Titre: Title:	Conception d'une architecture logicielle pour le positionnement au niveau atomique d'instruments scientifiques sous forme de robots miniatures pour des applications en nanotechnologies
Auteur: Author:	Dominic St-Jacques
Date:	2004
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	St-Jacques, D. (2004). Conception d'une architecture logicielle pour le positionnement au niveau atomique d'instruments scientifiques sous forme de robots miniatures pour des applications en nanotechnologies [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8404/

UNIVERSITÉ DE MONTRÉAL

CONCEPTION D'UNE ARCHITECTURE LOGICIELLE POUR LE POSITIONNEMENT AU NIVEAU ATOMIQUE D'INSTRUMENTS SCIENTIFIQUES SOUS FORME DE ROBOTS MINIATURES POUR DES APPLICATIONS EN NANOTECHNOLOGIES.

DOMINIC ST-JACQUES
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
AOÛT 2004

Your file Votre référence
ISBN: 978-0-494-47687-1
Our file Notre référence
ISBN: 978-0-494-47687-1

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

CONCEPTION D'UNE ARCHITECTURE LOGICIELLE POUR LE POSITIONNEMENT AU NIVEAU ATOMIQUE D'INSTRUMENTS SCIENTIFIQUES SOUS FORME DE ROBOTS MINIATURES POUR DES APPLICATIONS EN NANOTECHNOLOGIES.

présenté par: ST-JACQUES Dominic

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. GUIBAULT François, Ph.D, président

M. MARTEL Sylvain, Ph.D, membre et directeur de recherche

M. BOIS Guy, Ph.D, membre

À l'aube d'un jour nouveau...

REMERCIEMENTS

Ce mémoire et tout le travail y étant associé n'aurait pas été possible sans l'apport et le soutien d'un grand nombre d'individus, je tiens à les remercier. (Sans ordre précis, vous ne pouviez pas tous être en haut... ;))

Les incontournables

Sylvain Martel : Directeur de "mon" Lab. Visionnaire et stratège sans précédent, le projet NanoWalker ne serait pas sans sa profonde motivation.

François Guibault et Guy Bois : Respectivement président et membre du jury d'examen du présent ouvrage.

Merci à mes parents, sans qui je n'aurais jamais pu me rendre si loin et être ici aujourd'hui.

NATEQ et CRSNG : pour le financement bien apprécié.

Ceux du Lab

«Dr.» Thomas Boitani : Mon Italien ! Je lui ai fait découvrir la cabane à sucre, descendre une montagne en *crazy carpet*...la nuit ! Il m'a donné espoir quand je n'en avais plus... A lasting friendship that knows no bounds. Far too many things to thank you for in a single thesis...Grazie mille... Excusez-le, il est italien ! ;-)

Jean-Baptiste Mathieu : mon «F» préféré, jeune Padawan Photoshop ! Complice depuis le tout début. Il y aurait trop de choses à dire ! T'as raison JBunny, ça

-aurait- pu être pire ! Allez, on se revoit à Tahiti... ;) Merci JB !

Maurice Jacques-André Kakou Delafosse : entre SysAdmin, on se comprend ! Présent depuis le jour 1, toujours là pour un coup de main et éteindre les feux... Certainement un des plus grands hommes que je connaisse, sans jeu de mots ! :) Merci Kakou !

Marc-Antoine Fortin : Le gars d'électronique tant attendu ! Bon, ok, il vient de Sherbrooke ! ;) Ensemble, on en a vu des vertes et des pas mures ! L'optimisme légendaire de MAF a su tenir tête à mon «réaliste» tout aussi légendaire ! Oui, je sais que c'était pas ton idée à St-Tite ! :) Merci MAF !

Marc-Antoine Ducas : Un des parcours académique les plus étranges que j'ai pu voir, mais il est partout à sa place ! Un collègue, un associé, un ami ! Si jamais je lui arrive à la cheville, je serai fier ! Merci Marco !

Stefan Riebel & Andreas Schindler : mes premiers Allemands ! Avec qui toute cette histoire a débuté... Sans eux, on ne serait sûrement pas au même endroit aujourd'hui. Danke Schöen ! Qui aurait cru que des Allemands pouvaient devenir accrocs d'UT ? :)

Marc Rawji : Présent au tout début, alors qu'on se demandait encore si c'était un rêve ou juste une mauvaise joke ! ;) Pionnier de la grille atomique, maître Jedi à ses heures ! Faut encore qu'on aille grimper ensemble !

Eric L'Heureux : Mon freak personnel, rarement aies-je vu quelqu'un d'aussi compétent. J'ai tellement appris de lui en un simple été !

Guido Baumann : C'est peut-être Stefan et Andy qui ont introduit la tradition des beignes au Lab, mais c'est Guido qui l'a perfectionnée ! Toujours calme, toujours poli, toujours de qualité, ce fut un honneur...

Guillaume Langlois & Martin Mankiewicz : Deux autres des braves présents depuis le début. Parmis les personnalités les plus fortes du Lab. Franc tireur, tant à UT que dans la vie ! Un peu *rogues* sur les bords, partout en même temps, mais s'investissent toujours à fond...

Oliver-Don Truong : Oltru ! Le dernier venu du Lab, directement de l'UQAM ! Oui oui, il m'a battu au Babyfoot ! Mais j'ai pas dit mon dernier mot...

Charles C. Tremblay : Est-ce que c'est ça qu'on appelle *Alternative lifestyle* ? ;-) Avec lui, on ne s'ennuie jamais, il a su égayer nos journées les plus sombres avec ses histoires tordues ! Merci Charles.

Arnaud Chanu : SysAdmin, nouvelle génération ! Eh oui, je sais, j'avais raison... J'espère que je te laisse plus de solutions que de problèmes.

Ouajdi Felfoul : Il a su tenir tête à l'infâme "DansTaFace", on a créé un monstre en lui montrant à jouer à UT ! ;)

Walder André : Ou André Walder ? Lui-même se trompe ! Guitariste hors pair et un des plus acharné travailleur que la Terre ne connaisse, ça ne fait aucun doute ! Il n'y a pas que la CMC qui soit impressionné par ton travail Walder... moi aussi.

Simon McDougall, Pascal Hannoyer, Serge-Olivier Chimi-Ngakeng, Pierre-Alain Dumas, Na-Mi Bae, Eric "Abou" Aboussouan, Micheline Lafrenière, Albert Nsamirizi, Constantin Fortier, Frédérick Jubinville, Sylvain Boissé, Frédéric Nguyenphat-Therrien, José Pascual, Siaka Baro, Moufid Eyitayo, Wael Sabra, Marc Léger, David Salamanca, Neila Kaou, Kwang Soo Kim et tous les autres que j'oublie.

L'aide extérieure

Suzie Poulin (Poly), Hélène Bourque (Poly), Patricia Moraille (LCM UdeM), Eric Duchesne (CM^2 Poly), Peter Grütter (McGill), Geoffrey Bastien, Jeff Bengston (LPKF) pour les nombreuses heures passées au téléphone à réparer cette *scheiss-machine*, Termium Plus pour m'avoir enseigné le Français technique, Zophia Melançon pour nous avoir sauvé la vie plus d'une fois avec les boards électroniques, Julien Lardy pour avoir donné vie à notre imagination.

Les Impoly

La gang du lundi : amis depuis toujours et pour toujours ! Votre aide est bien plus grande que vous ne le savez...

Merci, encore merci...

RÉSUMÉ

Dans le cadre du projet NanoWalker du Laboratoire de NanoRobotique de l'École Polytechnique de Montréal, l'infrastructure informatique de la plate-forme de support est développée. Son objectif est de mettre en place les mécanismes qui permettront au robot NanoWalker d'atteindre un positionnement atomique répétable.

Trois principaux aspects sont abordés. Une architecture distribuée en six niveaux contrôle et coordonne une flotte de robots ainsi que leur environnement. Elle s'enracine dans les patrons de conceptions et les principes orientés-objets pour accroître sa modularité, sa robustesse et sa maintenabilité.

Le contrôle du microscope à effet tunnel (STM) monté à même le robot est implémenté sur une carte de développement pour DSP. L'intégration de la communication infrarouge est également tentée. Des tests révèlent cependant un problème lorsque la transmission infrarouge fonctionne simultanément avec le balayage du tube piézoélectrique du STM : le DSP semble incapable de garder le rythme.

L'élaboration d'une «grille atomique» basée sur la microscopie à effet tunnel est finalement étudiée. Ce positionnement de précision se divise en deux phases : la *Coarse Positioning Grid* et l'algorithme IAPA convergent de façon itérative vers le centre où une *Fine Positioning Grid* dépose les bases du positionnement fin du microscope.

ABSTRACT

This Master's thesis is inscribed within the framework of the NanoWalker project of the NanoRobotics Laboratory of the École Polytechnique de Montréal (EPM). The software infrastructure of the supporting platform is developed. Its goal is to put in place the software mechanisms that will enable the NanoWalker robot to repeatedly position itself above a single atom.

Three main aspects are approached. A six-levels distributed architecture controls and coordinates a fleet of robots and their environment. It find its basis in the design patterns and object-oriented principles to increase its modularity, stability and maintainability.

The control of the robot's built-in scanning tunneling microscope (STM) is implemented on a DSP development board. The infrared communication's integration is also attempted. Although, tests revealed a problem when IR transmission is simultaneous to the scanning of the piezoelectric tube of the STM : the NanoWalker's DSP seem unable to keep up the rhythm.

The elaboration of an "atomic grid" based on scanning tunneling microscopy is finally studied. This precise positioning is divised into two phases : the Coarse Positioning Grid and the IAPA algorithm iteratively converge toward the center where a Fine Positioning Grid lays down the basis for the microscope fine positioning.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ANNEXES

LISTE DES NOTATIONS ET DES SYMBOLES

ADC	Convertisseur analogique-numérique (*Analog-to-Digital converter*)
AFM	Microscope à force atomique (*Atomic Force Microscope*)
API	Interface de programmation d'application (*Application Programming Interface*)
CPLD	*Complex Programmable Logic Device*
CRC	*Cyclic Redundancy Check*
DAC	Convertisseur numérique-analogique (*Digital-to-Analog converter*)
DC	Courant continu (*Direct current*)
DSP	Processeur de signal numérique (*Digital Signal Processor*)
FIB	Faisceau d'ions focalisé (*Focused Ion Beam*)
FIFO	File (*First In, First Out*)
FIR	*Fast-Speed Infrared*
GPIB	*General Purpose Interface Bus* (`IEEE-488`)
GPIO	*General Purpose Input/Output*
HOPG	*Highly Oriented Pyrolytic Graphite*
IR	Infrarouge
IP	*Internet Protocol*
IrDA	*Infrared Data Association*
IRQ	*Interruption Request*
ISA	*Industry Standard Architecture*
ISR	Sous-routine d'interruption (*interrupt service routine*)
LED	*Light emitting diode* (DEL en français)
LSB	Bit de poids faible (*Least significant bit*)
MIT	Massachusetts Institute of Technology
MSB	Bit de poids fort (*Most significant bit*)
NI	Compagnie National Instruments

NW	NanoWalker
OO	Orienté objet (*Object-oriented*)
OS	Système d'exploitation (*Operating System*)
PID	Contrôleur proportionnel, dérivé et intégral
PMMA	Poly(méthacrylate de méthyle)
PLC	*Programmable Logic Controller*
PSD	*Position Sensing Device*
RAM	Mémoire vive (*Random Access Memory*)
RPM	Rotations par minute
SCI	Interface de communication sérielle (*Serial Communication Interface*)
SEM	Microscope à balayage électronique (*Scanning Electron Microscope*)
SPM	Microscope-sonde à balayage (*Scanning Probe Microscopy*)
STM	Microscope à effet tunnel (*Scanning Tunneling Microscope*)
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*

INTRODUCTION

Les nanotechnologies ont comme objectif l'étude et la manipulation de l'infiniment petit. Molécule par molécule ou atome par atome, elles visent à agencer la matière afin de bâtir des structures aux propriétés jusqu'ici inatteignables. Des matériaux de construction 50 fois plus résistants que l'acier et des puces électroniques plus petites et plus rapides ne sont qu'un aperçu des prouesses rendues possibles avec l'émergence des nanotechnologies.[46]

Cependant, cette science demeure bien jeune, elle est encore principalement confinée aux centres de recherches universitaires et des grandes entreprises œuvrant en haute-technologie. Ces centres se font compétition afin d'être les premiers à conquérir ce nouveau domaine où chaque découverte et chaque réalisation repousse les limites du savoir.

