Rapport de stage - $3^{ème}$ année E.S.S.I.
(ECOLE SUPERIEURE EN SCIENCES INFORMATIQUES)

Mise à jour du contrôleur du robot bipède BIP

FUCHET Jérôme

Maître de stage : Hervé Mathieu

Juin 2003

Remerciements

Je tiens à remercier toute l'équipe des Moyens Robotiques, nouvellement baptisée Support Expérimentation et Développement (SED), pour leur accueil, leur disponibilité et pour m'avoir fourni tous les outils nécessaires à mon travail. Merci à Roger Pissard-Gibollet, Gérard-Baille, Soraya Arias, Hervé Mathieu, Jean-François Cuniberto et Sébastien Jardé.

Je remercie M^r Tigli pour m'avoir aidé dans ma recherche de stage.

Enfin je remercie les autres stagiaires de la halle robotique pour leur bonne humeur et leur accueil.

Table des matières

Table des figures

Chapitre 1

Introduction

L'INRIA Rhône-Alpes et le Laboratoire de Mécanique des Solides de Poitiers (LMS) ont développé, dans le cadre d'un projet commun, un robot bipède. Ce robot, BIP, doté de quinze degrés de liberté, est capable de marcher sur un sol plat et de monter des marches d'escaliers. Mais il est mécaniquement capable de faire plus puisqu'il possède quasiment le même nombre de degré de liberté qu'un être humain au niveau des hanches et des jambes.
BIP embarque un ordinateur, le contrôleur, qui récupère les informations en provenance des capteurs, applique une loi de commande et retourne les consignes aux moteurs. Le contrôleur actuel du robot est assez ancien et ne permet pas d'exécuter des lois de commande complexes. L'équipe SED, en charge de l'évolution de la plateforme à l'INRIA Rhône-Alpes, a donc fait l'acquisition d'un nouveau contrôleur plus récent.
Mon travail a consisté à porter les drivers et les librairies existantes sur le nouveau contrôleur. Plus précisément, le but a été de passer d'un système avec un processeur Motorola 68040 et système d'exploitation VxWorks[1][1] à un système avec Motorola PowerPC sous Linux puis Linux-RTAI[2][2].
Ce rapport présente le travail que j'ai réalisé. Les personnes désireuses d'approfondir le côté technique de ce stage pourront consulter la documentation technique INRIA[4].

[1] VxWorks est un système d'exploitation temps réel crée par WindRiver Systems

[2] RTAI est un autre système d'exploitation temps réel développé initialement par le département d'ingénierie aérospatiale de l'école polytechnique de Milan

Chapitre 2

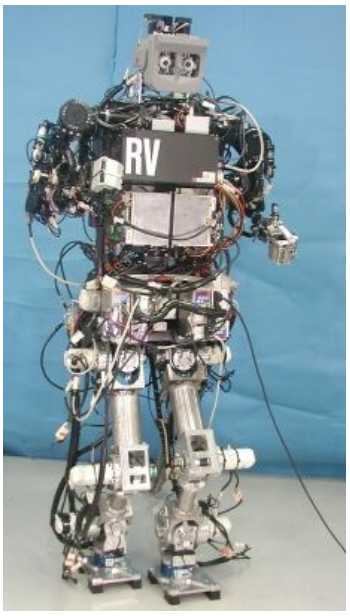
La plateforme BIP

2.1 Pourquoi créer un robot bipède ?

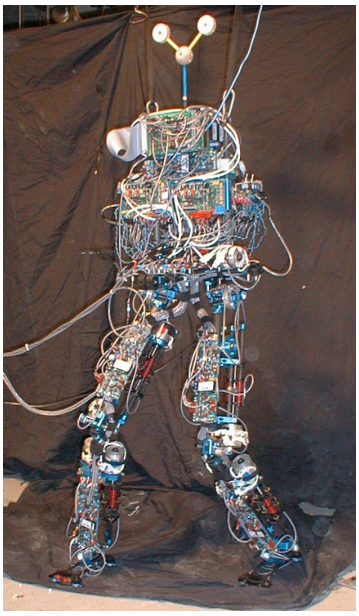
Les robots marcheurs sont très intéressants car ils sont capables d'évoluer sur des terrains accidentés. Les robots bipèdes offrent en plus l'avantage de pouvoir, a priori, évoluer dans des environnements créés pour des humains. Il est alors possible d'envisager des robots capables d'intervenir dans des sites dangereux pour les hommes, ou encore des robots d'aide à domicile. Mais tous ces débouchés sont encore de l'ordre de la science fiction. Le réel moteur, à l'heure actuelle, du développement de BIP est la compétition internationale.

2.2 Les robots bipèdes existants

La recherche sur les robots marcheurs a commencé dans les années soixante-dix. Les robots bipèdes sont plus complexes à commander que les robot à pattes : ce sont des systèmes où l'équilibre est très dur à maintenir. Les laboratoires Humanoid Robotics Institute de la Waseda University au Japon et le Leg Laboratory du Massachusetts Institute of Technology aux Etats-Unis sont particulièrement connus pour leur réalisations. Récemment, HONDA a elle aussi présentée une série de robots marcheurs dont les images ont fait le tour du monde. Il est à noter également que SONY construit de petits robots marcheurs. Ils sont plutôt destinés au marché ludique. Différents robots bipèdes sont visibles sur la figure 2.1.



WABIAN version 5 - laboratoire Humanoid Robotics Institute, Waseda University (Japon)



M2 - Leg Laboratory du MIT (Etats-Unis)

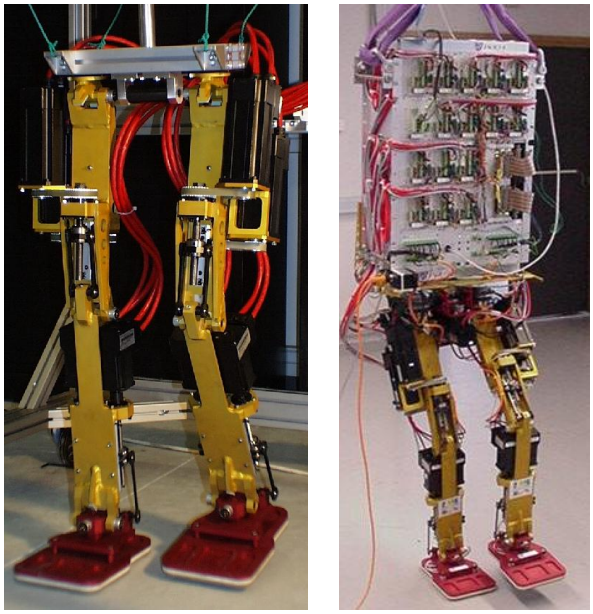
ASIMO - Honda (Japon)

SDR-4X - Sony (Japon)

FIG. 2.1 – Quelques robots bipèdes.

2.3 Présentation de la plateforme BIP

Le projet BIP a débuté en 1995. Il s'agit d'un projet commun de l'INRIA Rhône-Alpes et du Laboratoire de Mécanique des Solides (LMS) de Poitiers. La partie mécanique du robot bipède a été conçue par le LMS, tandis que les parties électroniques et informatiques ont été réalisées par l'INRIA Rhône-Alpes. Les jambes de BIP ressemblent beaucoup aux jambes humaines. La longueur et le poids de chaque "segment" d'une jambe sont semblables à ceux d'une personne humaine. De même les couples articulaires exercés par les moteurs sont du même ordre de grandeur que ceux engendrés par la musculature humaine.
Dans sa première version, celle de 1999, BIP était doté de huits degrés de liberté(ddl) (voir figure 2.2 (1)) et il était accroché à un chariot à quatre roues. Cette version pouvait uniquement effectuer des mouvements de marche dans le plan sagital (voir figure 2.3). En 2000, BIP a évolué dans sa deuxième version, avec quinze degrés de liberté (voir figure 2.2 (2)). Il lui a été rajouté un tronc comprenant toute l'électronique de commande, et sept degrés de liberté au niveau des hanches et du tronc. Cette version mesure 1,80m et pèse 100 kg. Elle est capable de réaliser tous les mouvements des jambes humaines à l'exception des mouvements des orteils.



(1) (2)

FIG. 2.2 – BIP dans sa version à huit ddl (1) puis quinze ddl (2).

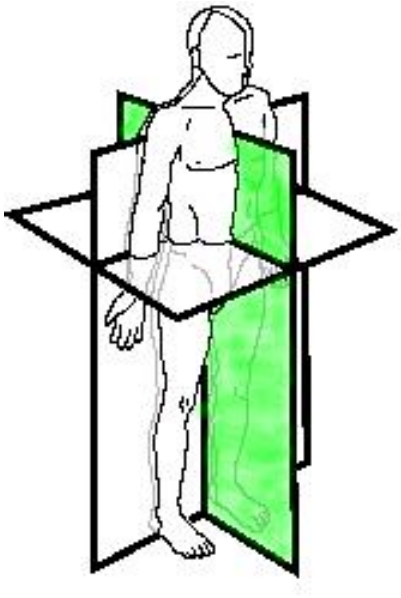


FIG. 2.3 – Illustration du plan sagital (colorié sur le dessin).

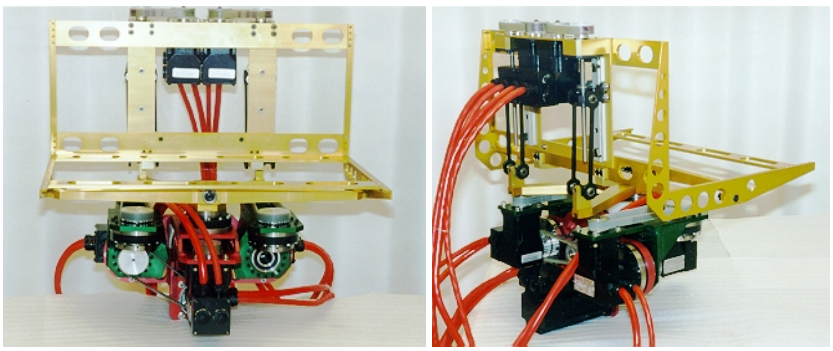


FIG. 2.4 – Les articulations du bassin et du tronc ajoutées sur la version quinze ddl.

