapode
- ODE : Open Dynamics Engine, librairie de simulation d’objets dynamiques
- SH : Micro-contrôleur de Renesas Technology

8 Bibliographie

- Exemples de codes pour micro-contrôleur H8 : http ://www.renesas.com

9 Annexes

Ce programme est celui du module de contrôle des pattes hybrides dont les algorithmes sont présentés dans les figures 16, 17 et 18.

Ce programme reçoit des ordres du Vaio sur l'UART nommé SCI3, puis les renvoie si nécessaire vers les servomoteurs Dynamixel sur l'UART nommé SCI3_2, en gérant la direction de la liaison RS485. Il renvoie également les données envoyées par les servomoteurs vers le Vaio.

Une analyse des paquets envoyés par le Vaio permet de traiter l'information pour savoir si cela concerne un capteur ou s'il faut changer le niveau de retour des erreurs des servomoteurs Dynamixel (configurable).

9.1 Main et Interruptions

```
/**************************
* Author: Julien CHEVRIER
* 2008
**************************/

#include <sysio.h>
#include <stdio.h>
#include "../include_h8/300H/header3687.h"
#include "uart_def.h"
#include "dynamixel.h"
#include "adc.h"

// comment the following line if dynamixels' status answer is desactivated by default
//#define DYNAMIXEL_STATUS_ANSWER_ACTIVATED_DEFAULT

/* Direction control of the RS485 serial */
#define READ (char)0
#define WRITE (char)1

#define toggle_led() IO.PDR3.BIT.B5^=1
/*toggle state of a LED (optionnal) */


#define SET_DIRECTION_RS485(dir) IO.PDR2.BIT.B3=dir
/* Direction Switch for Dynamixel RS485 communication */

#define IS_READING_RS485 (!IO.PDR2.BIT.B3)
/* Get direction switch status */


dyn_instr to_dyn[3];
// packet received from the Vaio and to be sent to Dynamixels
```

```
dyn_instr to_vaio[3];
// packet received from Dynamixels/sensors and to be sent to the Vaio




// Global variables

#ifdef DYNAMIXEL_STATUS_ANSWER_ACTIVATED_DEFAULT
unsigned char dynamixel_answer_activated=2;
#else
unsigned char dynamixel_answer_activated=1;
#endif

unsigned int time_counter; //time counter for dynamixel response time
char timeout_flag; //reception timeout for dynamixel response
unsigned char last_adc; //last adc number
unsigned char last_adc_id; //the ID of this adc



// Functions

void init_IO_ports(void)
{
        IO.PCR2=0x08; //P23 drives the direction of RS485


        IO.PCR3=0xA0;// P35 and P37 drive LEDs (optionnal)
                                // One is used for checking if the board is reading
         // or writing on the RS485 serial line,
                                // the other is used for checking if the board
         // receives correctly data from the Vaio
}



/******* ADC *******/
void interrupt adc_end(void) //vector number 25
{
        unsigned char nb=0;
        unsigned short data;
        AD.CSR.BIT.ADF=0; //Reset interrupt flag

        data=adc_get_result(last_adc);

        while(to_vaio[nb].transmit!=0) //Try find a free buffer
```

```
                nb=(nb+1)%3;

        /* Fill in the buffer */
        to_vaio[nb].ID=last_adc_id;
        to_vaio[nb].length=4;
        to_vaio[nb].instruction=0x00;
        to_vaio[nb].parameters[0]=(data&0xFF); // Lower 8 bits first
        to_vaio[nb].parameters[1]=((data>>8)&0x03); // Higher 2 bits
        calculate_checksum(&(to_vaio[nb]));

        to_vaio[nb].transmit=1; //buffer filled in, can be transmitted
}

/******* SCI3 *******/
void vaio_rx(void) // Receive from Vaio
{
        static char state=0; // For the state machine
        static char nb=0; //Number of the Buffer which is currently in use

        /*****
        nb: the number of the buffer (0,1 or 2)

        state: 0: no byte received
                   others: number of bytes received
        *****/

        unsigned char temp=SCI3.RDR;
        SCI3.SSR.BIT.RDRF=0; //Clear flag


        toggle_led(); //optionnal
        if( (state<2) && (temp==0xff) ) //start of incoming packet
        {
                state++;
        }
        else if (state==2) //ID
        {
                if(temp!=0xFF)
                {
                        to_dyn[nb].ID=temp;
                        state++;
                }
        }
        else if(state==3) //length
        {
                to_dyn[nb].length=temp;
                state=4;
        }
        else if(state==4) //instruction