La nanorobotique n'a pas pour objectif de créer des robots aux dimensions nanométriques, la science moderne n'en est pas encore capable. Elle cherche plutôt à appliquer les principes de la robotique au contrôle des atomes et molécules. Autrement dit, la nanorobotique est l'application de la robotique aux nanotechnologies, et non l'application des nanotechnologies à la robotique.

Le projet NanoWalker a initialement vu le jour au *BioInstrumentation Laboratory* du Massachusetts Institute of Technology (MIT) sous la direction des professeurs Ian Hunter et Sylvain Martel. Le NanoRunner, présenté à la figure 1, est une de leur réalisation ayant mené au NanoWalker. Le 6 mai 2002, le Laboratoire de NanoRobotique de l'École Polytechnique de Montréal, sous la direction de Sylvain Martel, accueille ses premiers étudiants aux cycles supérieurs. Sans chaise et sans ordinateur, cinq ou six étudiants gradués et une poignée de stagiaires s'affairent au

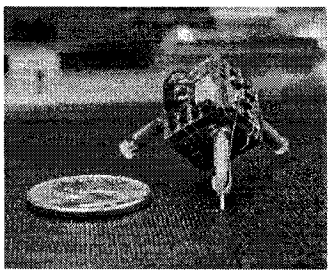


Figure 1 NanoRunner : l'ancêtre du NanoWalker

cours de l'été à bâtir le laboratoire qui doit maintenant poursuivre le projet et en entamer des nouveaux. (*i.e.* MR-Sub[40], WalkingDie[5])

Toujours en collaboration avec le MIT, le projet NanoWalker vise la mise sur pieds d'une flotte de robots miniatures, autonomes et sans fil capables d'effectuer des opérations aux niveaux atomique et moléculaire. Il s'agit d'une approche brisant avec la tradition établie voulant que l'échantillon manipulé soit promené d'appareils en appareils à l'instar d'une chaîne de montage automobile jusqu'à ce que toutes les opérations désirées soient complétées. Le déplacement de l'échantillon, souvent manuel, devient rapidement fastidieux et prompt à l'erreur lorsqu'une longue séquence d'opérations, requérant plusieurs appareils différents, est nécessaire. Ici, chaque robot constitue un instrument scientifique mobile qui peut se positionner au-dessus de l'échantillon et y pratiquer des opérations. Avec une flotte d'une centaine de robots, le traitement parallèle de plusieurs échantillons, et ce, sans intervention humaine, est envisageable. Dans une telle *nano-usine*, le débit (*throughput*) se voit considérablement augmenté. La vision à long terme du professeur Martel veut qu'à même la flotte de robots, on retrouve plusieurs types d'instruments scientifiques différents. Les robots se relayeraient, les uns après les autres, sur un même échantillon jusqu'à ce que l'ensemble des opérations soient complétées.

Figure 2 Modèle 3D du robot NanoWalker

L'existence d'une telle flotte de robots pourrait révolutionner plusieurs domaines scientifiques actuels tels que la nanofabrication, la médecine, l'étude des surfaces, etc. La plate-forme NanoWalker se veut d'abord et avant tout un outil à la disposition de la communauté scientifique, ses applications viendront avec l'usage. Mais, son développement reste encore à achever.

Chaque robot, nommé NanoWalker[43], est en fait un complexe circuit électronique replié autour d'un squelette fait de *Super Invar*[17], un métal doté d'un coefficient de dilatation thermique très faible ($0.630 \times 10^{-6} \frac{cm/cm}{^{\circ}C}$). Le cube ainsi formé mesure approximativement $30mm$ d'arrête. (voir Figure 2) La version actuelle du NanoWalker est équipée d'un microscope à effet tunnel ou *scanning tunneling microscope* (STM). Cette technologie, inventée par Binnig et Rohrer en 1981 au centre de recherche IBM à Zurich[28], permet non seulement l'imagerie d'une surface avec une résolution atomique, mais également la manipulation atomique. Un exemple célèbre fut publié en 1990 par Eigler et Schweizer[21] dans la revue scientifique *Nature*. On y voit le logo de la compagnie où ils travaillent assemblé un atome à la fois sur un substrat de Nickel. (voir Figure 3)

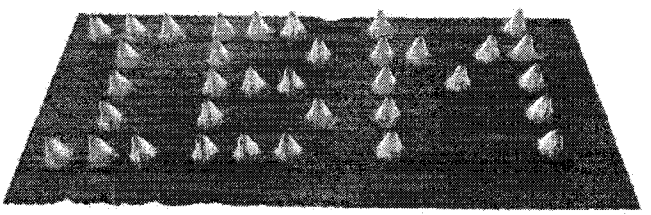


Figure 3 Logo d'IBM formé d'atomes de Xenon assemblés sur du Nickel (110)

La mobilité du circuit flexible, baptisé *Flex*, est assurée par trois pattes piézoélectriques sur lesquels le squelette est monté. En faisant littéralement vibrer ses pattes, le NanoWalker est capable de se mouvoir avec une précision de pas sous le micromètre. Le NanoWalker évolue sur un plancher spécial appelé *Powerfloor* d'où il tire son alimentation électrique ; les batteries conventionnelles capables de fournir les 15 à $25W$ nécessaires au robot étant trop grosses pour les lui fixer. Le champ d'action des NanoWalkers est donc limité à ce plancher de $0,8m \times 0,8m$ sur lequel sont intégrés plusieurs porte-échantillons circulaires, typiquement au nombre d'un par robot ou plus. Ceux-ci servent à recevoir les échantillons sur lesquels les robots travaillent. Un prototype du *Powerfloor* est présenté à la figure 4.

Cent robots dissipant de 15 à $25W$ chacun nécessite un environnement contrôlé. Le rayonnement seul n'est pas assez efficace pour maintenir la température d'opération des robots entre 0 et $+70^{\circ}$C. En dehors de cet intervalle, les composants électroniques du robot sont sujets aux défaillances. Le STM monté sur les robot étant très sensible aux vibrations, il est impensable de les refroidir en leur fixant un ventilateur comme on le fait pour les microprocesseurs d'ordinateurs. L'approche préconisée lors de l'été 2002 est donc de refroidir l'environnement même des robots. Une chambre de refroidissement[53] refroidit une atmosphère d'hélium à l'aide d'azote liquide afin d'assurer la survie des robots. Cette chambre, fabriquée

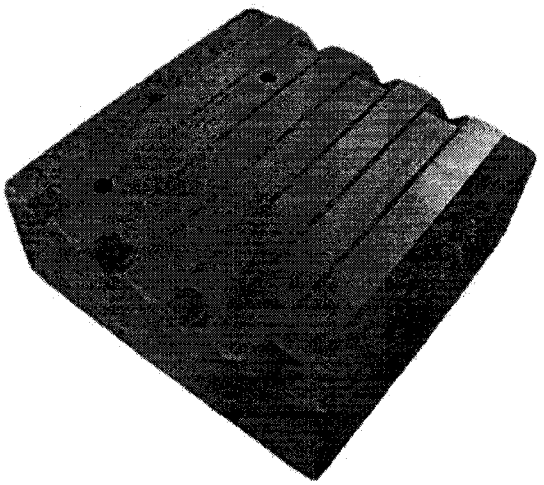


Figure 4 Prototype du *Powerfloor* avec ses porte-échantillons

par la compagnie montréalaise Cryotronix, permet d'atteindre des températures avoisinants les $-100^{\circ}C$.

Bien qu'ils soient sans fil, les difficultés de miniaturisation limitent les fonctionnalités pouvant être prises en charge par les robots mêmes. Conséquemment, le contrôle et la coordination de la flotte sont relégués à un système externe communiquant avec les robots grâce à un lien infrarouge bidirectionnel à $4Mbps$.

L'objectif du présent mémoire est la conception de l'infrastructure informatique du système de contrôle permettant le positionnement atomique des robots. Bien que le premier robot NanoWalker ne soit pas encore opérationnel et que ses spécifications continuent d'évoluer, le développement de la plate-forme doit progresser en parallèle.

L'héritage reçu du MIT en mai 2002 ne présente aucune ébauche de système logiciel. On n'y retrouve que quelques tests fonctionnels de certains composants du robot sans pouvoir parler d'intégration ou de vue d'ensemble. C'est précisément ce que

cette maîtrise cherche à corriger.

Le premier jalon à atteindre pour le projet NanoWalker est de se positionner de façon répétable au-dessus d'un atome unique. C'est l'objectif qui motive chaque décision et qui indique la direction générale du projet. Pour y arriver, plusieurs aspects sont à aborder et la décomposition du présent travail est comme suit. Le chapitre premier présente le projet NanoWalker dans son ensemble, les différents systèmes sont expliqués afin d'aider le lecteur novice au projet à bien saisir le contexte informatique et les enjeux. Le chapitre second élabore l'infrastructure informatique de la plate-forme de support. Une architecture client-serveur, qui n'est pas sans rappeler l'architecture CORBA[16], est développée. Le présent mémoire élabore les fondements de l'architecture en se basant sur une étude des caractéristiques des robots et des tâches qu'ils auront à effectuer.

En troisième partie, le développement du microscope à effet tunnel et de son contrôle est traité. Une étude des STM commerciaux permet de développer les algorithmes à embarquer sur le robot pour procéder à un scan adéquat. Puisque la transmission des données repose sur la fiabilité du lien infrarouge, celui-ci est également abordé.

Finalement, le projet de la «grille atomique» ainsi que les articles qu'il a engendrés est présenté. L'analyse des méthodes actuelles de positionnement de précision, comme l'interférométrie au laser, nous porte à développer une nouvelle approche. Ce chapitre dépose les bases de ce nouveau système de positionnement en vu d'atteindre un positionnement atomique répétable.