2.4 La partie matérielle de la plateforme BIP

Ma description de la partie matérielle se limitera à l'armoire de commande car mon travail concerne le contrôleur. La figure 2.5 permet de situer les jambes, l'armoire de commande et le contrôleur.
Le contrôleur est la partie gérant le calcul embarqué sur la plateforme BIP. Elle est intégrée à l'armoire de commande. Plus précisément il se compose de cartes d'entrée/sortie et d'une carte processeur.

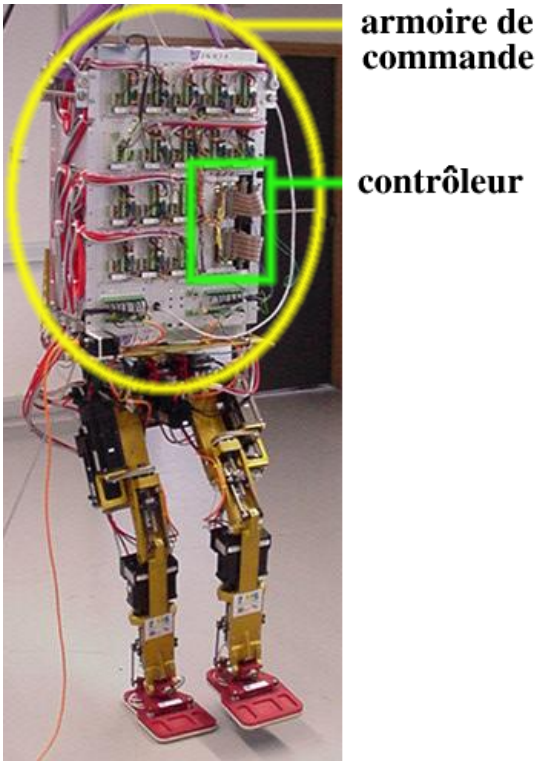


FIG. 2.5 – Présentation de l'armoire de commande.

2.4.1 Les entrées/sorties

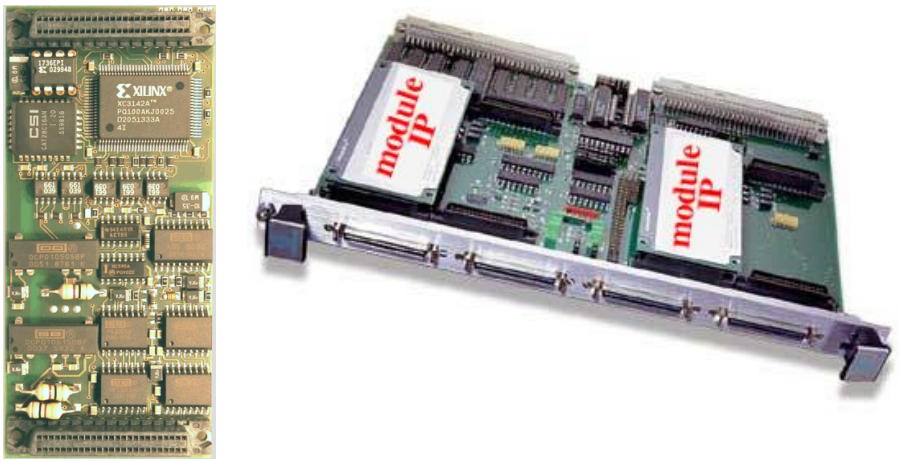
Les entrées/sorties sont assurées au travers de modules au format IP[1] qui viennent s'enficher dans une carte support interfacée avec le bus VME. C'est par l'intermédiaire de ces modules que la carte processeur connaît l'état du robot et lui envoie des consignes. La partie électro-mécanique du robot est composée de quinze moteurs électriques pour bouger les jambes et des capteurs suivants :

- quinze potentiomètres pour connaître la position absolue des articulations.
- quinze encodeurs relatifs montés sur les moteurs pour connaître leur position.
- quinze capteurs de courant pour connaître l'intensité traversant les moteurs.
- six capteurs de pression sous les pieds pour connaître les points d'appui.
- trois inclinomètres pour connaître l'inclinaison des pieds et du tronc.

La relation entre les capteurs et actionneurs avec les modules IP est donnée en figure 2.6.

Module IP	Fonction	Capteurs et actionneurs associés
TIP501	conversion analogique vers numérique	- potentiomètres - capteurs de courant - capteurs de force - inclinomètres
DACSU	conversion numérique vers analogique	commande des moteurs
Quadrature	lecture d'encodeurs de position	encodeurs sur les moteurs

FIG. 2.6 – Relation entre les modules IP, les capteurs et les actionneurs de BIP.



(1) (2)

FIG. 2.7 – Un module IP (1) et la carte mère VME équipée de deux modules IP (2).

[1] Industry Pack

2.4.2 La carte processeur

Il s'agit d'une carte au format VME[2] équipée d'un processeur Motorola 68040 à 32 bits et 32Mhz. Elle est munie d'un port Ethernet 10Mbit/s et utilise le système d'exploitation VxWorks^{TM}. La puissance de calcul de cette carte est insuffisante pour exécuter des lois de commande complexes et elle possède un contrôleur Ethernet lent.

2.5 Partie logicielle de la plateforme BIP

2.5.1 Les drivers et les libraires

L'architecture logicielle de BIP est visible sur la figure 2.8. Les flèches représentent des appels de fonctions.

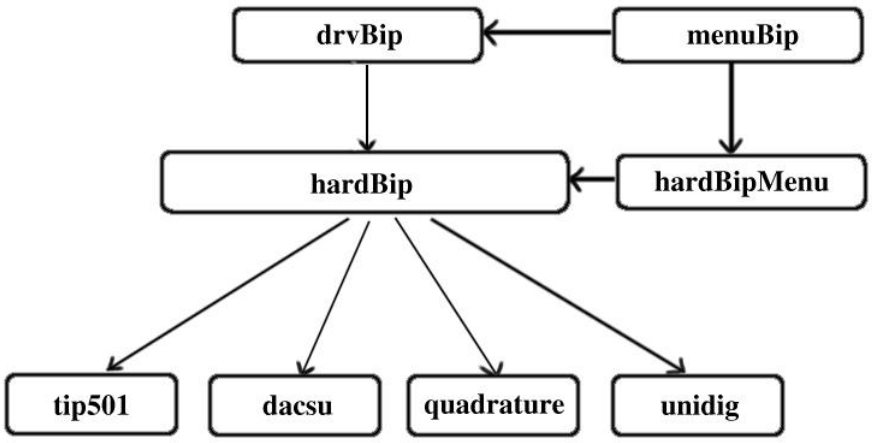


FIG. 2.8 – Architecture logicielle du robot BIP.

Les drivers `tip501`, `dacsu`, `quadrature` et `unidig` commandent les modules IP correspondants. Ces drivers sont utilisés par les librairies `hardBip` et `drvBip`.
`hardBip` est une première couche d'abstraction qui regroupe les fonctions des drivers utiles pour la plateforme. Elle introduit une numérotation des entrées/sorties des modules IP. Cette couche est utilisée par le programme `hardBipMenu` pour tester les drivers.
`drvBip` est une couche d'abstraction supplémentaire qui propose une interface plus en rapport avec l'organisation réelle de la plateforme. Elle propose par exemple de connaître la position de l'articulation du genou droit et non plus l'entrée numéro quatre du module IP quadrature numéro zéro. Toute la partie contrôle utilise uniquement cette librairie pour commander les jambes du robot. Elle est également utilisée par le programme `menuBip` qui permet de faire des tests plus poussés et de commander la plateforme manuellement.

[2]Le bus VME est un bus industriel créé par Motorola.

2.5.2 ORCCAD

Toute la partie contrôle du robot est réalisée sous le logiciel ORCCAD[7], développé à l'INRIA Rhône-Alpes. Il permet de concevoir graphiquement la partie contrôle associée à un robot. Cette conception est composée de quatres étapes : la spécification, la vérification, la simulation et enfin l'exécution. Ces étapes sont visibles sur la figure 2.9 qui est une capture d'écran de la fenêtre principale du logiciel.

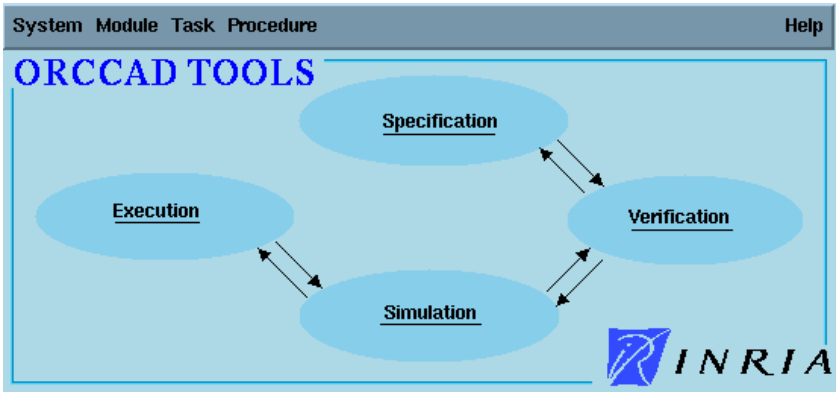


FIG. 2.9 – La fenêtre principale d'ORCCAD.

Sous ORCCAD, le contrôle associé à un robot s'exprime sous forme de "tâches robot" : il s'agit d'un organigramme orienté flot de données. Cette spécification est graphique et utilise une représentation par "boîte". Un exemple de tâche robot est donnée à la figure 2.10, il s'agit de la tâche d'initialisation du robot bipède BIP.

Le comportement d'un module ORCCAD (une "boîte") utilisé lors de la spécification est décrit par des fichiers comme l'illustre la figure 2.11 (1) et par des paramètres temporels visibles sur la figure 2.11 (2).