```

```
        {
                to_dyn[nb].instruction=temp;
                state=5;
        }
        else if(  (state>4) && ( state<(to_dyn[nb].length+3) ) ) //parameters
        {
                to_dyn[nb].parameters[state-5]=temp;
                state++;
        }
        else if( state==(to_dyn[nb].length+3)) //checksum
        {
                to_dyn[nb].checksum=temp;
                to_dyn[nb].transmit=1;
                state=0;

                if( (to_dyn[nb].ID==0xfe) && \
        (to_dyn[nb].instruction==0x03) && \
        (to_dyn[nb].parameters[0]==0x10))//changing status return level
                {
                        dynamixel_answer_activated=to_dyn[nb].parameters[1];
                }
                /*************************************************
                * Add things here to control different sensors *
                *************************************************/
                if(to_dyn[nb].ID==0x09 || to_dyn[nb].ID==0x08) //sensor(s)
                {
                        to_dyn[nb].transmit=0; // empty

                        /* case for only 2 sensors */
                        if(to_dyn[nb].ID==0x08)
                        {
                                last_adc=0;
                                last_adc_id=8;
                        }
                        else
                        {
                                last_adc=4;
                                last_adc_id=9;
                        }
                        adc_start_convert(last_adc);
                }


                /*************************************************
                * End of sensors control part                    *
                *************************************************/

                nb=(nb+1)%3; // Use next buffer
        }
```

```
        else //error
        {
                state=0;
        }

}

void vaio_tx(void) // Send to Vaio
{
        static char state=0; // For the state machine
        static char nb=0; //Number of the Buffer which is currently in use

        if (to_vaio[nb].transmit!=2) //not transmitting
        {
                if(to_vaio[nb].transmit==1) //ready to transmit
                {
                        state=0;
                        to_vaio[nb].transmit=2; //put it in transmit state
                }
                else if(to_vaio[(nb+1)%3].transmit==1)
      //other buffer ready to be transmitted
                {
                        nb=(nb+1)%3;
                        state=0;
                        to_vaio[nb].transmit=2; //put it in transmit state
                }
                else if(to_vaio[(nb+2)%3].transmit==1)
      //other buffer ready to be transmitted
                {
                        nb=(nb+2)%3;
                        state=0;
                        to_vaio[nb].transmit=2; //put it in transmit state
                }

                else    //nothing to do
                {
                        SCI3.SCR3.BIT.TIE=0; //stop transmission interrupt
                        return;
                }
        }


        // State machine for transmitting data

        if(state<2) //start of packet
        {
                state++;
                SCI3.TDR=0xFF;
        }
```

```
        else if (state==2) //ID
        {

                SCI3.TDR=to_vaio[nb].ID;
                state=3;

        }
        else if(state==3) //length
        {
                SCI3.TDR=to_vaio[nb].length;
                state=4;
        }
        else if(state==4) //instruction
        {
                SCI3.TDR=to_vaio[nb].instruction;
                state=5;
        }
        else if(  (state>4) && ( state<(to_vaio[nb].length+3) ) )
    //parameters
        {
                SCI3.TDR=to_vaio[nb].parameters[state-5];
                state++;
        }
        else if( state==(to_vaio[nb].length+3)) //checksum
        {
                SCI3.TDR=to_vaio[nb].checksum;
                to_vaio[nb].transmit=0; //no more things to transmit
                state=0;
                SCI3.SCR3.BIT.TIE=0; //stop transmission interrupts

                nb=(nb+1)%3; // Use next buffer
        }

}

/************************************************************