La méthodologie est similaire pour l'ensemble des projets et découle directement du principal handicap affligeant le projet NanoWalker à son arrivée à Montréal : de par la nouveauté du projet et du Laboratoire, personne ne connaissait réellement

l'étendue des choses à faire, ni la technologie pour y arriver. Lorsqu'on ne connaît pas une technologie, il est quasi-impossible de prévoir comment l'utiliser sciemment pour solutionner les problèmes rencontrés. Il est donc nécessaire de prototyper avec deux objectifs en tête : premièrement, maîtriser la technologie disponible, mais plus important encore, comprendre le problème auquel on est confronté.[20, 60, 51, 3]

Ces prototypes ne sont pas sensés faire parties du système final. Ils n'agissent qu'à titre de guides, une étape de design suit donc celle de prototypage. Le développement proprement dit (le code lui-même) vient à la toute fin. Cependant, comme dans tout projet évolutif, les requis ne sont pas immuables et le processus de développement logiciel est itératif plutôt que linéaire pour permettre une révision constante des modules afin d'incorporer les derniers requis et de modifier l'architecture pour refléter la meilleure compréhension du système global.

La programmation devrait normalement être effectuée par des étudiants stagiaires ou en projet de fin d'études (PFE). Mais, lorsqu'on parle d'ajout de personnel, on doit également parler de supervision. Ainsi, une partie non négligeable du temps de cette maîtrise fut passée à assurer la supervision des divers étudiants amenés à mettre la main à la pâte pour faire avancer le projet. (voir annexe I)

Plusieurs projets de construction de STM «maisons» existent sur Internet, le plus sérieux étant probablement celui de Jürgen Müller débuté en 1999[33]. Aucun de ces projets personnels ne tente cependant d'intégrer un STM à une unité mobile et sans fil comme le fait le NanoWalker.

L'institut de microtechnique de l'Université de Neuchatel en Suisse, en collaboration avec la NASA, gère un projet qui s'inscrit dans la même lignée que le NanoWalker. Son but est l'implémentation d'un microscope à force atomique (AFM) sur le robot Phoenix Lander prévu pour être envoyé sur Mars en 2007[26, 27].

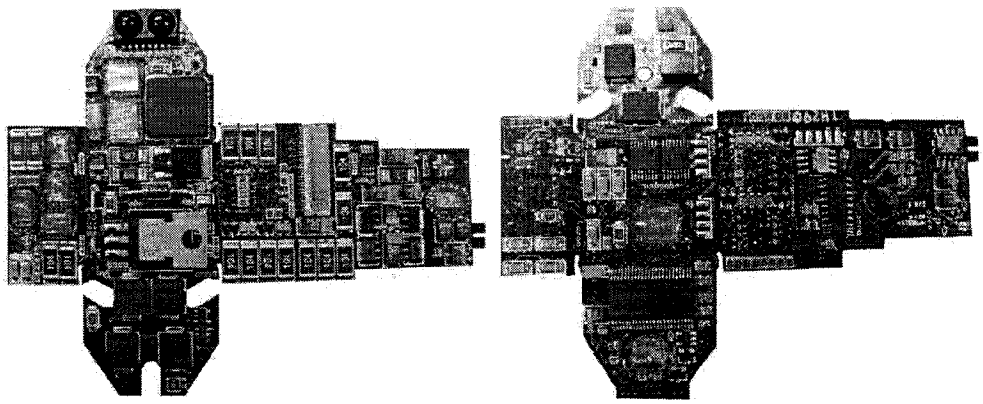
CHAPITRE 1

SYSTÈMES DU PROJET NANOWALKER

Le projet NanoWalker fait intervenir un ensemble de systèmes, tant dans le domaine de l'informatique que de l'électronique, de la physique et de la mécanique. L'interaction entre eux doit être précise et efficace. Le présent chapitre a pour but de faire un survol des différents composants du projet NanoWalker et d'expliquer les liens qu'ils ont avec l'informatique.

1.1 Le robot NanoWalker

Le circuit *Flex* de la figure 1.1 possède un total de 12 couches et plus de 300 composants. Dire qu'il est très dense est une litote. À son centre se trouve un DSP de la compagnie Texas Instruments qui lui sert de cerveau. Ce DSP, un TMS320C25-50, roule à une fréquence de $48MHz$. Sa responsabilité est le contrôle de tous les systèmes électroniques du circuit. Le choix d'un DSP 16-bits plutôt qu'un de 32 bits vient du MIT. Celui-ci s'explique par les difficultés de routage supplémentaires que présenterait un bus de données 32-bits et par le fait qu'une version en dé (*die*) est disponible pour le TMS320C25-50 ; c'est-à-dire que seul le circuit électronique, sans son boîtier, est apposé sur le *Flex*. Des microfils assurent la connexion des plots vers les traces du circuit imprimé. Cette approche permet une économie d'espace si importante pour la miniaturisation. Un CPLD de la compagnie Xilinx (XC95144XL) assiste le DSP et agit à titre de moelle épinière en lui procurant des raccourcis vers diverses fonctions telles que l'adressage des composantes, la communication sérielle I^2C avec les capteurs de température, etc.



(a) Vue du dessus (b) Vue du dessous

Figure 1.1 Circuit Flex du NanoWalker

La présente version du robot NanoWalker est équipée d'un microscope à effet tunnel (STM) dont les détails de fonctionnement sont donnés au chapitre 3. Le contrôle de ce microscope passe par l'interface avec un convertisseur analogique-numérique et un convertisseur numérique-analogique. (un ADC976A de la compagnie Analog Devices et un DAC7644 de la compagnie Burr-Brown respectivement)

Trois piézos font office de pattes et permettent au robot de se mouvoir. Une boule d'acier est fixée au bout de chaque patte et crée le contact électrique avec le *Powerfloor*. Les bandes du *Powerfloor* (voir Figure 4 de l'introduction) sont alternativement alimentées à une tension de 0V et 12V pour fournir la puissance électrique nécessaire au robot. Une bande non conductrice est insérée entre chacune d'elle afin d'éviter tout court-circuit. Le design mécanique du *Powerfloor* est tel qu'il est impossible pour une même patte de toucher simultanément deux bandes de potentiels différents. De plus, le triangle que forme les pattes garantit au robot d'avoir en tout temps au moins une patte sur l'alimentation positive et une patte sur le neutre. Un circuit de redressement permet d'inverser les potentiels appliqués aux pattes sans causer de problème.

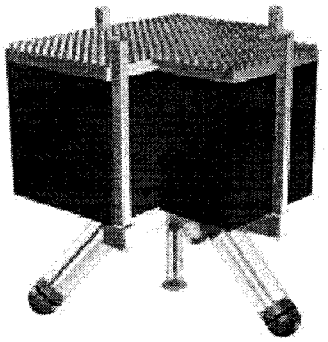


Figure 1.2 Position des convertisseurs DC-DC à l'intérieur du NW

En comptant le piézo du STM, un total de quatre tubes piézoélectriques sont installés sur le NanoWalker. Leur déformation mécanique est due à l'application de tensions électriques élevées sur leur électrodes. Pour atteindre ces niveaux de tensions, trois convertisseurs DC-DC sont placés au centre du squelette, à l'intérieur même du cube formé par le circuit *Flex* replié. (voir Figure 1.2) Une plaque de refroidissement (*heat-sink*) en cuivre est déposée sur les trois convertisseurs et un «manteau» de cuivre vient recouvrir la totalité du circuit pour tenter de garder la température du robot entre 0 et $+70^{\circ}C$. Trois senseurs de température (des LM75 de National Semiconductors) sont fixés au circuit pour surveiller l'évolution de la température : un sur la face intérieure près des DC-DC, les deux autres sur la face extérieure, dont un sous le régulateur 5V.

Un contrôleur infrarouge TIR2000 de Texas Instruments, couplé à un émetteur-récepteur combiné (*transceiver*) HSDL-3600 de Agilent (en remplacement du HSDL-1100) reçoit à $4Mbits$ semi-duplex les ordres du système de contrôle et transmet les réponses du NanoWalker. Le format physique des messages suit le protocole *Fast-InfraRed* (FIR) de la norme IrDA 1.1[64].

Le robot ne connaît pas sa propre position dans le référentiel du *Powerfloor*. Son

positionnement repose sur deux LEDs infrarouges situées sur le dessus du cube. Ces LEDs, des LNA2W01L (LN57) de Panasonic en remplacement du modèle HSDL-4420 de Agilent, émettent une lumière à une longueur d'onde de $950nm$. Celle-ci est recueillie par un capteur de la chambre. En faisant clignoter alternativement les LEDs à une certaine fréquence, la position du centre du robot et son orientation peuvent être déterminées.

La programmation du DSP est faite en langage C puisqu'un compilateur est disponible directement de la compagnie TI. Quelques tests fonctionnels ont été réalisés par le MIT, mais la plupart du code est à refaire car celui-ci n'est pas conçu avec une optique d'intégration des modules du robot. Le CPLD, quant à lui, est programmé en VHDL. Il assiste le DSP en contrôlant l'accès au divers périphériques, dont les deux mémoires présentes sur le circuit : une mémoire *Flash* non volatile de $2Mbits$ qui contient le programme du NanoWalker et une SRAM volatile de 1 Mbit ($64k \times 16bits$) où sont stockées les données de communication IR et celles provenant du STM en plus des données temporaires nécessaires à l'exécution du programme. Le DSP complète la mémoire totale disponible avec une RAM interne de 544 mots de 16 bits. Plusieurs intervenants, tant au MIT qu'à Polytechnique ont travaillé sur le code du CPLD. Nommons notamment Albert Nsamirizi, Felix Poulin, Marc-André Prud'Homme et Philippe Ouimet[48], à qui l'on doit la majeure partie du travail d'intégration du code du CPLD.

1.2 Environnement

L'environnement du NanoWalker consiste en une chambre de refroidissement (voir Figure 1.3) ayant pour principal objectif de refroidir son atmosphère d'hélium afin d'augmenter l'évacuation de la chaleur des robots. L'écoulement du gaz y est laminaire de gauche à droite et sa vitesse contrôlée par un ventilateur tournant à

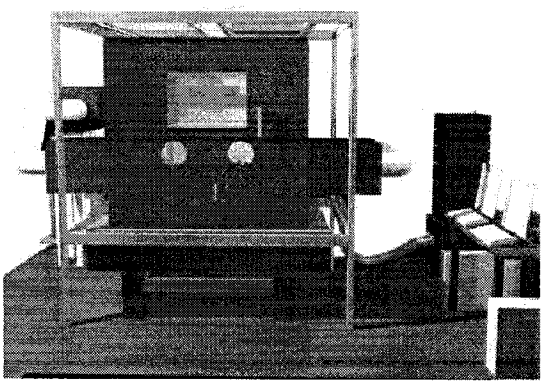


Figure 1.3 Modèle de la chambre de refroidissement

une vitesse de 0 à 1200RPM. Un PLC (modèle T100MD1616+ de la compagnie Triangle Research International) implémente un PID régissant l'ouverture de la valve d'azote servant à refroidir l'hélium de la chambre. Deux thermocouples situés à l'entrée et à la sortie d'air fournissent les mesures utilisées par le contrôleur pour calculer la rétroaction à appliquer. Des tests réalisés par Haritz Macicior[38], étudiant au laboratoire, ont cependant démontrés que ce PID est plutôt mal conçu et qu'il y aurait avantage à le transférer à l'extérieur du PLC dans le système de contrôle. De cette manière, plusieurs autres thermocouples placés ça et là dans la chambre pourraient être utilisés par le contrôleur. Le PLC est donc relégué à un rôle passif d'exécution des ordres qui lui sont communiqués à travers un lien série RS232 par le système de contrôle.