Une fois la tâche robot spécifiée, l'outil permet de générer automatiquement le code correspondant à l'architecture cible sur laquelle tourne le contrôleur. La fenêtre d'exécution et les architectures supportées sont visibles sur la figure 2.12.

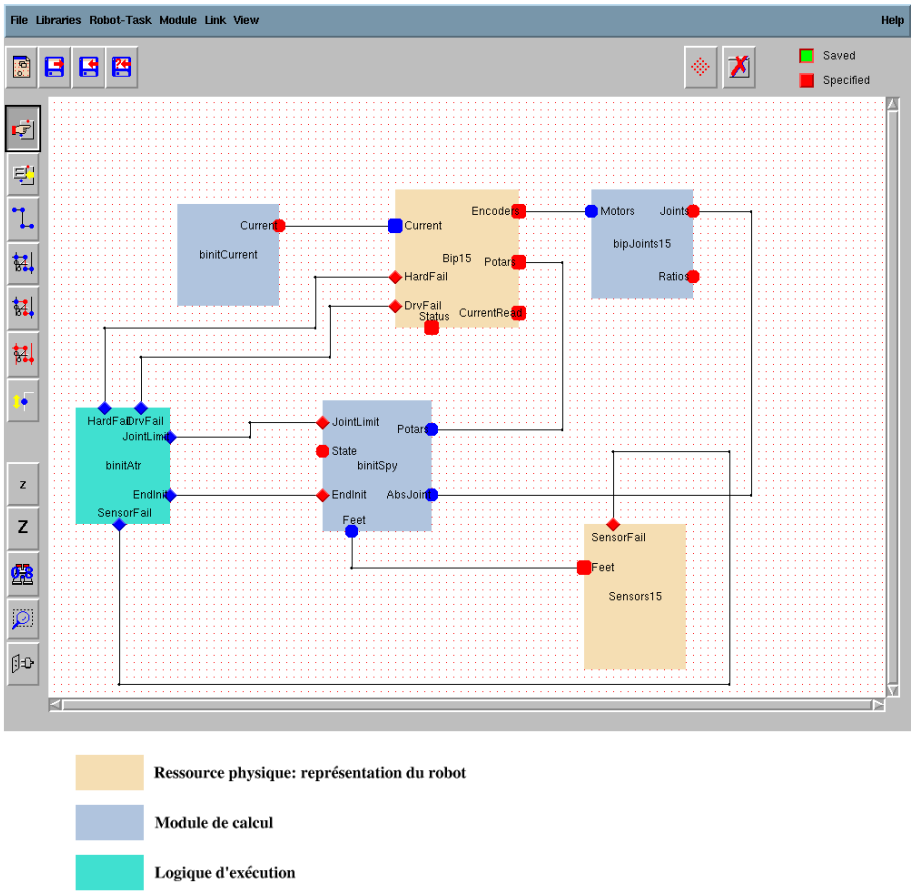


FIG. 2.10 – Une tâche robot sous ORCCAD qui implémente l'initialisation du robot bipède.

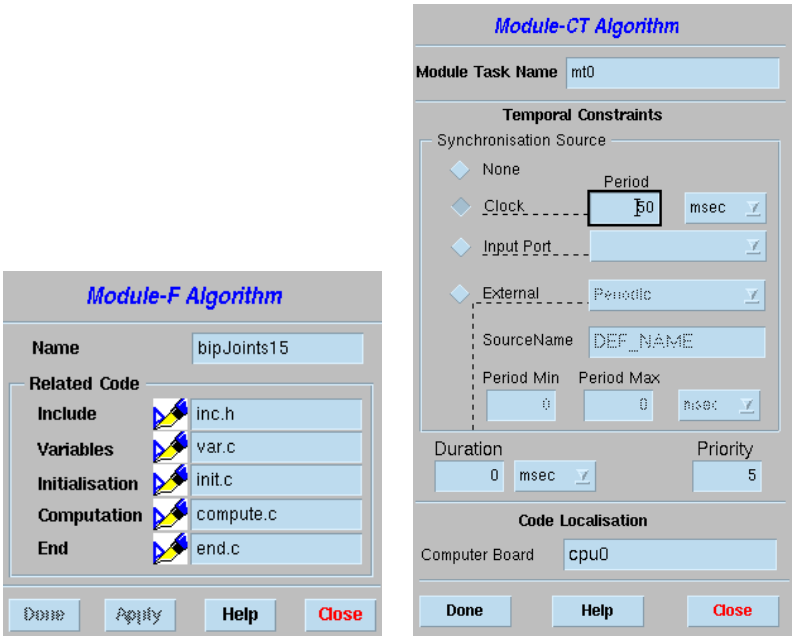


FIG. 2.11 – Spécification d'un module : (1) fichiers sources utilisés, (2) comportement temporel.

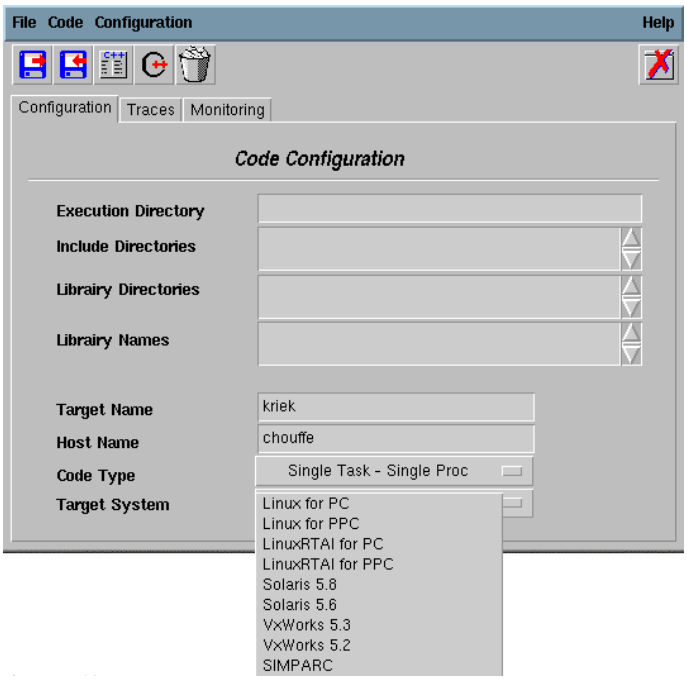


FIG. 2.12 – La fenêtre d'exécution qui permet la génération de code pour une plate-forme particulière.

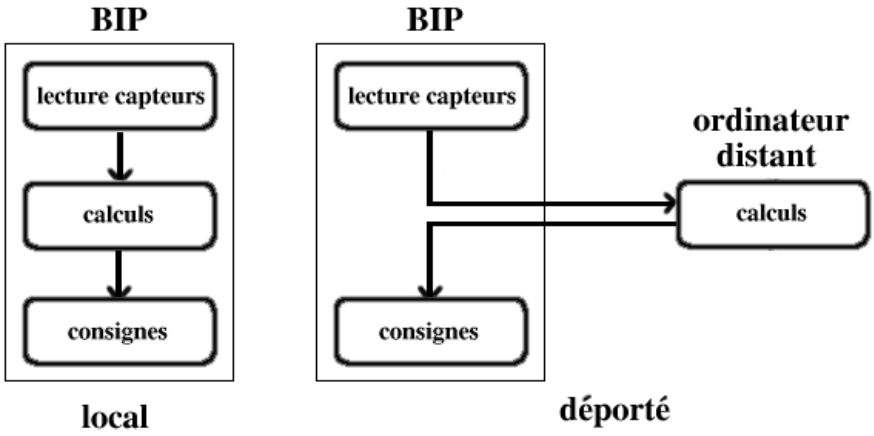


FIG. 2.13 – Les configurations calcul local/déporté.

2.5.3 Calcul en local / calcul déporté

Jusqu'à présent tous les calculs étaient exécutés sur la carte processeur embarquée sur le robot. Mais les nouvelles lois de commande mises au point pour commander le robot deviennent de plus en plus complexes et sont par conséquent de plus en plus demandeuses en puissance de calcul.
Une des solutions possibles consiste à déporter la partie calcul gourmande en ressources sur une machine distante plus puissante.

2.6 Une nouvelle carte processeur

Comme nous l'avons vu dans la présentation du contrôleur actuel, ce dernier n'est pas capable de fournir les ressources nécessaires à l'évolution future de BIP. C'est pourquoi, il a été décidé d'équiper BIP avec un nouveau calculateur embarqué.
Cette nouvelle carte, fabriquée par ACTIS Computer[13], est équipée d'un processeur Motorola PowerPC à 200MHz, de plus de mémoire et d'un contrôleur Ethernet à 100Mbit/s. Elle est livrée avec le système d'exploitation Linux.
Cette carte permet d'envisager des lois de commande en local plus complexes pour le robot, et sa connexion Ethernet 100Mbits ouvre également la voie à l'utilisation de calculs déportés avec des latences de communication réduites.

2.7 Le travail demandé

Etant donné les différences au niveau du système d'exploitation sur la nouvelle carte processeur il faut, dans un premier temps, porter les drivers et les librairies sous Linux et voir si Linux nous permet de contrôler le robot. Puis ensuite, ils devrons être portés sous Linux-RTAI pour voir ce qu'apporte les tâches temps réel offertes par ce système. Tout au long du développement, il faut faire en sorte que les drivers et les librairies soient encore utilisables sous VxWorks. Ce critère est important car le LMS[3], qui possède un deuxième exemplaire de BIP, utilise toujours l'ancienne carte processeur.

[3]Laboratoire de Mécanique des Solides de Poitiers

Chapitre 3

Portage des drivers et des librairies

Le portage des drivers et des librairies s'est fait en deux étapes. Dans un premier temps, comme Linux était livré avec la nouvelle carte processeur, j'ai réalisé le portage vers Linux. Une fois les drivers testés sous Linux, j'ai réalisé le portage pour Linux-RTAI.