* On H8-3687, interrupts on Receiving and Transmitting of SCI3 *
* share the same vector. We need to determine which          *
* of these interrupts have been started.                     *
************************************************************/
void interrupt vaio(void) // vector number: 23
{
        if(SCI3.SSR.BIT.RDRF==1)
                vaio_rx();
        else if(SCI3.SSR.BIT.TDRE==1)
                vaio_tx();
}

/******* End of SCI3 *******/
```

```
/******* SCI3_2 *******/

void dynamixel_rx(void) // Receive from Dynamixels
{
        static char state=0;
        static char nb=0;
        static char errors=0;

        /*****
        nb: the number of the buffer

        state: 0: no byte received
                   others: number of bytes received
        *****/

        unsigned char temp=SCI3_2.RDR;
        SCI3_2.SSR.BIT.RDRF=0; //Clear flag


        /*** Reception errors ***/
        time_counter=0; // We received a byte, time counter resetted

        // Time to receive a packet too long
        // The last packet has been ignored and we restart reading a new packet
  // This is not launched just after the timeout detection
  // but at the next byte received
        if(timeout_flag==1)
        {
                state=0;
                timeout_flag=0;
                to_vaio[nb].transmit=0;
        }

        /*** End of reception errors ***/

        if( (state<2) && (temp==0xff) ) //start of incoming packet
        {
                state++;
        }
        else if (state==2) //ID
        {
                if(temp!=0xFF)
                {
                        to_vaio[nb].ID=temp;
                        state++;
                }
```

```
        }
        else if(state==3) //length
        {
                to_vaio[nb].length=temp;
                state=4;
        }
        else if(state==4) //instruction
        {
                to_vaio[nb].instruction=temp;
                state=5;
        }
        else if(  (state>4) && ( state<(to_vaio[nb].length+3) ) ) //parameters
        {
                to_vaio[nb].parameters[state-5]=temp;
                state++;
        }
        else if( state==(to_vaio[nb].length+3)) //checksum
        {
                to_vaio[nb].checksum=temp;
                to_vaio[nb].transmit=1; // Ready to be transmitted
                state=0;

                SET_DIRECTION_RS485(WRITE); // We successfully read the packet,
                                // nothing more to be read

                nb=(nb+1)%3;
        }
        else //error
        {
                state=0;
                errors++;
                if (errors>=2)
                {
                        SET_DIRECTION_RS485(WRITE);
                        errors=0;
                }
        }
}


void dynamixel_tx(void) // Send to Dynamixels
{
        static char state=0; // For the state machine
        static char nb=0; //Number of the Buffer which is currently in use

        if (to_dyn[nb].transmit!=2) //not transmitting
        {
                if(to_dyn[nb].transmit==1) //ready to transmit
                {
```

```
                        state=0;
                        to_dyn[nb].transmit=2; //put it in transmitting status
                }
                else if(to_dyn[(nb+1)%3].transmit==1)
        //other buffer ready to be transmitted
                {
                        nb=(nb+1)%3;
                        state=0;
                        to_dyn[nb].transmit=2; //put it in transmitting status
                }
                else if(to_dyn[(nb+2)%3].transmit==1)
        //other buffer ready to be transmitted
                {
                        nb=(nb+2)%3;
                        state=0;
                        to_dyn[nb].transmit=2; //put it in transmitting status
                }
                else    //nothing to do
                {
                        SCI3_2.SCR3.BIT.TIE=0;
                        return;
                }
        }


        // State machine for transmitting data

        if(state<2) //start of packet
        {
                state++;
                SCI3_2.TDR=0xFF;
        }
        else if (state==2) //ID
        {

                SCI3_2.TDR=to_dyn[nb].ID;
                state=3;

        }
        else if(state==3) //length
        {