Du matériel d'acquisition de données fabriqué par la compagnie National Instruments est apposé dans la chambre. Ainsi, une carte PCI-6036E communique avec un chassis SCXI-1000 pour recueillir les données des modules qu'il contient. La carte SCXI-1112 possède huit canaux auxquels sont branchés des thermocouples (type T) distribués dans la chambre. (leur emplacement final n'est pas encore déterminé) La carte SCXI-1530, quant à elle, permet l'acquisition de données de vibrations pro-

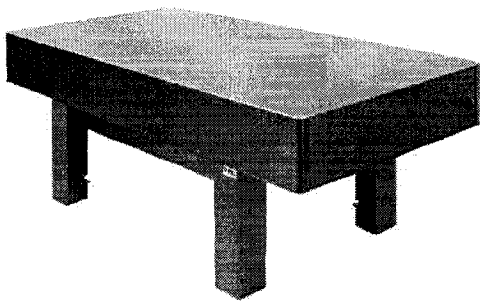


Figure 1.4 Table optique sur laquelle repose la chambre de refroidissement

Figure 1.5 *Position Sensing Device* PSM-10 de la compagnie On-Trak

venant de deux accéléromètres. Ces données sont importantes car de trop grandes vibrations sont extrêmement nuisibles au fonctionnement du STM. Visant à réduire au maximum les perturbations mécaniques, la chambre de refroidissement est déposée sur une table optique de la compagnie TMC. (voir figure 1.4) Cette table, d'une masse approximative de $513kg$, «flotte» sur un coussin d'air et amortit les vibrations mécaniques de hautes fréquences ($> 100Hz$)[62].

Comme deuxième dispositif anti-vibrations, le *Powerfloor* lui-même est déposé sur un système de ressorts. Ancré à ce support se trouve une charpente de métal à laquelle est fixée un *Position Sensing Device* (PSD) (voir Figure 1.5) chargé de capter la lumière infrarouge émise par les LEDs de positionnement des NanoWalkers ainsi qu'un transmetteur infrarouge pour la communication. Il est primordial que cette structure soit aussi rigide que possible afin d'atténuer au maximum les facteurs d'erreurs sur l'acquisition de la position.

Sous l'effet d'un faisceau lumineux, la photodiode du PSD complète un circuit de sorte que, une fois amplifiées, deux tensions fonctions du point d'incidence en XY sur la matrice du PSD sont émises en sortie. Cette matrice de $10mm \times 10mm$ n'est pas assez grande pour couvrir toute la surface du *Powerfloor*, une lentille Componon-S 2,8/50 (en remplacement d'une Apo Componon 4/45) de Schnei-

der Optics y est fixée afin d'augmenter son champ de vision. Un filtre optique HQ950/60m de la compagnie Zeiss complète le montage. Quatre PSD sont nécessaires pour couvrir la totalité de la superficie. Guido Baumann[8], Guillaume Langlois[35] et Eric Aboussouan[1] se sont affairés à obtenir la meilleure précision possible avec un système composé d'un seul PSD. En avril 2004, Eric Aboussouan indique qu'une précision de $\pm 75\mu m$ est atteignable avec une bonne répétabilité des mesures.

L'intérieur de la chambre est isolé du monde extérieur, quatre orifices (*cable ports*) sont prévus pour passer les câbles d'alimentation et de données : deux sur le dessus et deux à l'arrière. Pour permettre une communication IR efficace, la longueur des câbles doit être limitée à, au plus, un mètre. Un ordinateur de contrôle doit donc être installé sur le dessus de la chambre, près du PSD et du transmetteur IR.

Le bloc d'alimentation électrique capable de fournir les quelques deux cents ampères nécessaires au fonctionnement continue d'une flotte de cent NanoWalkers est branché à un ordinateur par une interface GPIB. Ceci permet de surveiller la puissance totale des robots et d'implémenter un mécanisme d'arrêt d'urgence automatique si jamais un problème survenait.

L'architecture logicielle à bâtir doit tenir compte de tous ces systèmes hétérogènes dans sa conception.

CHAPITRE 2

PLATE-FORME LOGICIELLE - ARCHITECTURE

L'informatique dans le projet NanoWalker peut être segmenté en deux aspects distincts : le logiciel embarqué sur le DSP du robot et le logiciel de contrôle de la plate-forme de support. Ce dernier interagit de façon directe et indirecte avec les NanoWalkers. Chaque système de la plate-forme fait intervenir des composantes matérielles ainsi que des composantes logicielles qui régissent leur utilisation. La figure 2.1 illustre d'une façon schématique les trois mécanismes d'interactions actuels avec les robots, soient le système de communication infrarouge, le système de positionnement global (PSD) et les sous-systèmes de la chambre de refroidissement. Pris individuellement, chaque mécanisme possède une raison d'être qui lui est propre et ne dépend en aucune façon des autres composants du système global. Chacun possède un pilote (*driver*) qui contrôle l'accès au matériel selon une approche de type «boîte-noire», c'est-à-dire que les détails internes de fonctionnement sont dissimulés à l'utilisateur. L'intégration, la synchronisation et l'utilisation des différents pilotes pour contrôler sciemment les robots sont régies par un système supérieur et spécifique au projet NanoWalker.

Ce système logiciel de haut niveau se voit imposer les diverses responsabilités pré-

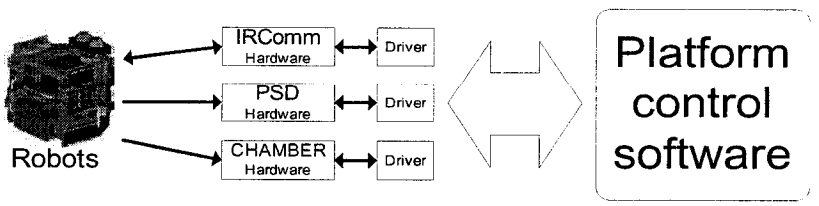


Figure 2.1 Représentation schématique du système logiciel

Tableau 2.1 Liste des responsabilités du système logiciel de la plate-forme du projet NanoWalker

Responsabilités
– Gestion de la communication entre les robots et la plate-forme – Surveillance de l'environnement de travail des robots (température, vibration, alimentation électrique, etc.) – Coordination des déplacements des robots (éviter les collisions, etc.) – Interprétation des communications provenant du robot (données STM, état, température, etc.) – Maintient des données de calibration spécifiques à chaque robot – Mémorisation de la position des zones de travail sur le *Powerfloor* – Fournir une interface simple pour le développement d'applications spécifiques faisant usage de la plate-forme NanoWalker – Gestion des requêtes des applications clients.

sentées dans le tableau 2.1. Sans être exhaustif, les principales sont la gestion de la communication, la coordination des déplacements et la régularisation de l'environnement. Pour accomplir chacune de ces responsabilités, le logiciel doit faire intervenir un ou plusieurs des composants de bas niveaux. L'architecture développée pour cette plate-forme doit être souple afin de permettre une adaptation rapide à tout changement de configuration ou de technologie qui pourrait survenir au cours de l'évolution du projet. L'objectif à long terme de cette architecture est de permettre le fonctionnement ininterrompu des NanoWalkers pendant de longues périodes, et ce, en nécessitant un minimum d'intervention humaine.

2.1 Évaluation des alternatives

Deux approches sont envisageables d'un point de vue informatique pour implémenter le contrôle des systèmes de la plate-forme. Les deux options étudiées sont le développement d'une architecture logicielle distribuée dans laquelle le contrôle

des systèmes est fragmenté en plusieurs entités logicielles distinctes communiquant entre elles via divers mécanismes de communication, l'autre est une architecture monolithique où tout le contrôle est effectué par une machine-maîtresse unique.

Ces deux options présentent des avantages et des désavantages qui leur sont propres. Le tableau 2.2 en fait un court résumé. Le principal problème avec l'approche monolithique est que tout le système repose sur l'intégrité d'une seule machine. Dans le cas d'un système critique comme celui de la présente plate-forme, une panne incontrôlée peut avoir des répercussions sévères, tant sur le point de vue financier que sur l'intégrité physique du système lui-même. Aucun système informatique ne peut être totalement à l'abri des pannes puisque les sources potentielles sont multiples, allant d'une erreur de conception jusqu'à un simple bris matériel. Avec une architecture monolithique où la totalité du système est basée sur un même processeur, l'impact d'une panne se fait d'autant plus ressentir car celle-ci peut engendrer l'effondrement complet du système.

De plus, puisque le système logiciel à concevoir se veut lui-même une plate-forme de développement pour des applications futures, il n'existe aucun moyen de connaître *a priori* la puissance de calcul nécessaire pour ces applications. Dans une approche monolithique, tous les programmes se partagent la même puissance de calcul finie. L'arrivée d'une nouvelle tâche ajoute forcément des délais dans l'exécution des programmes déjà présents. Une surcharge pourrait paralyser le système et compromettre la survie même des robots.

L'approche monolithique est donc éliminée au profit de l'approche distribuée où le système est composé d'un ensemble plus ou moins imposant de processus logiciels qu'on appellera «nœuds». Chaque nœud réside possiblement sur une machine distincte et échange de l'information par divers mécanismes de communications, par exemple réseaux tels que les protocoles UDP et TCP/IP. Cette approche présente

Tableau 2.2 Comparaison entre une architecture monolithique et distribuée

Architecture monolithique	Architecture distribuée
Avantages	
– Simplicité d'implantation – Rapidité des communications inter-modules.	– Charge de traitement répartie à travers plusieurs nœuds – Modularité facilitée – Fiabilité accrue en cas de panne – Expansion future facilitée
Désavantages	
– Machine-maîtresse devient un goulot d'étranglement – Si la machine-maîtresse tombe en panne, tout le système est compromis – Difficulté de mise à jour et d'expansion	– Latence accrue par les mécanismes de communication – Événements totalement asynchrones, synchronisation difficile entre les modules – Complexité de design accrue

l'avantage indéniable de permettre l'ajout de nouveaux modules logiciels simplement en ajoutant des nouveaux nœuds à l'architecture déjà en place. La puissance de calcul associée à ces nouveaux nœuds est indépendante des nœuds existants et peut être adaptée selon les besoins particuliers du module ajouté. L'envers de la médaille est bien sûr la complexité accrue du système nécessaire pour gérer les communications et interactions entre les nœuds. Le temps requis pour former et transmettre un message vers un nœud distant est également plus long que celui requis pour un simple appel de fonction dans un même espace mémoire.

2.2 Hiérarchisation des niveaux logiciels

À partir des responsabilités définies dans le tableau 2.1 et du désir de modularité du système final, six niveaux logiciels sont identifiés. Chacun d'eux écope d'un

certain nombre de responsabilités. À mesure que l'on progresse à travers les couches, le niveau d'abstraction augmente. De plus, chaque niveau connaît les détails de l'interface de programmation (API) du niveau inférieur immédiat, mais pas celui du niveau supérieur. Les six niveaux sont comme suit :

1. **Hardware :** Le niveau matériel (ou `hardware`) regroupe toutes les composantes physiques du système. Il sert à réaliser le transfert et l'acquisition des données.

   Ex. : Un capteur mesure un phénomène physique sous la forme d'une tension.

2. **Driver :** Le niveau pilote (ou `driver`) contrôle l'accès au niveau `hardware` en fonction des ordres reçus du niveau supérieur. Il ne connaît d'aucune façon le but du matériel et ne peut faire aucun traitement ou interprétation sur les valeurs lues.