3.1 Documentation et découverte de l'existant

L'équipe SED a mis à ma disposition des documents techniques sur le robot bipède et plus particulièrement sur l'armoire de commande. Ainsi, j'ai pu avoir un aperçu d'ensemble du robot. J'ai ensuite étudié la documentation fournie avec la nouvelle carte contrôleur. Cette première étape de documentation n'a duré qu'un ou deux jours. L'étude du fonctionnement des modules IP s'est fait, quand à elle, au fur et à mesure du portage des drivers.

3.2 L'environnement de travail

3.2.1 Le rack VME de test

Pour mettre à jour le contrôleur, je ne pouvais pas monopoliser le robot bipède, et surtout empêcher son utilisation pendant plusieurs mois. L'équipe SED a donc mis à ma disposition un rack VME visible sur la figure 3.1 et équipé de la nouvelle carte processeur, d'une carte VME support de modules IP et de modules IP identiques à ceux présents sur le robot.

3.2.2 Le robot bipède lui-même

Le rack de test est limité : il ne permet pas des tester les drivers dans les conditions réelles. Je ne disposais que d'un unique exemplaire de chaque module IP alors que le robot peut en utiliser plusieurs exemplaires. Et il aurait été trop compliqué de simuler les valeurs en provenance des capteurs.
Aussi, après avoir validé chaque driver individuellement pour commander un unique

FIG. 3.1 – Le rack VME utilisé lors du développement.

module IP, je suis passé sur le robot bipède pour réaliser des tests plus complets, en conditions réelles.

3.2.3 Les autres outils mis à ma disposition

Je disposais également d'une station de travail sous Linux et équipée de tous les outils de compilation croisée[1] adaptés à la nouvelle carte processeur.
J'ai utilisé une console série pour dialoguer avec la nouvelle carte.
Enfin, j'ai pu utiliser divers autres outils nécessaires à mon travail tels qu'un oscilloscope numérique ou encore une alimentation stabilisée.

3.3 Migration vers Linux

Les drivers sont des programmes très proches du matériel et sont par conséquent assez indépendants du système d'exploitation. Cette remarque est particulièrement vraie pour VxWorks. Les drivers sous VxWorks sont minimaux : ils contiennent le strict minimum pour assurer le fonctionnement des modules IP.
Note : à partir de maintenant, j'appelerai driver "natif" le driver tel qu'il existe pour l'ancienne carte processeur, et son interface de communication sera qualifiée d'"interface native" du driver.

3.3.1 Développement logiciel

La carte processeur était livrée avec tous les outils de développement ainsi qu'avec un noyau Linux compilé et opérationnel. Ma tâche principale a donc été le portage des drivers des modules IP depuis le système d'exploitation VxWorks vers Linux. Pour le portage des drivers, deux différences majeures entre ces systèmes d'exploitation ont du être considérées.

[1] La compilation croisée permet de compiler du code source pour une architecture différente de celle sur laquelle on développe. Dans mon cas, je développais sur une architecture Intel alors que la carte cible se base sur une architecture PowerPC.

1 - Accès au matériel

Chaque module IP possède une plage d'adresses d'entrée/sortie physiques pour communiquer avec le processeur. Sous VxWorks, l'accès à ces adresses physiques est très simple : il suffit de placer les adresses dans des pointeurs et de manipuler les registres des modules IP comme s'il s'agissait de variables du programme. Sous Linux et le processeur PowerPC, en revanche, cet accès direct n'est pas possible : il est empêché par le système. Il faut procéder à un "mapping" mémoire, c'est-à-dire que le système fait correspondre à l'adresse physique une adresse virtuelle, et c'est cette adresse virtuelle qui est manipulée par le programme. Ces modifications sont très simples et permettent de conserver un code facilement lisible et maintenable.
J'ai donc modifié les fonctions d'initialisation et de fermeture des drivers natifs. J'ai rajouté l'appel à la fonction de mapping mémoire dans la fonction d'initialisation du driver tandis que j'ai rajouté l'appel à la fonction de "de-mapping" mémoire dans la fonction de fermeture du driver.

2 - Communication avec les drivers

La seconde différence est liée à l'architecture de ces systèmes d'exploitation. Sous VxWorks, il n'existe pas de distinction entre l'espace utilisateur et l'espace noyau du système d'exploitation. L'utilisateur peut accéder à n'importe quelle fonction : il lui suffit de connaître son nom. Par contre sous Linux, il existe un espace utilisateur et un espace noyau très bien délimités. L'utilisateur, pour utiliser des fonctions du noyau, doit passer par des canaux de communication particuliers. Il existe trois moyens de communication entre l'utilisateur et le noyau :

- les appels systèmes
- le système de fichiers virtuel(vfs[2])
- les entrées `/proc`

Lorsque l'on écrit un driver pour l'espace noyau, on utilise généralement une communication avec l'utilisateur à l'aide de vfs. En effet, les appels systèmes nécessitent une modification du noyau, ce qui est peu pratique, et les entrées `/proc` sont utilisées pour recupérer des informations sur le système.
Pour la version Linux des drivers, j'ai choisi d'utiliser l'interface vfs. Le système vfs proposé par Linux permet de manipuler un driver comme s'il s'agissait d'un fichier. L'accès au driver se fait par l'intermédiaire de fonctions comme `open`, `close`, `read` ou encore `write`.

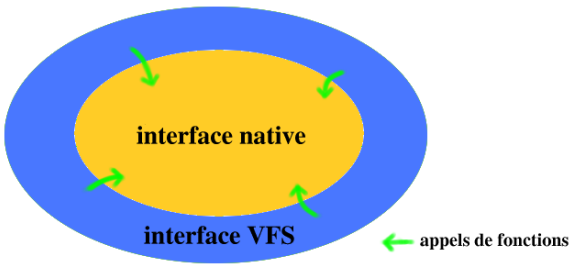


FIG. 3.2 – L'interface VFS utilise les fonctions de l'interface native du driver.

[2]Vitual File System, il s'agit d'une interface pour manipuler plusieurs types de systèmes de fichiers différents au travers d'une interface commune.

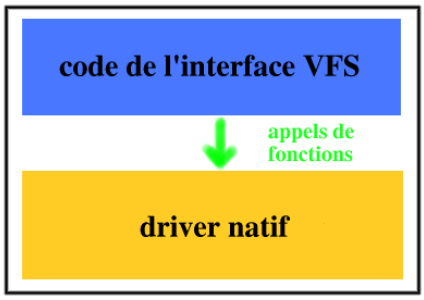


FIG. 3.3 – Les modifications pour VFS sont localisées et n'influent pas sur le code du driver natif.

Les drivers natifs ne proposent pas ces fonctions, mais des fonctionalités équivalentes. J'ai donc rajouté, pour la version Linux, les fonctions de l'interface vfs. Comme l'illustre la figure 3.2, ces nouvelles fonctions utilisent les fonctions du driver natif. Cette solution permet de grouper les modifications : voir figure 3.3 et de conserver un code lisible. De même, puisque tout le code de dialogue avec les modules IP est dans les drivers natifs, une mise à jour de ce dialogue est automatiquement prise en compte aussi bien par les drivers en version VxWorks qu'en version Linux. La maintenance des drivers est facilitée.
A présent, tous les drivers des modules IP sont au niveau noyau du système d'exploitation tandis que les couches logicielles supérieures sont au niveau utilisateur.

Les adaptateurs

L'introduction de l'interface vfs sous Linux modifie l'interface d'accès aux drivers du point de vue utilisateur : il ne manipule plus l'interface native mais l'interface vfs. Dans un premier temps, j'ai commencé par modifier la couche logicielle `hardBip` en conséquence, mais les modifications étaient nombreuses et le code devenait plus difficile à lire. Certains bouts de code étaient dupliqués puis légèrement modifiés pour la version Linux. La maintenance du logiciel devenait beaucoup plus complexe. J'ai donc cherché une approche différente qui minimise les changements à apporter aux couches logicielles supérieures.
Les adaptateurs ont apportés une réponse à ce problème. Il s'agit de couches logicielles supplémentaires que j'ai créées et qui s'intercalent entre les drivers et la couche `hardBip`. Ces adaptateurs sont au niveau utilisateur et transforment l'interface vfs pour lui redonner l'interface des drivers natifs. Ils effectuent en fait l'opération inverse de celle qui a lieu au niveau du noyau. Cette approche permet de conserver quasiment à l'identique le code des librairies utilisées pour l'ancienne carte processeur. La figure 3.4 permet de situer les adaptateurs dans l'organisation logicielle.

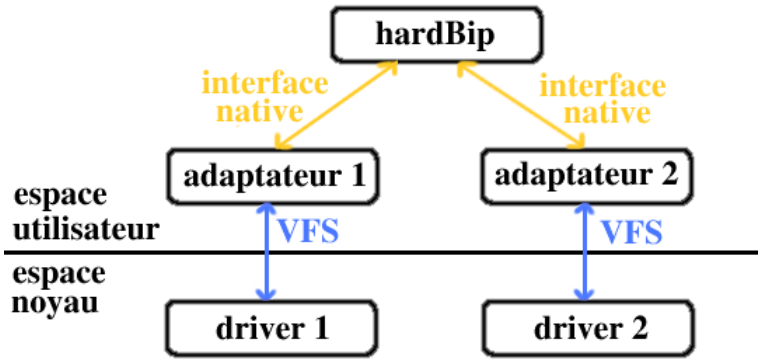


FIG. 3.4 – La place des adaptateurs sous Linux.

3.3.2 Tests des nouveaux drivers

Une fois les nouveaux drivers développés, il fallait s'assurer qu'ils fonctionnaient encore sous VxWorks puis faire quelques tests de performance.