                SCI3_2.TDR=to_dyn[nb].length;
                state=4;
        }
        else if(state==4) //instruction
        {
                SCI3_2.TDR=to_dyn[nb].instruction;
                state=5;
        }
```

```
        else if(  (state>4) && ( state<(to_dyn[nb].length+3) ) ) //parameters
        {
                SCI3_2.TDR=to_dyn[nb].parameters[state-5];
                state++;
        }
        else if( state==(to_dyn[nb].length+3)) //checksum
        {
                SCI3_2.TDR=to_dyn[nb].checksum;
                state++;

        }
        else if (state==(to_dyn[nb].length+4))
        {

                /****************************************
                * Check if we have to read the          *
                * RS485 serial.                         *
                * This depends on the return level status *
                * and if we sent broadcast or not.      *
                ****************************************/
                if(to_dyn[nb].ID!=0xFE)
                {
                        switch(dynamixel_answer_activated)
                        {
                                case 1:
                                        if(to_dyn[nb].instruction!=0x02)
                                                break;
                                case 2:
                                        SET_DIRECTION_RS485(READ);
                                        break;
                                case 0:
                                default:
                                        break;
                        }
                }


                to_dyn[nb].transmit=0; // Stop transmitting
                nb=(nb+1)%3; // Next buffer
                SCI3_2.SCR3.BIT.TIE=0; // Stop transmitting interrupt
                state=0; // Reset state machine
        }
}


/*************************************************************
* On H8-3687, interrupts on Receiving and Transmitting of SCI3_2 *
* share the same vector. We need to determine which          *
* of these interrupts have been started.                     *
```

```
*****************************************************************/
void interrupt dynamixel(void) // vector number: 32
{
        if(SCI3_2.SSR.BIT.RDRF==1)
                dynamixel_rx();
        else if(SCI3_2.SSR.BIT.TDRE==1)
                dynamixel_tx();
}

/******* End of SCI3_2 *******/

int main()
{
        // Initialisation of buffers
        to_dyn[0].transmit=0;
        to_dyn[1].transmit=0;
        to_dyn[2].transmit=0;
        to_vaio[0].transmit=0;
        to_vaio[1].transmit=0;
        to_vaio[2].transmit=0;

        init_IO_ports();
        SET_DIRECTION_RS485(WRITE);

        adc_init(IT_ON);

        SCI3_2_init();
        SCI3_init();

        // Initialisation of receive error detection
        time_counter=0;
        timeout_flag=0;


        _ei(); // Enable Interrupts
        toggle_led(); //optionnal


        while(1) // Main loop
        {
                // Is something ready to be transmitted to Dynamixels ?
                if(((to_dyn[0].transmit==1) || \
        (to_dyn[1].transmit==1)|| \
        (to_dyn[2].transmit==1)) && !IS_READING_RS485)
                {
                        SCI3_2.SCR3.BIT.TIE=1;
                }

                // Is something ready to be transmitted to Vaio ?
```

```
                if(to_vaio[0].transmit==1 || \
        to_vaio[1].transmit==1 || \
        to_vaio[2].transmit==1)
                {
                        SCI3.SCR3.BIT.TIE=1;
                }

                IO.PDR3.BIT.B7=IS_READING_RS485;
    //optionnal, a LED gives the status of the line

                if(IS_READING_RS485)
      /*** Detects errors in communication ***/
                {
                        time_counter++;
                        if(time_counter>=10000)
        // It took too much time between reception of 2 bytes
                        {
                                timeout_flag=1;
                                time_counter=0;
                                SET_DIRECTION_RS485(WRITE);
        // Stop reading the RS485 serial
                        }
                }
                else
                        time_counter=0;

        }
        return 0;
}
```