   Ex. : Un `driver` est l'ensemble des fonctions permettant de lire les valeurs de tension recueillies par le capteur matériel.

3. **Manager :** Le niveau `manager` traduit en information intelligible les valeurs recueillies par les fonctions des `drivers`. Le traitement fait dans un `manager` particulier se limite à un conditionnement partiel des valeurs dans le but d'en fournir une interprétation cohérente au niveau supérieur. Il ne peut jamais faire intervenir de l'information qui relève d'un autre `manager` ou d'un niveau supérieur. Un `manager` ne connaît que le `driver` qu'il utilise et ne peut en aucune circonstance présumer du traitement qui sera fait aux niveaux supérieurs.

   Ex. : Un `manager` utilise un `driver` pour recueillir la tension du capteur puis il la transforme dans les unités correspondant au phénomène physique mesuré, par exemple, des degrés Celsius.

4. **Agent :** Le niveau `agent` sert à fournir une interface vers les différents aspects conceptuels du système physique. Autrement dit, un `agent` est responsable d'un aspect particulier du système global et fourni l'interface nécessaire pour agir sur celui-ci. Un `agent` coordonne l'activité d'un ou plusieurs `managers` afin d'exécuter le plus efficacement possible les commandes qui lui sont transférées par le niveau supérieur.

   Ex. : Un `agent` reçoit la demande d'obtenir la valeur d'un paramètre physique. L'`agent` demande au `manager` correspondant la valeur numérique du phénomène physique mesuré par le capteur associé. Il majore ensuite cette valeur en fonction de variables extérieures telles que la pression, la température, la configuration actuelle du système, etc. et retourne la valeur corrigée au requérant.

5. **NanOS :** Le niveau `NanOS` sert à contrôler l'interaction entre les différents `agents` ainsi qu'à fournir une interface simple et unique pour les applications clients désirant utiliser la plate-forme NanoWalker. On y retrouve également des modules de surveillance (*monitoring*) faisant intervenir de l'information provenant des différents agents. Son but est d'assurer l'intégrité du système et d'agir en cas de problème majeur en plus de gérer l'exécution des requêtes des applications clients.

   Ex. : Un module surveille la valeur du paramètre physique mesuré et lors que celle-ci dépasse une certaine valeur, une alarme est activée et propagée à travers l'ensemble du système.

6. **Application :** Le niveau `application` englobe tout logiciel client qui cherche à utiliser la plate-forme et les fonctionnalités qu'elle offre afin d'effectuer un travail spécifique à un domaine d'application donné. Celle-ci ne contrôle pas directement les robots mais envoie des requêtes au système qui les traitera de

façon transparente et retournera le résultat une fois celui-ci obtenu.

Ex. : Faire le logo d'IBM un atome à la fois.

Les données recueillies par les différents capteurs de la plate-forme traversent successivement les niveaux où elles sont traitées et transformées en données utilisables par les couches subséquentes. Chacun des mécanismes d'intéraction de la figure 2.1 possède au minimum `driver` et un `manager`. Un `agent` utilise un ou plusieurs `managers` pour arriver à fournir l'information nécessaire aux `applications` qui en font la demande via des requêtes envoyées au `NanOS`.

2.3 Communication inter-niveaux

Dans cette architecture distribuée, les niveaux logiciels `driver`, `manager` et `agent` sont appelés à travailler en étroite collaboration les uns avec les autres. Ces trois niveaux sont solidaires et les éléments logiciels traitant d'un même aspect physique peuvent être implantés sur un même nœud. Par contre, les niveaux `agent`, `NanOS` et `application` sont conceptuellement séparés par le fait qu'ils ont des responsabilités beaucoup plus larges et font l'intégration de plusieurs modules des niveaux inférieurs. Ils peuvent être implantés dans des processus distincts s'exécutant possiblement sur des machines séparées. Tout échange d'information entre ces niveaux doit être réalisé par envoi et réception de messages à travers un canal de communication quelconque. Le canal choisi pour les échanges entre les différents niveaux «internes», c'est-à-dire les niveaux inférieurs au niveau `application`, est le réseau *Ethernet* sous la forme d'échange de paquets TCP. Chaque message est un paquet TCP[32] envoyé sur le réseau d'une source à une destination. Le choix du protocole TCP plutôt que UDP[50] repose sur plusieurs facteurs. TCP établit une connexion permanente entre l'émetteur et le récepteur. Tant qu'elle reste active, elle s'approprie un socket de façon exclusive. Le protocole UDP, quant à lui, ne

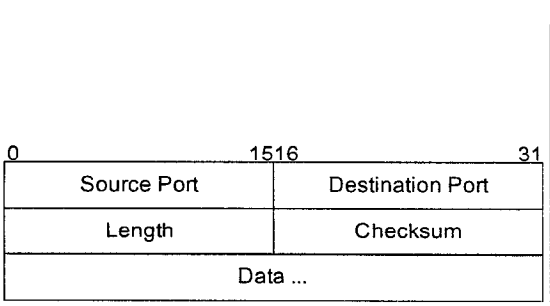


Figure 2.2 Format d'un paquet UDP

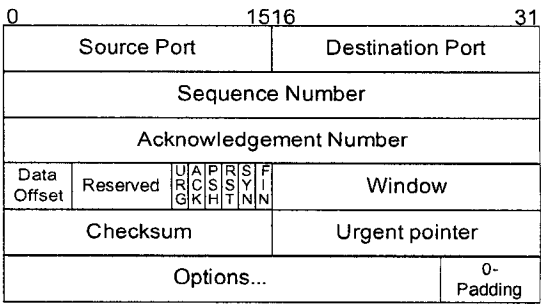


Figure 2.3 Format d'un paquet TCP

fonctionne pas en mode connecté, toutes les transmissions peuvent s'effectuer sur un même socket. Mais contrairement à TCP, il n'offre pas de garantie de service, c'est-à-dire qu'il n'existe aucun mécanisme à même le protocole pour garantir la livraison d'un paquet, ni la correspondance temporelle d'une série de paquets entre son émission et sa livraison. Un paquet A envoyé avant un paquet B peut se retrouver à la destination après celui-ci, s'il se rend. Ce type de communication n'est donc pas approprié pour l'envoi de longs messages à caractère temporel critique, mais il trouve sa force dans la faible surcharge système (*overhead*) qu'il ajoute lors de son traitement. Ceci rend UDP beaucoup plus déterministe que TCP mais, à la base, le réseau Ethernet demeure non déterministe. Certains auteurs, comme Kerkes[34], propose des modifications aux protocoles TCP/IP et UDP pour rendre Ethernet déterministe. Un survol des différents protocoles de communication temps-réel est également fait par Hassen[4].

Une autre caractéristique intéressante du protocole UDP est la possibilité de faire du *broadcasting*, c'est-à-dire qu'un même message est envoyé à tous les processus écoutant sur un socket particulier, chose impossible avec TCP. Une combinaison des deux protocoles peut alors être souhaitable dans certaines situations. Cependant, TCP demeure le choix de la première implémentation.

Les applications clients qui se connectent à l'interface de services du `NanOS` ne sont

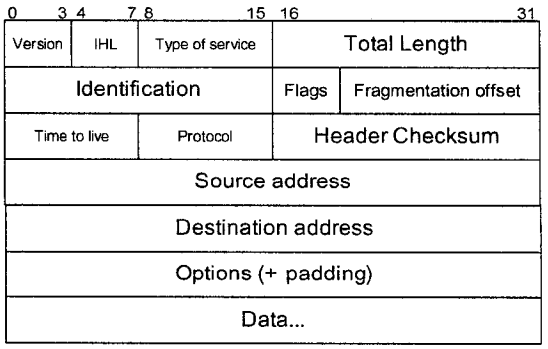


Figure 2.4 Format d'un paquet IP

pas forcément dans un lieu physique rapproché du reste de la plate-forme. Une connection à travers le réseau Internet est même envisageable, il est donc nécessaire d'assurer un lien de qualité entre les deux applications logicielles. Lorsqu'un client se connecte au système, un lien sécurisé TCP/IP est établi entre celui-ci et le serveur de la plate-forme afin de protéger et d'assurer la livraison des paquets, et ce, même si une grande distance physique les sépare. Il serait, par exemple, envisageable qu'un client à New York utilise une plate-forme NanoWalker située à Montréal à travers une application utilisant l'Internet commercial comme médium de communication.

Lorsque de l'information nouvelle doit être propagée vers les différents modules du système informatique, deux mécanismes de bases sont disponibles et supportés à différents niveaux. Il s'agit des mécanismes communément appelés *push* et *pull*[67].

Le mécanisme dit *pull* est le plus simple. Lorsqu'une application logicielle a besoin d'une information particulière, elle envoie une requête au détenteur de cette information. Ce dernier traite la requête et renvoie sa réponse contenant l'information désirée. On dit alors que l'application «tire» l'information vers elle. L'avantage de cette technique est sa simplicité, tant au point de vue de sa compréhension que des principes d'implantation. Par contre, elle présente le désavantage de nécessiter plusieurs messages à chaque fois qu'une information est demandée. En effet,

une séquence de deux messages est nécessaire pour obtenir toute information (la requête et la réponse). Lorsqu'une application a besoin d'être constamment tenue au courant de la valeur d'un paramètre, elle doit fait du *polling*, c'est-à-dire qu'elle demande périodiquement la valeur de ce paramètre. Lorsque la valeur du paramètre étudié ne varie que très rarement, une quantité impressionnante de messages peut alors être échangée inutilement, ce qui a pour effet d'engorger le réseau et de réduire l'efficacité globale des communications.

Pour éviter ceci, on peut utiliser le mécanisme dit *push* qui consiste à ce que l'information désirée par une application lui soit automatiquement transmise dès qu'un changement pertinent se produit. L'application doit d'abord signaler au détenteur de l'information son intérêt envers celle-ci. Cette étape se nomme «inscription». Par la suite, le détenteur de l'information a la responsabilité d'avertir tous ses inscrits lorsque l'information qu'il possède se voit modifiée. Cette façon de procéder permet d'éviter les messages inutiles associés au *polling*, mais a le désavantage de ne pas distinguer entre l'absence de messages due à une absence de variation de la valeur observée et celle due à une panne empêchant les messages de mise à jour d'être livrés à leurs destinataires.

Une astuce consiste alors à combiner les deux mécanismes en utilisant le *pull* seulement lorsqu'un long délai sans mise à jour survient. Ceci permet de s'assurer qu'aucune panne empêchant la livraison des mises à jour n'est survenue et que le système fonctionne toujours adéquatement.

2.4 Gestion des requêtes

Le terme «gestion des requêtes» indique la façon dont les requêtes arrivant à un serveur d'information en provenance de ses différents clients sont traitées. Avant

Tableau 2.3 Caractéristiques générales des requêtes

Caractéristiques
– Temps de traitement par requête est inconnu *a priori* – Certaines requêtes pourraient être plus prioritaires que d'autres – Nombre de requêtes simultanées inconnu mais proportionnel au nombre de clients – Plusieurs requêtes simultanées peuvent tenter d'accéder aux mêmes ressources – Un même client peut faire plus qu'une requête à la fois

d'étudier les différentes possibilités disponibles, il est utile d'établir quelques caractéristiques sur les requêtes à traiter. Le tableau 2.3 présentent quelques caractéristiques des requêtes que les applications pourront faire à la plate-forme. On remarque qu'une stratégie *multithread* est nécessaire pour accommoder plusieurs requêtes simultanées dont on ne connaît pas le temps d'exécution. Trois stratégies bien documentées dans la littérature sont envisageables pour répondre aux besoins spécifiques du système.