Tests sous VxWorks

L'une des contraintes lors du portage des drivers et des librairies était de conserver la compatibilité des nouveaux drivers et des nouvelles librairies avec les anciens : le résultat du portage devait pouvoir être compilé et fonctionner pour la cible 68040-VxWorks. C'est donc le premier test que j'ai effectué.
Les nouveaux drivers et librairies ont été compilés pour l'ancienne carte processeur puis ont été testés sur le robot. Au premier essai, tous les drivers à l'exception de celui pour le module IP tip501 de conversion analogique vers numérique ont fonctionné correctement.
Le driver pour le module IP tip501 est celui qui a subi le plus de modifications. A l'origine, ce driver avait une interface différente de celle des autres drivers. Nous avons donc décidé, dans un soucis de simplicité, de lui donner une interface semblable à celle des autres modules IP. Or, lors de cette modification d'interface, j'ai nettoyé le code et j'en ai enlevé un peu trop. Une ligne qui me semblait inutile s'est avérée dans les faits indispensable. Il s'agissait d'un cast sur un pointeur. Une fois cette ligne réintégrée dans le nouveau driver, nous avons relancé les tests et cette fois tout a fonctionné.
Objectif atteint : les nouveaux drivers fonctionnent toujours sur l'ancienne plateforme.

Tests de performance des drivers

Comme la nouvelle carte processeur utilise un autre système d'exploitation et surtout qu'elle est plus puissante, il est intéressant de comparer les nouvelles performances avec les anciennes.
J'ai mesuré les performances de chaque driver séparément à l'aide de petits programmes de test. Le principe est d'effectuer dix mille opérations sur les drivers et d'en extraire le temps moyen d'exécution pour une opération. Par exemple, pour la carte de conversion analogique/numérique, il s'agit de lire la valeur des seize entrées analogiques. Les résultats des mesures sont présentés dans le tableau 3.5.

module/système d'expoitation	VxWorks	Linux
IP ADC	600μs	535μs
IP DAC	100μs	93μs
IP quadrature	50μs	40μs

FIG. 3.5 – Résultat des tests de performance sur VxWorks et sous Linux-PowerPC.

Nous constatons que les temps sont très peu différents. Ces résultats sont prévisibles puisque lors des accès aux modules IP, ce n'est pas la vitesse du processeur qui limite, c'est la vitesse des modules IP. Par exemple, le temps de conversion d'une voie analogique en une valeur numérique nécessite 10μs. Ce temps est incompressible puisque c'est une caractéristique intrinsèque du convertisseur utilisé par le module IP. Ensuite, tous les modules IP utilisés fonctionnent avec un bus à 8MHz. Les modules IP et le bus sont donc des goulots d'étranglement indépendants de la vitesse du processeur.
Néanmoins, puisque notre processeur est plus rapide, il est tout de même capable d'exécuter plus d'instructions par seconde, il peut donc mener à bien des calculs plus conséquents. Cette carte possède également une interface Ethernet 100Mbits/s, ce qui est un plus non négligeable pour l'option calcul déporté qui est envisagée pour disposer de plus de puissance de calcul.
Nous avons ensuite fait des tests sur le robot équipé de la nouvelle carte processeur. Aucun problème majeur n'a été rencontré.

Tests sur la gestion des tâches par Linux

Il est prévu d'utiliser RTAI car les lois de commande sont calculées selon des contraintes temporelles fortes. Mais nous avons tout de même voulu savoir si Linux, même s'il ne propose aucune garantie quand aux temps de réponse des tâches périodiques, pouvait être utilisé pour contrôler le robot.
Dans un premier temps, les tests ont été effectués avec un seul processus. Ce processus change toutes les 10 millisecondes la valeur d'une sortie du module IP d'entrée/sortie numérique. La valeur de 10ms n'est pas choisie au hasard : c'est la période utilisée par la boucle de contrôle sur le robot. La sortie du module IP est observée sur un oscilloscope numérique qui nous permet d'effectuer des mesures précises sur le signal observé. Le dispositif expérimental est visible sur la figure 3.6. Les résultats obtenus sont plutôt bons puisque la période de réveil du processus de 10ms est très bien respectée.
Ensuite, j'ai fait la même expérience avec en plus trois processus concurrents. La période est également respectée mais elle est un peu moins stable (voir figure 3.7).

Cette instabilité est toutefois raisonnable et nous pouvons supposer que Linux pourrait permettre de commander correctement le robot.
Ces tests m'ont également permis de vérifier que la période de 10ms est la valeur minimale de granularité pour l'ordonnancement des processus sous Linux standard. Toute période inférieure est par défaut changée à 10ms.

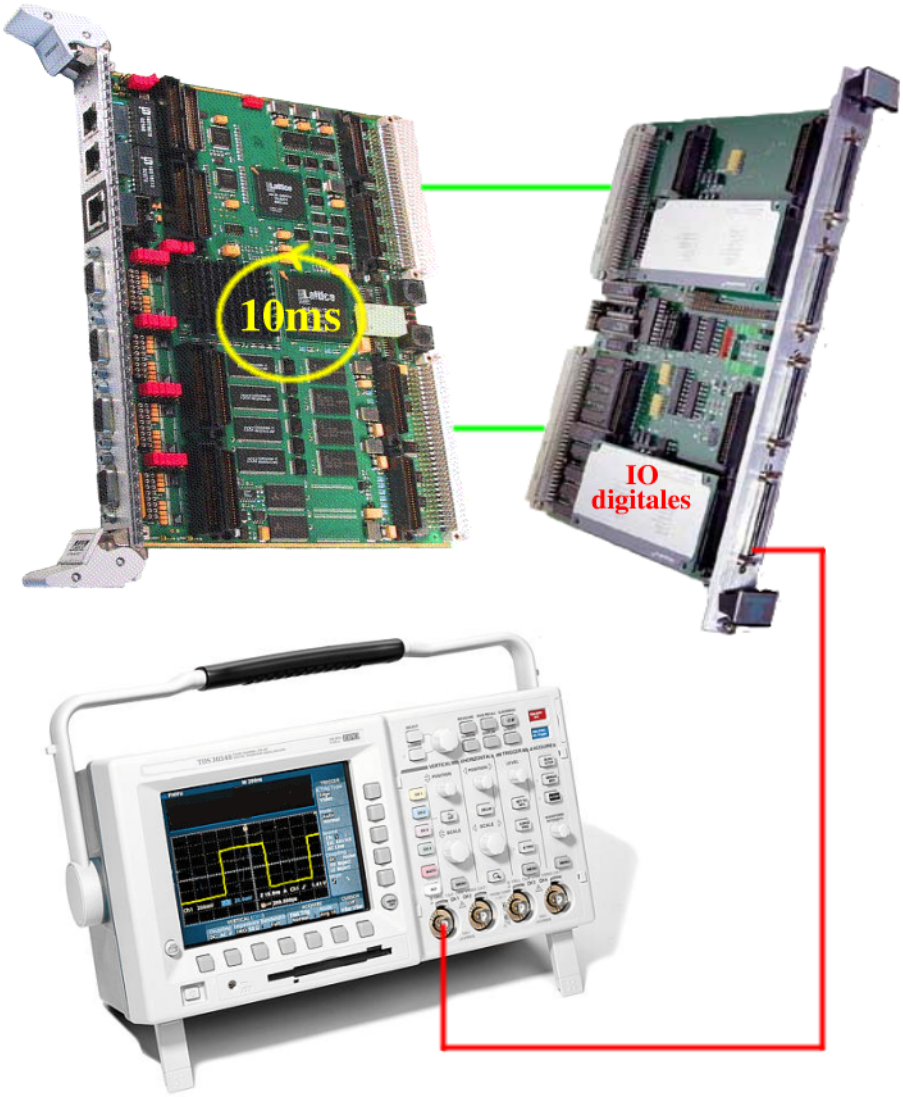
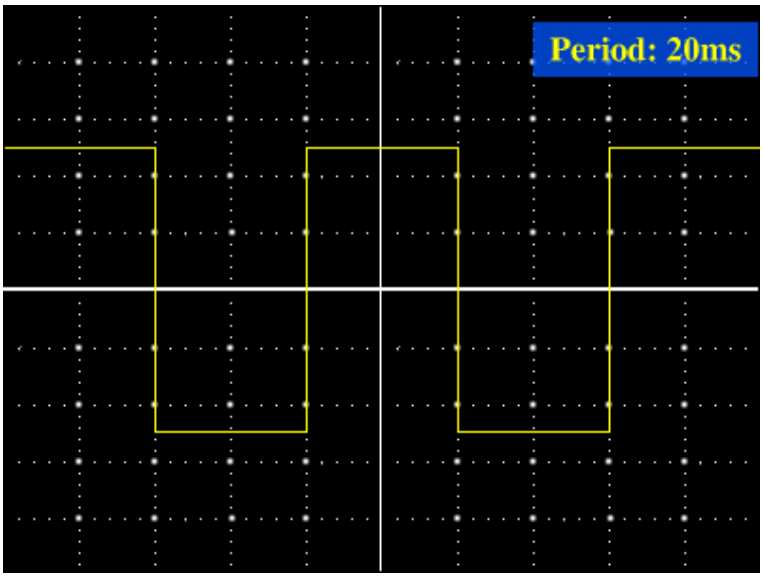


FIG. 3.6 – Dispositif de mesure du respect des contraintes temporelles sous Linux.

L'écran de l'oscilloscope affiche le graphique suivant :



Lorsque un unique processus s'exécute (celui qui produit le signal affiché sur l'oscilloscope) la période d'exécution du processus est bien respectée. Nous constatons que la période mesurée par l'oscilloscope est de 20ms, ce qui correspond bien à deux réveils consécutifs du processus espacés de 10ms. Cette période varie entre 19ms et 21ms, mais les excursions en période sont rares.

Lorsque trois processus concurrents parasites supplémentaires sont exécutés, la période reste à 20ms. En revanche, les excursions en période sont plus fréquentes et sont compris entre18ms et 22ms.

FIG. 3.7 – Résultats des tests sur la gestion des tâches par Linux standard.

3.4 Migration vers RTAI

RTAI[2] est un système d'exploitation temps réel : il permet de respecter de contraintes temporelles fortes pour l'exécution de processus. Il utilise Linux pour profiter de toutes ses fonctionnalitées et de tous ses programmes. Nous aurions pu utiliser RT-Linux[3], dont la philosophie est identique, mais sa licence est plus ou moins commerciale contrairement à celle de RTAI qui est libre. Lorsque Linux est patché pour RTAI, le noyau Linux n'est plus la base du système : il devient un processus géré par RTAI. Cette situation est illustrée par la figure 3.8. RTAI gère les processus temps réel et le noyau Linux. Mais ce dernier possède la priorité la plus faible : il s'exécute uniquement si il reste du temps processeur disponible.
Les processus RTAI sont dans l'espace noyau et sont créés à l'aide de modules[3] qui sont chargés dans le noyau. Un module peut voir et appeller les fonctions et les variables d'un autre module chargé.