9.2 ADCs

9.2.1 adc.h

```
/*************************
* Author: Julien CHEVRIER
* 2008
*************************/

#define IT_ON 1
#define IT_OFF 0

//initialisation of adc
// argument is IT_OFF or IT_ON for unvalidating/validationg interrupts on end of conversio
void adc_init(char);

//start conversion
```

```
//argument is the number of ADC to use
void adc_start_convert(unsigned char);

//read the result of conversion
//argument is the number of ADC to use
//Stopping function ! (polling method)
unsigned int adc_get_result(unsigned char);

//Convert and read the result
//argument is the number of ADC to use
//Stopping function ! (polling method)
unsigned int adc_get_convert(unsigned char);
```

9.2.2 adc.c

```
/*************************
* Author: Julien CHEVRIER
* 2008
*************************/

#include "adc.h"
#include "../include_h8/300H/header3687.h"

void adc_init(char activate_it) // Initialisation of ADC
{
        if (activate_it==0)
                AD.CSR.BYTE=0;
        else
                AD.CSR.BYTE=0x40;
}

void adc_start_convert(unsigned char adc_number) // Start converting a value
{
        AD.CSR.BIT.CH=adc_number;
        AD.CSR.BIT.ADST=1;
}

unsigned int adc_get_result(unsigned char adc_number) // Get the result of a conversion
{
        adc_number%=4;
        while(AD.CSR.BIT.ADST==1);
        switch(adc_number)
        {
                case 0:
                        return AD.DRA>>6;
                case 1:
                        return AD.DRB>>6;
                case 2:
```

```
                        return AD.DRC>>6;
                case 3:
                        return AD.DRD>>6;
        }
}

unsigned int adc_get_convert(unsigned char adc_number) // Convert a value, then return the
{
        adc_start_convert(adc_number);
        return adc_get_result(adc_number);
}
```

9.3 UARTs

9.3.1 uart_def.h

```
/*************************
* Author: Julien CHEVRIER
* 2008
*************************/

/* Initialisation functions */
void SCI3_init(void);
void SCI3_2_init(void);

/* Data transmitting functions */
void SCI3_Tx_Char (unsigned char data);
void SCI3_2_Tx_Char (unsigned char data);
```

9.3.2 uart_def.c

```
/*************************
* Author: Julien CHEVRIER
* 2008
*************************/

//All values for UART Bit Rate have calculated for a microcontroller using 16MHz quartz

#include "uart_def.h"
#include <../include_h8/300H/header3687.h>

void SCI3_init(void)
{
 int i ;
 IO.PMR1.BIT.TXD=1; //enable transmitting
 SCI3.SCR3.BYTE = 0x00;
```

```
 SCI3.SMR.BYTE = 0x00;
 SCI3.BRR = 8; //12: Bit Rate: 38400 8: Bit Rate 57600, 4: 100000 3:125000
 for(i=0;i<700;i++); //wait few cycles
 SCI3.SCR3.BYTE = 0x70; //RX and TX activated, interrupts on Reception allowed
 }

void SCI3_2_init(void)
{
 int i ;
 IO.PMR1.BIT.TXD2=1; //enable transmitting
 SCI3_2.SCR3.BYTE = 0x00;
 SCI3_2.SMR.BYTE = 0x00;
 SCI3_2.BRR = 3; //Bit Rate: 125000, error rate: 0% (crystal: 16MHz)
 for(i=0;i<700;i++); //wait few cycles
 SCI3_2.SCR3.BYTE = 0x70; //RX and TX activated, interrupts on Reception allowed
 }

void SCI3_Tx_Char (unsigned char data) //send a byte
{
 while(SCI3.SSR.BIT.TDRE==0);
 SCI3.TDR=data;
 SCI3.SSR.BIT.TDRE=0;
}

void SCI3_2_Tx_Char (unsigned char data) //send a byte
{
 while(SCI3_2.SSR.BIT.TDRE==0);
 SCI3_2.TDR=data;
 SCI3_2.SSR.BIT.TDRE=0;
}
```

9.4 Utilitaires pour le contrôle des servomoteurs Dynamixel

9.4.1 dynamixel.h

```
/*************************
* Author: Julien CHEVRIER
* 2008
*************************/

typedef struct instruction_paquet
{
        unsigned char ID;
        unsigned char length;
        unsigned char instruction;
        unsigned char parameters[60];
```

```
        unsigned char checksum;
        unsigned char transmit; //0: nothing,
                                //1: ready to be transmitted
                                //2: transmitting
} dyn_instr;

/* Calculate the checksum of a packet */
void calculate_checksum(dyn_instr * di);
```

9.4.2 dynamixel.c

```
/*************************
* Author: Julien CHEVRIER
* 2008
*************************/

#include "dynamixel.h"

void calculate_checksum(dyn_instr *di) //Calculate the checksum of a packet
{
        unsigned char i;

        unsigned char temp=di->ID+di->length+di->instruction;
        for(i=0;i<di->length-2;i++)
        {
                temp+=di->parameters[i];
        }

         di->checksum=~temp;
}
```