2.4.1 Thread par connexion

L'idée de base du mécanisme de traitement des requêtes «thread par connexion»[54] est que pour chaque client se connectant au serveur, un *thread* est créé. Toutes les requêtes provenant de ce client sont prises en charge par ce dernier. Le désavantage principal est que si jamais une requête particulière se voit bloquée pour un temps indéterminé, aucune autre requête provenant du même client ne pourra être traitée par le serveur, et ce, même si les deux requêtes ne sont pas interdépendantes. De plus, dans un tel mécanisme, il est impossible d'implanter une priorité des requêtes provenant d'un même client.

2.4.2 Thread par requêtes

Dans le mécanisme «thread par requêtes»[54], chaque requête donne lieu à la naissance d'un nouveau *thread*. L'avantage évident est que le problème affligeant le mécanisme «thread par connexion» est corrigé. Une requête bloquée ne nuira pas aux requêtes futures d'une même application, sauf si elles tentent toutes les deux d'accéder à la même ressource. Par contre, selon le nombre de requêtes simultanées arrivant au serveur, celui-ci peut se retrouver avec un nombre de *threads* concurrentiels si grand que ses ressources disponibles se voient sérieusement diminuées à un point tel qu'il n'est plus capable de répondre à la demande. Lorsque le nombre de threads concurrentiels dans un système devient trop important, le temps de changement de contexte entre chacun d'entre eux devient un facteur important dans la détérioration de la qualité du service. De plus, la création d'un thread induit un délai dans le traitement de la requête. Ce délai supplémentaire peut même devenir dominant pour des requêtes dont le temps de traitement est très court.

2.4.3 Threadpool

Le concept du *threadpool*[54] est qu'un nombre fini de *threads* est disponible pour traiter l'ensemble des requêtes provenant des clients connectés au serveur. Celles-ci sont placées dans une queue à mesure qu'elle sont livrées au serveur. Quand un *threads* termine l'exécution d'une requête, il passe à la prochaine requête d'importance dans la file, et ce, indépendamment du client l'ayant initiée. Puisque toutes les requêtes se retrouvent dans une même queue, il est facile de modifier l'ordre d'enfilage en fonction d'une valeur de priorité attribuée selon divers critères tels le type de message, le temps déjà passé dans la file d'attente, etc. Ce mécanisme possède les avantages des deux autres sans pour autant souffrir des mêmes désavantages. La difficulté se situe dans le choix du nombre de *threads* à inclure dans le

threadpool. Ce nombre peut cependant être dynamiquement modifié en fonction du nombre de clients connectés ou du volume de requêtes reçues pour mieux s'adapter à la charge de travail actuel, faisant ainsi du *threadpool* le meilleur mécanisme pour l'architecture distribuée de la plate-forme NanoWalker.

2.4.4 Patrons de conception

Pour faciliter la modularité du système, les divers éléments logiciels sont implantés selon une approche orientée-objet. On cherche ainsi à découpler les couches du système en créant des «boîtes noires» pour chaque élément et en définissant une interface de communication entre ceux-ci. Procéder de la sorte favorise et facilite la maintenance et la réutilisabilité des éléments logiciels. Les patrons de conception fournissent des solutions de design renforçant ces concepts. Ils sont tout indiqué pour le design d'un tel système complexe. Le tableau 2.4 résume l'ensemble des patrons utilisés dans le design de l'architecture du système de contrôle.

Le patron *Façade*[25] présent entre les diverses applications et le NanOS fournit une interface pour l'ensemble des applications cherchant à utiliser la plate-forme. Toutes les applications étant à prime abord équivalentes et l'ensemble des fonctionnalités de la plate-forme leur étant présentées étant le même, une interface unique va de soit. Toute l'implémentation technique de l'exécution de leurs requêtes et du contrôle des NanoWalkers leur est cachée, seul le résultat leur est transmis.

Une utilisation du patron *Remote Proxy* se retrouve également entre les applications et le NanOS. Chaque application instancie un proxy vers le NanOS. Ce proxy sert à masquer l'utilisation du réseau et les mécanismes de synchronisation nécessaires pour gérer les requêtes. De la même façon, le NanOS instancie un proxy vers chacune des applications s'y étant connectée. On retrouve également des proxies entre

Tableau 2.4 Patrons de conception utilisés dans le design de l'architecture

Patrons	**Rôle**
Façade	Procure un point d'entrée unique vers un ensemble de fonctionnalités prises en charge par des sous-systèmes. Facilite l'utilisation des sous-systèmes par un tiers.
Remote Proxy	Fournit un figurant pour un objet particulier afin d'en masquer les détails d'utilisation.
Observer	Définit une relation *1-à-n* entre divers objets permettant à un sujet d'avertir tous ses observateurs d'un changement survenu à son état.
Mediator	Définit un objet gérant l'interaction d'un ensemble d'objets entre eux. Fait la promotion d'un couplage faible entre les objets en les empêchant de se faire référence directement.
Command	Encapsule une requête en un objet pour la transmettre vers un autre objet capable de l'exécuter et effectuer des opérations sur celle-ci.
Factory Method	Définit l'interface de création d'objets tout en laissant aux classes dérivées le soin de choisir quelle classe doit être réellement instanciée.
Singleton	Assure qu'une classe n'a qu'une seule instance et fournit un point d'accès global et unique vers celle-ci.
Wrapper Facade	Encapsule les fonctions et données fournies par un API non orienté-objet à l'intérieur d'une classe plus concise, robuste et portable respectant les principes de design OO.
Acceptor-Connector	Dissocie l'étape de connexion et d'initialisation d'une application client à un serveur dans un système distribué du traitement exécuté par les deux modules une fois ceux-ci connectés.
Reactor	Permet à une application de réagir aux événements générés par l'arrivée de requêtes de services effectuées par un ou plusieurs clients.
Leader/Follower	Modèle d'architecture parallèle où plusieurs *threads* prennent tour à tour un rôle actif dans la détection et l'exécution de requêtes de service dans un système basé sur l'arrivée d'événements. Une fois le traitement terminé, le *thread* actif cède la place au suivant.
Active Object	Dissocie l'exécution d'une méthode de son appel afin d'améliorer le parallélisme et de simplifier la synchronisation d'accès à des objets se trouvant dans leur propre *thread* de control.

les niveaux `NanOS` et `Agent` de l'architecture. La localisation physique et logique des modules est ainsi cachée et un appel à une fonctionnalité éloignée s'apparente à un simple appel de fonction. Les proxies écopent également de la responsabilité d'assurer la conversion des messages transmis entre les différents participants du système en un langage commun si jamais il s'avérait qu'ils ne parlaient pas le même langage. Cette étape est communément appelée *marshalling*.

L'état des NanoWalkers physiques est amené à changer de façon totalement asynchrone et plusieurs modules sont intéressés à connaître leur état ou une partie de leur état. Plutôt que désigner le *polling* comme manière d'obtenir l'information désirée, on fait appel au patron *Observer* où chaque NanoWalker logique possède une liste d'observateurs intéressés à obtenir de l'information sur leur état. Lorsque l'état de celui-ci se voit modifié, il se charge d'avertir ses observateurs de ce changement. Ces derniers sont, par la suite, libres de prendre action s'ils le jugent nécessaire. C'est l'implémentation directe du mécanisme *push* discuté plus tôt.

Chaque agent contrôle un aspect complet du système, mais ne connaît pas en profondeur les autres aspects de la plate-forme dont il n'est pas responsable. La tâche de corréler l'information provenant de divers agents et de la propager vers les autres agents qui pourraient en avoir besoin tombe sous la responsabilité d'un médiateur. C'est le NanOS qui joue ce rôle.

Les applications effectuent des requêtes à la plate-forme par l'entremise du NanOS. Celles-ci sont ensuite transformées et propagées vers les agents capables d'y répondre. Ultimement, elles peuvent être acheminées jusqu'aux NanoWalkers physiques. Tout au long de cette chaîne, la requête subit des transformations afin de s'adapter aux divers intervenants. L'utilisation d'un patron *Command* facilite cette encapsulation des requêtes et leur manipulation.

On sait d'ores et déjà que différentes requêtes sont possibles. Sans pour autant en connaître les détails d'implémentation, on peut prendre pour acquis que la façon de les traiter différera considérablement. Pis encore, l'éventail de commandes existantes est amené à évoluer à mesure que les robots s'améliorent et que de nouvelles générations différemment équipées font leur apparition. L'architecture logicielle doit donc prévoir une technique pour permettre l'ajout de nouvelles commandes sans pour autant avoir à adapter tout le fonctionnement des classes existantes. C'est le but du patron *Factory Method.* Il permet de définir une interface pour la création d'objets dont on ne connaît pas le type avant leur utilisation. Dans le cas présent, à mesure que de nouvelles commandes sont supportées, une classe implémentant leur traitement est ajoutée à l'architecture. Seul un objet créateur de commandes doit être modifié afin de créer les commandes spécifiques en fonction des demandes des instances supérieures. Chaque commande a la responsabilité de créer les objets nécessaires à son exécution.

Plusieurs entités du système sont uniques, tant dans leur rôle que dans leurs propriétés. Le patron *Singleton* s'assure que ces classes ne puissent être instanciées qu'une seule fois dans un même processus logiciel et que chaque utilisation fasse toujours appel au même objet logique. On empêche ainsi le dédoublement de classes vitales qui se feraient concurrence et rendraient ainsi le système instable.

Les fonctions de bas niveau, par exemple de synchronisation de processus et *threads*, et les pilotes de périphériques sont habituellement implémentés dans un langage structural comme le `C` ou même l'assembleur. Ils sont aussi fortement liés au système d'exploitation utilisé, au matériel et à la technologie qu'ils contrôlent. La conséquence directe est qu'ils ne sont pas portables et parfois difficiles à utiliser. Le patron *Wrapper Facade* encapsule un ensemble de fonctions et de données conceptuellement liées dans un environnement orienté-objet qui agit à titre d'interface simple et portable. Il devient ainsi simple de modifier l'implémentation sans pour

autant modifier l'interface externe.

Dans un environnement distribué, l'établissement des connexions entre les modules clients et les serveurs d'information ne doit pas être couplé à leur mode de fonctionnement. Le patron *Acceptor-Connector* sert à dissocier ces deux étapes. Le `NanOS` implémente un «accepteur» qui attend simplement que des applications clients (possédant un «connecteur») le contactent. Par la suite, le patron *Reactor* réagit à cet événement en créant un proxy de l'application client du côté serveur.

Le traitement des requêtes se fait par un ensemble de *threads* réunis dans un *threadpool.* Chaque requête reçue d'une application client est placée dans une file. Un *thread* actif la retire et l'exécute. Une fois l'exécution terminée, il laisse à un autre thread le soin de traiter la prochaine requête. Ceci revient à implanter le patron de conception *Leader/Follower* où plusieurs *threads* joue à tour de rôle le rôle de meneur (celui qui exécute l'action) et de successeur (celui qui attend son tour). Lorsque le meneur termine son travail, il cède la place au premier successeur et reprend rang à la fin de la liste.