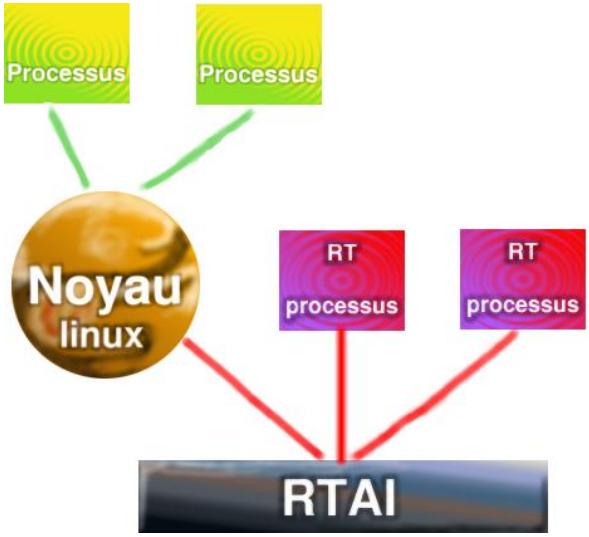


FIG. 3.8 – Architecture de RTAI.

3.4.1 Installation de RTAI

La carte processeur est livrée avec Linux mais pas avec RTAI. Il a donc été nécessaire d'installer ce dernier. Sa compilation et son installation se sont relativement bien déroulées grâce à la documentation fournie avec les sources. J'ai tout de même rencontré un bug, ce qui m'a amené à modifier légèrement les sources. RTAI utilisait un symbole qui n'était pas exporté par le noyau Linux, j'ai fait en sorte que RTAI utilise un symbole exporté et dont la fonctionnalité est identique.
J'ai également rédigé un petit guide d'installation ainsi qu'un patch pour reproduire la modification que j'ai apporté aux sources RTAI.

[3]un module est un programme crée pour s'exécuter dans l'espace noyau du système d'exploitation

3.4.2 Développement logiciel

Comme nous l'avons vu au chapitre 2.5.1, la librairie `drvBip` contient les routines utilisateur d'accès aux drivers du robot, elle est l'unique librairie utilisée par la partie contrôle.
Puisque la partie contrôle présente un fonctionnement périodique elle sera implémentée sous la forme d'un processus RTAI. Il est alors logique de mettre les couches logicielles `hardBip` et `drvBip` au niveau RTAI. Mais un deuxième cas de figure est possible : `drvBip` peut être utilisée par un processus non périodique. C'est la cas par exemple du programme de test `menuBip`. Dans ce cas, il semble plus logique d'exécuter ce programme en tant que processus Linux standard. `drvBip` doit alors être visible depuis le processus Linux.
Finalement, nous avons choisi de mettre `hardBip` sous la tutelle de RTAI tandis que `drvBip` peut être compilé aussi bien pour fonctionner avec un processus Linux standard qu'avec un processus RTAI. L'organisation retenue est visible sur la figure 3.9.

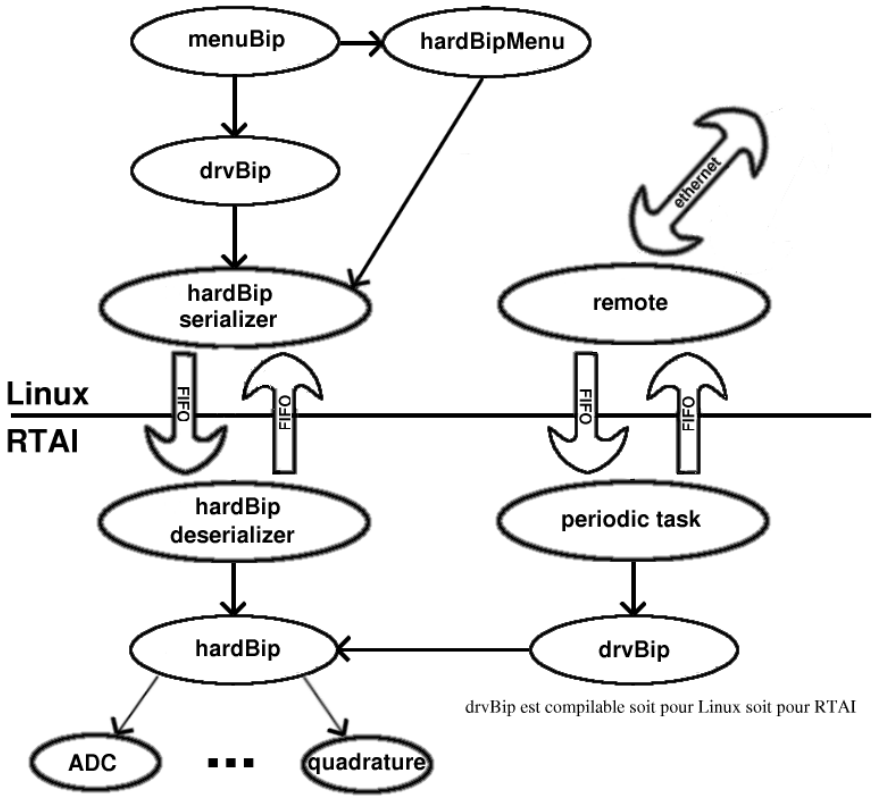


FIG. 3.9 – Architecture logicielle sous RTAI.

La librairie `hardBip`, qui se présente sous forme d'un module sous RTAI est par conséquent dans l'espace noyau. Elle peut appeler directement les fonctions des drivers : nous n'avons plus besoin de l'interface vfs ni des adaptateurs. Par contre, le problème d'accès aux adresses physiques reste identique : le "mapping" mèmoire est conservé.
Dans le cas où `drvBip` est utilisée par un processus Linux standard, aucune modification par rapport à la version Linux écrite au chapitre 3.3 n'est nécessaire. En revanche, pour que `drvBip` puisse être utilisée par un processus RTAI, il faut la convertir en un module. J'ai donc rajouté les fonctions d'initialisation et de fermeture d'un module et je l'ai compilé avec les options appropriées.
Cependant, un problème de communication subsiste dans le cas où `drvBip` est utilisée par un processus Linux standard. Dans ce cas, `drvBip` est dans l'espace utilisateur tandis que `hardBip` est dans l'espace noyau et sous la tutelle de RTAI. Nous ne pouvons pas cette fois encore utiliser l'interface vfs car RTAI ne la propose pas. RTAI propose toutefois d'autres mécanismes spécifiques pour la communication entre des processus Linux standard et des processus RTAI temps réel. Parmis ces mécanismes nous avons choisit le plus simple d'entre-eux : les FIFOs[4].
Le système de communication basé sur les FIFOs RTAI est composé des couches logicielles `hardBip_serializer` et `hardBip_deserializer`.
`hardBip_serializer` émule la couche `hardBip` : elle propose la même interface. Mais au lieu d'appeler les fonctions des drivers, comme le fait `hardBip`, elle transforme les ordres qu'elle reçoit pour les faire passer au travers d'une FIFO. `hardBip_deserializer` effectue l'opération inverse : elle lit les ordres dans la FIFO et transforme ces ordres en appels de fonctions de `hardBip`. Les valeurs de retour des ces appels de fonctions sont transmises au niveau Linux par l'intermédiaire d'une deuxième FIFO.

3.4.3 Tests

Les tests ont dans un premier temps été effectués sur le banc de test présenté au chapitre 3.2.1, module par module, à l'aide du programme `menuBip`. Une fois que nous avons constaté que chaque module fonctionnait de façon indépendante, nous avons procédé aux tests sur le robot avec tous les modules installés. Ici encore, aucun problème particulier ne s'est présenté.

[4]First In First Out : structure de donnée où les données sont lues dans l'ordre où elles ont été écrites

Chapitre 4

Changement du bootloader

Une fois que les drivers et les librairies ont été portés sous Linux, puis ensuite sous RTAI (voir chapitre 3), le robot est devenu utilisable avec la carte processeur PowerPC. Mais la nouvelle carte processeur ne proposait pas une fonctionnalité disponible sur l'ancienne version : le boot automatiqe. La fonction de boot est assurée par un bootloader.
Le bootloader est le premier programme lancé par la carte processeur. C'est lui qui est chargé de configurer la carte puis de lancer le système d'exploitation en lui fournissant les bons paramètres.

4.1 ECMON

La carte processeur PowerPC était livrée avec un bootloader du nom d'ECMON[5]. Je l'ai utilisé lors de la première phase du développement et des tests (chapitre 3). Avec ce bootloader l'utilisateur est obligé de lancer au moins une commande sur la console série reliée à la carte pour lancer le système d'exploitation.

CAS 1 :	**Le noyau est sur un ordinateur distant :** 1. Charger le noyau Linux en mémoire à l'aide de l'interface réseau 2. Lancer le noyau (il faut passer l'adresse mémoire où le noyau a été chargé)
CAS 2 :	**Le noyau est dans la mémoire Flash[1] de la carte processeur :** 1. Lancer le noyau (il faut connaître l'adresse du noyau en mémoire Flash)

[1] Le mémoire Flash est une mémoire non volatile, c'est-à-dire qu'elle conserve ses données même lorsque son alimentation électrique est coupée.

FIG. 4.1 – Lancer un système d'exploitation avec ECMON.

Or, il est intéressant de disposer d'un système capable de booter automatiquement, comme le fait un ordinateur personnel. Le boot automatique permet également de se passer de la console série, inutile en dehors de la phase de démarrage.