Le client et le serveur ne s'exécutent pas dans le même espace mémoire, ni dans le même processus logiciel. L'appel d'une méthode du proxy donne lieu à une requête envoyée sur le réseau et traitée par un *thread* du *threadpool.* Cette requête devient ainsi un «objet actif» selon l'implémentation du patron *Active Object.* Le client initiateur de la requête obtient le résultat par l'entremise d'un objet *Future* retourné par le proxy lors de l'appel de la méthode. Ce *Future* est mis à jour par la requête, une fois celle-ci complétée.

Tableau 2.5 Évolution d'une requête à travers les modules logiciels

Nom	Niveau	Description
Requête	NanOS	Reçue d'une application client ou d'une application de monitoring du NanOS même, une requête correspond à une demande de travail plus ou moins complexe qui utilise l'architecture logicielle.
Commande	Agent	Obtenue de l'analyse d'une requête, une commande décrit l'ensemble des actions à entreprendre par un agent pour mener à terme une partie de la requête l'ayant engendrée. Une même requête peut donner lieu à plusieurs commandes.
Instruction	Manager	Opération élémentaire à envoyer au matériel. Dans le cas des NanoWalkers, une instruction est transmise sous la forme d'un message (NWMsg) à un ou plusieurs NanoWalkers. Dans le cas du contrôle de l'environnement, une instruction se résout à un appel de fonction du pilote. Une même commande peut donner lieu à une ou plusieurs instructions. Chacune d'entre elles servant à faire progresser la commande vers sa complétion.

2.4.5 Évolution des requêtes

Le cheminement d'une requête à travers les modules logiciels de la plate-forme de sa création jusqu'à sa complétion dépend de sa nature et de sa complexité. Une requête nécessitant l'action d'un ou plusieurs NanoWalkers subit plusieurs transformations avant d'être complétée. À chacune des étapes de son évolution, un nom différent l'identifie. Le tableau 2.5 explique les distinctions.

Une requête parvient au `NanOS` soit d'une application client ou d'un module interne du `NanOS` même. Très générale, elle correspond à une demande de travail plus ou moins complexe qui devra être exécutée par la plate-forme. Sa durée de vie est

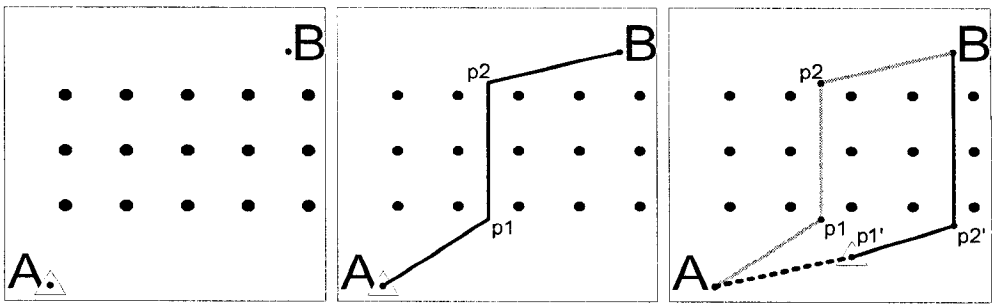
indéterminée, c'est pourquoi plusieurs requêtes doivent être capables de s'exécuter de façon parallèle sans se nuire. À ce niveau, le `NanOS` n'est responsable que de coordonner le travail des agents et de retourner une réponse au requérant. La requête est analysée et fragmentée en une série de commandes que le `NanOS` achemine aux agents capables de les exécuter.

Une commande est donc une étape nécessaire à la complétion d'une requête donnée. Lorsqu'une commande est amenée à agir directement sur les robots, elle est reçue par le `RobotAgent`. Elle est, à son tour, analysée et fragmentée en une série d'opérations élémentaires capable d'être interprétées et exécutées par un NanoWalker.

C'est à partir de ce jeu d'instructions simples compris par les robots que les commandes sont définies. Une approche *bottom-up* est utilisée pour leur élaboration. En premier lieu, les instructions sont définies. De l'étude de ces instructions, des commandes plus générales sont formées. Le même exercice est effectué pour la section «contrôle de l'environnement» de l'architecture logiciel. Une fois un ensemble de commandes possibles défini, l'agencement des commandes des différents agents permet de définir des requêtes que les applications clients pourront effectuer.

Il est à noter que le niveau d'abstraction diminue à mesure que l'on s'approche des systèmes physiques. Lorsque le résultat d'une instruction est retourné par un NanoWalker, il doit être vérifié et la commande de plus haut niveau l'ayant initiée doit être révisée afin de palier à toute erreur qui aurait pu survenir lors de l'exécution et qui introduirait un écart notable entre le résultat obtenu et celui espéré.

La figure 2.5 illustre l'évolution d'une requête de déplacement et d'une commande en fonction du résultat de la première instruction transférée au NanoWalker. Une application demande le déplacement d'un NanoWalker (représenté ici par un triangle) du point A au point B. Puisque des obstacles infranchissables (représentés



(a) Requête de l'application (b) Commande initiale (trajectoire) (c) Commande révisée

Figure 2.5 Évolution d'une requête de déplacement d'un NanoWalker du point A au point B.

par des cercles noirs) tels les porte-échantillons ou d'autres robots empêchent la ligne droite entre les deux points, une trajectoire est calculée par le `RobotAgent` et représentée par une commande. De cette commande, une série d'instructions à envoyer au robot est générée. La première instruction lui ordonne de se déplacer du point A au point $p1$. Cependant, celui-ci dévie lors de l'exécution et se retrouve à la position $p1'$. Lorsque le robot rapporte la complétion de l'opération de déplacement, la commande doit être mise à jour pour palier à toute erreur survenue. Le `RobotAgent` vérifie la position réelle du NanoWalker et révise la commande afin de tirer profit de l'erreur de position plutôt que de chercher à la corriger inutilement. Une nouvelle trajectoire vers le point B est calculée à partir du point $p1'$. La prochaine instruction envoyée au robot lui ordonne donc de se déplacer du point $p1'$ vers $p2'$. L'étape de vérification et modification est reprise jusqu'à la complétion de la requête, soit l'atteinte du point B.

2.4.6 Diagrammes de classes

Une fois l'étude des besoins et des mécanismes disponibles pour les combler avancée, une conception détaillée du système logiciel est nécessaire. Celle-ci a comme principal objectif de clarifier les idées de conception avant d'entamer la programmation. Elle revêt d'autant plus d'importance lorsqu'on sait qu'une modification au code d'une application engendre des coûts de 10 à 100 fois plus élevés que des modifications lors de sa conception[9].

Dans une conception orientée-objet, les diagrammes de classes permettent de visualiser facilement et rapidement l'ensemble du design. Ils présentent d'une façon schématique les classes appelées à travailler de concert pour solutionner un problème précis. Chaque couche architecturale du système présentée à la section 2.2 peut être représentée à l'aide de ces diagrammes. Les diagrammes traités dans cette section sont présentés à l'annexe IV.

La figure 2.6 illustre les liens entre les différents paquetages de l'architecture. Un paquetage est le regroupement d'un ensemble de classes reliées par un point en commun, par exemple, un objectif similaire ou un domaine d'application semblable. Il donne souvent lieu, par la suite, à un ou plusieurs diagrammes de classes plus détaillés.

Le paquetage `nSys` définit plusieurs *wrappers* C++ vers des fonctionnalités de base d'un système d'exploitation, telles que les *threads*, les mutex et sémaphores. Le tableau 2.6 résume son contenu. Ces classes permettent de définir une interface de programmation unique peu importe le système d'exploitation utilisé. La portabilité du code est ainsi accrue. Il faut souligner que de telles librairies existent déjà, notamment la librairie gratuite ACE[56] de Douglas C. Schmidt, mais que le développement de la librairie `nSys` a débuté avant de connaître leur existence.

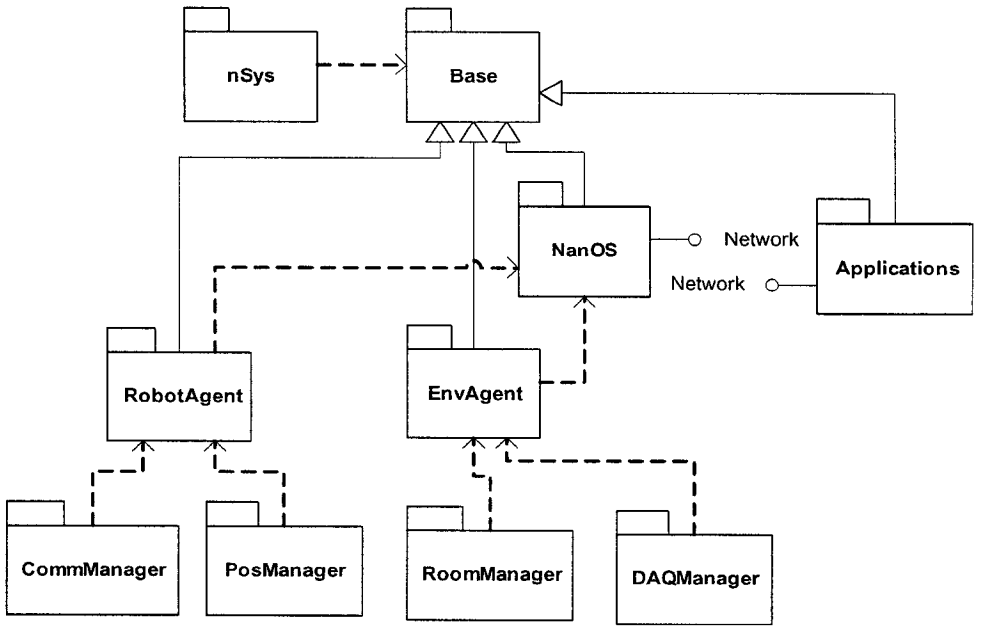


Figure 2.6 Diagramme de paquetages du système de contrôle

Après avoir tenté de migrer vers ACE, des problèmes d'utilisation dans un DLL de Windows nous ont porté à continuer avec la librairie `nSys` et, par le fait même, de parfaire notre connaissance de l'API Windows et Linux.

Les figures IV.2 et IV.3 illustrent le contenu du paquetage `Base`. Comme son nom l'indique, les classes qu'il contient sont des composants de base de l'architecture utilisés par les autres paquetages. On y retrouve notamment la définition des proxies utilisés. L'architecture du patron d'un *Proxy* suit la figure IV.1. La classe abstraite `AbsService` fournit l'interface du service. Les méthodes qu'elle définies sont surchargées différemment dans les deux classes dérivées `Proxy` et `Service`. Le `Client` instancie un `Proxy` et l'utilise comme s'il s'agissait du vrai objet `Service`.