ECMON propose une option de boot automatique mais seulement sur sa version commerciale. La version d'ECMON livrée avec la carte processeur est une version allégée qui ne propose pas toutes les fonctionnalitées de la version complète et commerciale. Le boot automatique fait partie des fonctionnalités supprimées. J'ai donc cherché sur Internet un bootloader susceptible de fonctionner avec notre carte et offrant la fonctionnalité de boot automatique.

4.2 u-boot

Cette recherche m'a conduit à trouver un bootloader du nom de u-boot[6]. Il a été initialement développé pour les cartes à base de processeur PowerPC mais par la suite il a été étendu à d'autres types de processeurs. Il s'agit d'un logiciel libre dont les sources sont disponibles.
Notre carte processeur ne faisait pas partie des cartes supportées en standard par u-boot. Je l'ai donc porté pour notre carte processeur. La documentation fournie avec la carte processeur PowerPC comporte toutes les informations nécessaires pour configurer le processeur PowerPC, même si elles étaient parfois erronées. Mais en comparant les valeurs de configuration écrites par ECMON et celles écrites par ma version d'u-boot, j'ai pu corriger les données incorrectes. Le portage a été assez rapide puisqu'il m'a suffit de trois jours pour porter le code et le tester.
En revanche, j'ai rencontré plus de difficultés avec le bus VME : je ne pouvais pas effectuer des accès sur le bus supérieur à huit bits. Or, mes modules IP utilisent un bus seize bits. Il était hors de question de laisser les choses en l'état car j'ai constaté que mes temps d'accès aux modules IP étaient considérablement allongés. J'ai donc comparé très précisément toutes les valeurs écrites par ECMON pour trouver la valeur qu'il me manquait. Il s'agissait d'un problème de configuration d'un registre de la carte processeur elle-même et non pas d'un registre interne au processeur. Ce problème, à lui tout seul, m'a occupé près de deux jours complets.
A présent, u-boot fonctionne sur notre carte processeur PowerPC et nous pouvons booter automatiquement le noyau du système d'exploitation.

Chapitre 5

Travail sous ORCCAD

5.1 Passage des tâches ORCCAD sous Linux PowerPC

Pour que le robot puisse fonctionner avec la nouvelle carte PowerPC, il faut également porter la partie contrôleur réalisée à l'aide du logiciel ORCCAD (voir chapitre 2.5.2). Les modifications à apporter devraient être minimes grâce aux adaptateurs vus au chapitre 3.3.1.

5.1.1 La tâche d'initialisation

Avant que BIP ne puisse être utilisé, il faut l'initialiser et pour ce faire, il faut déterminer dans quelle position il se trouve. Cette initialisation, sous VxWorks, peut être effectuée soit par un programme indépendant, soit par un programme généré sous ORCCAD. Ces deux solutions ont été portées sous Linux mais je parlerai ici de la tâche robot générée par ORCCAD car par la suite seules de telles tâches seront portées.
Cette phase d'initialisation est obligatoire avant toute utilisation de BIP, elle constitue la tâche de base. Elle est également très simple et est par conséquent une très bonne candidate pour un premier essai du robot bipède sous Linux.
Comme prévu, le portage s'est révélé très simple. Les seules modifications ont été apportées dans la fenêtre "exécution" d'ORCCAD. La spécification de la tâche et le code des "boîtes" n'ont pas été modifiés. Seuls les paramètres du générateur de code ont été changés, ils sont visibles sur la figure 5.1.

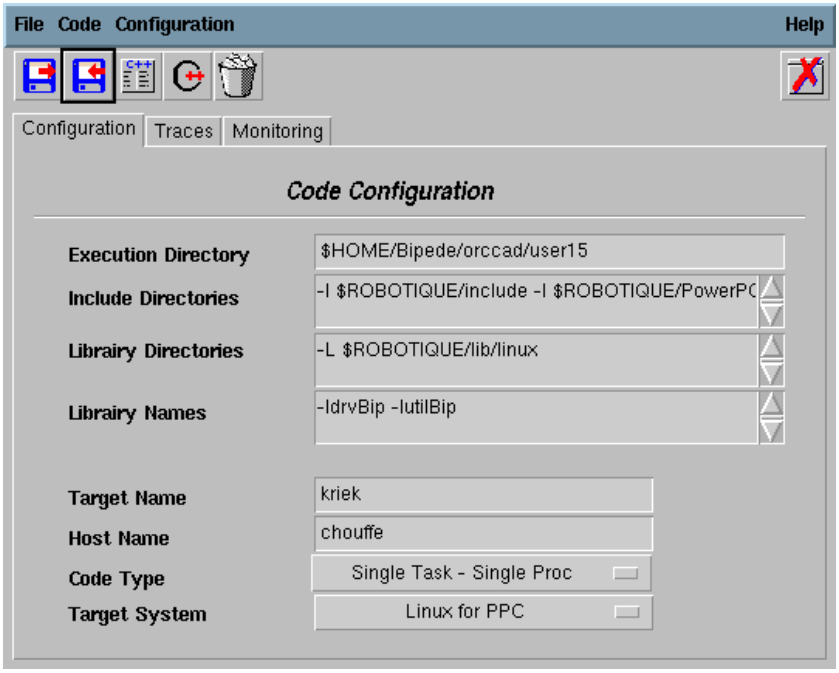


FIG. 5.1 – Les paramètres de la fenêtre d'exécution, sous ORCCAD, pour la tâche d'initialisation sous Linux PowerPC.

5.1.2 La tâche de suivi de trajectoire

Présentation de la tâche

Il s'agit d'une tâche qui lit un ou plusieurs fichiers de trajectoires et qui anime les jambes de BIP en fonction de ces trajectoires. Elle est composée de quatres étapes principales :

1. l'initialisation
2. le chargement des trajectoires à jouer
3. l'exécution de la boucle de contrôle de suivi de trajectoire
4. une sauvegarde des données de l'expérience

Sous VxWorks, l'exécution de cette tâche se déroule en deux étapes. Dans un premier temps, l'initialisation est effectuée à l'aide d'un programme indépendant. Ce dernier positionne les valeurs d'initialisation dans une structure de données particulière utilisée ensuite pour la deuxième phase. Dans un second temps, la tâche de contrôle générée avec ORCCAD est exécutée. Elle charge les trajectoires à jouer puis lance la boucle de contrôle de suivi de trajectoire.

Portage sous Linux

Ce mode de fonctionnement, découpé en deux étapes, est possible sous VxWorks car toute librairie ou programme peut utiliser les fonctions et les variables d'un autre. Sous Linux, en revanche, les choses sont différentes. Si l'approche utilisée sous VxWorks est conservée sous Linux, nous obtenons deux processus : un pour l'initialisation et un autre pour la tâche générée avec ORCCAD. Or sous Linux, deux processus ne peuvent pas partager des données sauf au moyen de mécanismes spécifiques. Ainsi, la structure de données mise à jour lors de la phase d'initialisation n'est pas visible par la tâche de suivi de trajectoire : BIP ne bouge pas.
Pour Linux, j'ai légèrement modifié la tâche de suivi de trajectoire. Pour que la structure de données d'initialisation soit partagée par les deux phases, elles doivent avoir lieu dans le même processus. Ceci est obtenu en appelant la fonction d'initialisation de BIP dans la tâche robot, avant de lancer le suivi de trajectoire. J'ai fait cet ajout dans le fichier `init.c` de la spécification de la "boîte" `bip15` qui représente le robot bipède (voir figure 2.11 (1)).
Cette fois, deux interventions ont été nécessaires :

- la modification de la spécification de la "boite" `bip15`
- le menu "exécution" subit les mêmes changements que ceux présentés au paragraphe 5.1.1.

Mais ces changements demeurent toutefois très légers.

5.2 Passage des tâches ORCCAD sous RTAI PowerPC

Pour RTAI, seule la tâche de suivi de trajectoire a été portée. Les adaptateurs ne sont plus nécessaires comme nous l'avons vu au chapitre 3.4.2. En revanche, d'autres problèmes se posent.
La phase d'initialisation utilise un dialogue homme/machine puisqu'elle a besoin de connaître des informations que seul l'utilisateur peut lui fournir. La tâche robot doit également être capable de lire des fichiers pour charger les trajectoires à jouer. Or, sous RTAI, il n'existe pas de fonctions pour intéragir avec l'utilisateur ni pour lire des fichiers.
J'ai donc déplacé toute la communication homme/machine et la lecture des fichiers au niveau utilisateur sous Linux. Cette partie correspond au programme d'initialisation indépendant sous VxWorks. Le programme Linux utilisateur remplit la structure de données d'initialisation, mais il faut à présent passer cette structure à la boucle de contrôle générée par ORCCAD qui est une tâche RTAI. Nous retrouvons la problématique de la communication entre un processus Linux et une tâche RTAI déjà vue au chapitre 3.4.2. Cette fois encore, nous faisons appel aux FIFOs. Lorsque la structure de données d'initialisation est remplie, nous l'envoyons au travers d'une FIFO à destination d'un module RTAI. Ce module recopie la valeur de la structure de données reçue par la FIFO dans celle utilisée par la boucle de contrôle. Enfin, le module RTAI lance la tâche de suivi de trajectoire. Cette organisation est visible sur la figure 5.2 et le cheminement de la structure de données d'initialisation est visible à la figure 5.3.
La spécification de la tâche robot sous ORCCAD utilisée est la même que celle utilisée avec l'ancien calculateur sous VxWorks. Seul le paramètre concernant l'architecture cible pour la génération de code est changé. Il n'y a pas ici de réelles modifications, il s'agit surtout de l'ajout de composants. J'ai créé le programme Linux utilisateur de dialogue homme/machine et le module RTAI de réception des données par la FIFO qui

doit être chargé dans le noyau avant d'exécuter le programme utilisateur.

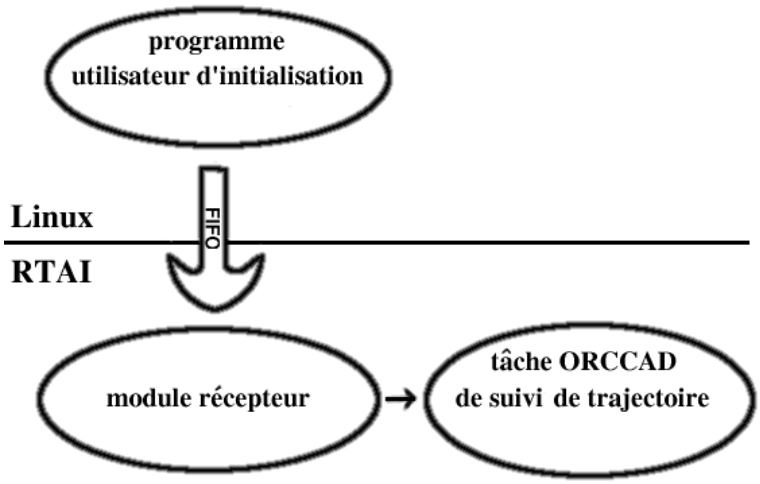


FIG. 5.2 – Organisation de la tâche de suivi de trajectoire sous RTAI

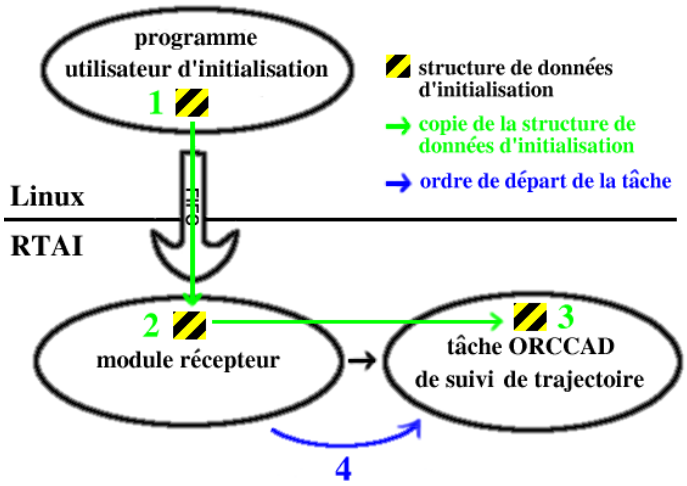


FIG. 5.3 – Chronogramme des opérations pour le suivi de trajectoire sous RTAI

Chapitre 6

Planning du stage

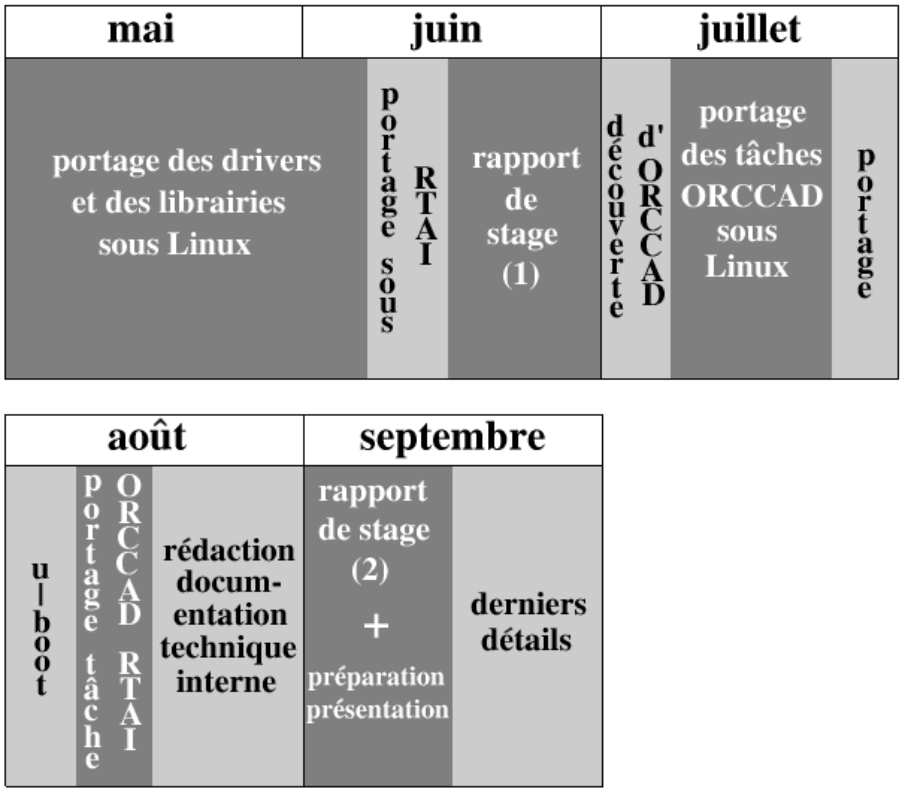


FIG. 6.1 – Le planning du stage

Dans un premier temps, de début mai jusqu'à la mi-juin, je me suis focalisé sur le portage des drivers et des librairies sous Linux puis sous RTAI.
J'ai ensuite rédigé une première partie du rapport de stage basée sur cette première phase de travail.
Ensuite, afin de vérifier que mon travail fonctionnait correctement sous Linux et pour

pouvoir jouer des trajectoires, je me suis familiarisé avec l'outil ORCCAD. Une fois l'outil en main j'ai porté des tâches ORCCAD pour la nouvelle carte processeur sous Linux.
A la fin du mois de juillet, le besoin s'est fait sentir d'un système de boot automatique pour la carte PowerPC. Mais comme ECMON, le bootloader original, n'en est pas capable, je me suis attelé au portage de u-boot.
Le portage de la tâche ORCCAD de suivi de trajectoire sous RTAI a été la dernière étape du développement.
Enfin, j'ai rédigé un rapport technique pour le service SED et j'ai finalisé la rédaction de mon rapport de stage lors du mois de septembre.

Chapitre 7

Conclusion

La plateforme BIP est en pleine évolution. Cette évolution nécessite des ressources en calcul et en communication supplémentaires. La nouvelle carte processeur est capable de fournir ces ressources. Mon travail a été de migrer tous les logiciels utilisés par l'ancien calculateur embarqué. J'ai tenté tout au long de mon travail de conserver le maximum de code existant et la compatibilité avec les outils utilisés auparavant. Ceci afin de ne pas perdre le travail déjà effectué depuis plusieurs années et afin que l'exemplaire de BIP à Poitiers soit toujours fonctionnel. Les contraintes sur compatibilité, sur la minimisation des modifications et sur la facilté de maintenance ont été délicates à atteindre mais la solution trouvée me semble convaincante.
Le nouveau calculateur embarqué est une brique essentielle de l'évolution de BIP. A présent, il peut gérer un plus grand nombre de capteurs, il peut utiliser des lois de commande plus complexes. Enfin, la liaison Ethernet haut débit ouvre la voie à l'utilisation de calculs déportés sur des machines encore plus puissantes et donc des lois de commandes ou des traitements de signaux encore plus élaborés.
Ce projet m'a également permis de me confronter à plusieurs niveaux d'abstraction. J'ai ainsi manipulé le niveau matériel lors du portage des drivers des modules IP et du bootloader. Je suis ensuite monté au niveau du système d'exploitation lors du portage du bootloader, des librairies et des tâches ORCCAD. Mais j'ai également vu un aperçu du niveau contrôle lors de la manipulation de l'outil ORCCAD.
Enfin, cette expérience dans un laboratoire de recherche m'a permis de rencontrer de nombreuses personnes motivées par leurs projets respectifs et disponibles pour expliquer leur travail et leurs recherches.

Bibliographie

[1] Site internet de l'éditeur de VxWorks : `http://www.windriver.com`

[2] Site internet de RTAI
`http://www.aero.polimi.it/projects/rtai/`

[3] Site internet de FSMLabs - la société qui développe RTLinux.
`http://www.fsmlabs.com`

[4] *Mise à jour du contrôleur du robot bipède BIP.*
Document technique INRIA, de FUCHET Jérôme

[5] Site internet de la société ECRIN Systems - créatrice du bootloader ECMON :
`http://www.ecrin.com`

[6] Site internet du bootloader u-boot :
`http://sourceforge.net/projects/u-boot/`

[7] Site internet d'ORCCAD :
`http://www.inrialpes.fr/iramr/pub/Orccad/`

[8] *L'armoire de commande du robot bipède bip2000*
Par : Gérard Baille, Pascal Di Giacomo, Hervé Mathieu et Roger Pissard-Gibollet
Rapport technique INRIA n° 0243, juillet 2000

[9] VSBC-6862 User's Guide
Manuel utilisateur fournit avec la carte ACTIS vsbc-6862.

[10] Guide utilisateur du processeur PowerPC MPC8260.
Site internet de motorola : `http://e-www.motorola.com`
Le fichier s'appelle `MPC8260UM`.
A l'heure où j'écris ces lignes il est accessible à l'adresse :
`http://e-www.motorola.com/files/issue_files/cat_not_blade/MPC8260UM_D.pdf`

[11] *Linux device driver*
De Alessandro Rubini et Jonathan Corbet
Edition O'REILLY, version électronique disponible à l'adresse :
`http://www.xml.com/ldd/chapter/book`

[12] Site internet du projet BIP :
`http://www.inrialpes.fr/bipop/resultats`

[13] Site internet de ACTIS Computer - constructeur de la nouvelle carte processeur
`http://www.actis-computer.com`

[14] Site internet de SBS Technologies - constructeur de modules IP
`http://www.sbs.com`

[15] Site internet de TEWS Technologies - constructeur de modules IP
`http://www.tews.com`

Résumé

L'INRIA Rhône-Alpes et le Laboratoire de Mécanique des Solides de l'université de Poitiers (LMS) ont développés, dans le cadre d'un projet commun, un robot bipède doté de quinze degrés de liberté. Ce robot, BIP2000, est capable de marcher sur un sol plat et de monter des marches d'escaliers.

L'équipe SED souhaite équiper le robot bipède d'un nouveau contrôleur. Mon travail consistait à porter les drivers et les librairies existants sur ce nouveau contrôleur. Il s'agissait de passer d'un système sous Motorola 68040 et VxWorks^{TM} à un système Motorola PowerPC^{TM} sous Linux puis RTAI.