Regardons en premier la couche `application` dont la structure est présentée à la figure IV.7. À ce jour, cette couche n'est que très peu définie puisque les applications clients amenées à utiliser la plate-forme ne sont pas encore connues. Il

Tableau 2.6 Description de la librairie nSys

Classe	Description
nException	Classe générale d'exception C++ pour la gestion des erreurs. Elle comprend une string de la librairie STL et un niveau de sévérité (Fatal, Critical, Error, Warning)
nMutex	*Wrapper* C++ autour d'un mutex du système d'exploitation.
nReadGuard	Implémentation du patron de conception *Scoped Locking*[55]. Classe générique encapsulant un mécanisme de synchronisation (mutex, sémaphore, etc.), la ressource est libérée automatiquement une fois la portée de la fonction quittée.
nSemaphore	*Wrapper* C++ autour d'un sémaphore du système d'exploitation.
nStreamOwner	Classe abstraite propriétaire d'un objet nTCPStream. Force la surcharge de la méthode ProcessMsg().
nTCPAcceptor	Implémentation du patron de conception *Acceptor-Connector* pour les flux (*stream*) TCP/IP. Classe générique recevant en paramètre le type d'objet à créer lorsqu'une nouvelle connexion survient.
nTCPStream	*Wrapper* C++ autour d'un flux TCP/IP. Un *thread* écoute continuellement sur le réseau et réagit à l'arrivée d'un nouveau message en appelant la méthode ProcessMsg() de son objet propriétaire (nStreamOwner)
nThread	*Wrapper* C++ autour d'un *thread* du système d'exploitation. À sa création, la fonction d'entrée du *thread* est spécifiée. Des méthodes pour gérer l'exécution du *thread* sont fournies. (Start(), Suspend(), Kill(), Join())
nThreadPool	Implémentation d'un *threadpool* de taille dynamiquement ajustable à partir d'objets nThread. Un point d'entrée pour un *thread* est enfilé et les différents *threads* défile un à un et exécute la fonction spécifiée dans la file.
nThreadSafeQ	Wrapper C++ autour d'une file avec intégration de mécanismes d'accès exclusif dans un environnement *multi-threads*.

est néanmoins possible d'établir certains paramètres communs. Toutes les applications accèdent au même système, les fonctionnalités accessibles sont les mêmes et le traitement interne qu'effectue la plate-forme doit être invisible du point de vue des applications. La classe `ClientAppInterfaceProxy` fournit cette encapsulation des méthodes de la plate-forme accessible par les applications clients. Les applications concrètes héritent de la classe abstraite `AbsConcreteClientApp` qui possède le proxy. Ce dernier maintient un lien avec la plate-forme réelle par l'entremise d'un canal de communication `nTCPStream` duquel il hérite. La classe `AbsConcreteClientApp` dérive de la classe `AbsClientApp` définie dans le paquetage `Base`. Celui-ci sert à définir l'interface commune avec les proxies instanciés par le `NanOS` lorsqu'une nouvelle application se connecte au système : `ClientAppProxy`. Elle dérive également de la classe `nStreamOwner`. Cette classe abstraite a pour unique but de forcer l'implémentation d'une méthode `ProcessMsg()` dans toute classe qui possédera un proxy. Cette méthode est appelée par le proxy lorsqu'il reçoit un message réseau. `ProcessMsg()` est donc responsable d'analyser le message reçu et d'agir en fonction de son contenu.

Chaque application concrète peut posséder une interface graphique de type `AbsUI`. Celle-ci force la surcharge de méthodes simples pour rapporter des erreurs ou la mise à jour de l'état de l'application à n'importe quelle classe qui en dérive.

La couche `NanOS` cumule deux fonctions : l'interaction avec les applications et la surveillance de la plate-forme. La classe `ClientAppInterface` reçoit et traite toute demande de connection à la plate-forme par l'entremise de l'objet `nTCPAcceptor` qu'elle possède. Lorsqu'une connection d'un client est acceptée, l'accepteur crée un `ClientAppProxy`. Lors de sa création, le nouveau proxy s'enregistre auprès du `ClientAppRegistry` ayant pour mission de maintenir la liste des applications client présentement connectées au système et de les associer avec un numéro d'identification unique utilisé à travers le reste de l'architecture.

Le propriétaire des proxies demeure le `ClientAppInterface` et c'est lui qui analyse le contenu des messages reçus. Lorsqu'une requête est reçue, il la transfère au `RequestCreator` qui instancie le bon objet de requête dérivé de `AbsRequest`. La requête concrète ainsi créée est enfilée dans une file du `RequestHandler`. C'est ce dernier qui a la charge de gérer l'exécution parallèle d'un certain nombre de requêtes. Pour ce faire, il implémente un *threadpool* avec un objet de la classe `nThreadPool`. Chaque requête encapsulée selon le patron commande surcharge à sa manière la méthode virtuelle `Execute()` de la classe mère. Le *threadpool* appelle cette méthode et «active» ainsi l'objet requête jusqu'à sa complétion ou son avortement.

Les requêtes concrètes communiquent avec les agents du système par l'entremise d'un aiguilleur de commandes appelé `AgentInterface`. Il est responsable de diriger vers le bon agent une commande ordonnée par une requête. Bien que le nombre d'agents présents dans le système n'est à ce jour que de deux, cette approche a l'avantage de permettre la modification facile des responsabilités des différents agents sans avoir à modifier outre mesure la structure des requêtes. L'`AgentInterface` communique avec les agents par l'entremise de proxies et de canaux de communication de la même façon que les applications client avec le système.

Le deuxième rôle du `NanOS` est assuré par le `NWMonitor`. Ce module de surveillance a la responsabilité de corréler l'information d'environnement avec l'état actuel des robots afin d'assurer leur survie. Lorsque l'intégrité d'un robot ou de son travail est compromise, le `NWMonitor` signale la situation au `NanOS` qui prend action. Les actions entreprisent par le `NanOS` ont priorité sur l'ensemble des requêtes des clients afin d'assurer une intervention rapide à toute situation dangereuse. Ainsi, à part l'application client elle-même, seul le `NanOS` a le droit d'interrompre, d'annuler ou de retarder une requête.

Le `RobotAgent` (voir Figure IV.5) reçoit les ordres du `NanOS` par son proxy sous la forme de commandes. Elles sont transférées au `CmdCreator` à qui revient la tâche d'instancier des commandes concrètes dérivant de la classe abstraite `AbsNWCmd`. La commande créée est ajoutée à la file de commandes que contient l'objet `NanoWalker` adressé. Cette classe `NanoWalker` se veut un cliché *snapshot* virtuel de l'état présent d'un NanoWalker physique. Lorsqu'une commande est ajoutée à la file du `NanoWalker`, il la modifie de façon à tenir compte de sa calibration et des caractéristiques physiques qui influence son comportement.

Il y a autant de classes «Commande» dérivées qu'il y a de types de commandes supportées. Puisque chaque type de commande est fondamentalement différent, la façon de les traiter l'est également. Ainsi, à chaque type de commande est associé un analyseur (*parser*) qui la décompose en instructions. Le `AbsCmdParser` ainsi que ses classes dérivées connaissent donc de façon exhaustive l'ensemble des instructions supportées par les robots (voir section 2.6.1). Chacune des instructions engendrées donne lieu à un message qui peut être envoyé tel quel au robot. Puisque la commande doit être révisée de façon systématique après chaque instruction, il ne sert à rien de prévoir d'avance toutes les instructions potentielles à envoyer à un robot. Le `CmdParser` concret devra donc permettre de convertir les commandes par fragments.

Tour à tour, le `CommManager` parcourt les différents NanoWalker et retire la prochaine instruction à leur être envoyée. Il transmet le message associé via le `IRDriver`. Le `PosManager` utilise le PSD pour recueillir l'information sur l'emplacement des deux LEDs de positionnement IR des robots et transfère cette information au `RobotAgent` qui met à jour l'objet `NanoWalker` correspondant à la position acquise.

Le second agent contrôle les systèmes relatifs à l'environnement des NanoWalkers. Sa structure est présentée à la figure IV.6 ; le `EnvAgent` en est l'entité principale. La

classe `TemperatureMap` cartographie la température dans la chambre de refroidissement. Les valeurs de température proviennent principalement de thermocouples dans la chambre, mais également de l'information des senseurs des robots après avoir transitée par le `NanOS` jusqu'au `EnvAgent`. Ces données sont simplement traitées comme des couples température-coordonnées supplémentaires, ainsi, le `EnvAgent` demeure indépendant des robots. Un contrôleur PID se base sur cette carte thermique et tâche de régulariser la température dans la chambre en ordonnant au `RoomManager` le pourcentage d'ouverture de la valve d'azote[23]. Le DAQManager se charge d'acquérir les données de vibrations et de température du module NI-SCXI[6].

Le `EnvAgent` utilise également le `PwrManager` pour contrôler le bloc d'alimentation du *Powerfloor* à travers une interface GPIB. L'étendu du contrôle qu'il applique se limite cependant à une surveillance de la tension, du courant et à implanter un mécanisme d'arrêt d'urgence.

2.4.7 Diagrammes de séquence

Les diagrammes de classes présentent la structure logicielle d'un système informatique, mais ils ne procurent aucune information temporelle sur la séquence d'opérations à effectuer pour accomplir un travail particulier. C'est le rôle des diagrammes de séquence d'illustrer l'échange de messages entre les différentes entités de l'architecture. Les diagrammes relatifs à la présente section sont présentés à l'annexe V.

L'interaction d'une application client avec la plate-forme est présentée à la figure V.1. Le processus d'initialisation commence par une demande de connexion du client. Cette demande, reçue par l'accepteur du patron *Acceptor-Connector*, résulte

en la création d'un proxy encapsulant le canal de communication. Le proxy créé a la responsabilité de signaler au `ClientAppInterface` son existence. Le `ClientAppRegistry` lui assigne un numéro d'identification unique qui devra être transmis avec toute communication future. Le *thread* de réception des messages réseau du `nTCPStream` sous-jacent au proxy est alors démarré. L'initialisation de la connexion est ainsi complétée. Le `ClientAppInterface` demande de l'information supplémentaire sur la nouvelle application telle que son nom, sa description, etc. Après avoir répondu, l'application client peut commencer à envoyer des requêtes au système. Le proxy reçoit le message et le transfère au `ClientAppInterface` pour analyse. L'instanciation du bon objet «requête» est délégué au `RequestCreator`. Elle est ensuite transférée au `RequestHandler` pour traitement. Avant d'obtenir la réponse, un objet `Future` est envoyé à l'application client à titre d'accusé de réception. Lorsque la réponse à la requête est disponible, elle est transmise à l'application client. Cette série d'événements est répétée jusqu'à la déconnexion de l'application.

Si l'on reprend l'exemple d'un déplacement de NanoWalker commandé par une application, la figure V.2 montre l'évolution des messages échangés par chacun des modules de l'architecture. On y voit qu'une fois la requête rendue au `RequestHandler`, celui-ci lance son exécution. Dans le cas d'une requête de déplacement, seul le `RobotAgent` est concerné, la requête lui communique ainsi la commande de déplacement. Il fait appel au `CmdCreator` pour instancier le bon objet «commande», c'est-à-dire une trajectoire contournant les différents obstacles fixes et mobiles se trouvant sur le *Powerfloor*. Une fois la commande obtenue, elle s'ajoute dans la file du NanoWalker adressé. Il l'ajuste alors en fonction de sa calibration et la décompose en instructions élémentaires par l'entremise du `MoveCmdParser`. Le `CommManager` s'affaire parallèlement à interroger chaque robot tour à tour pour mettre à jour leur état. Quand un robot est prêt à recevoir une instruction, il demande au `RobotAgent` l'accès à l'objet `NanoWalker` dans le but de retirer la pro-

chaine instruction devant lui être transmise. L'instruction est traduite en message `NWMsg` et envoyée au NanoWalker qui accuse immédiatement sa réception. Le robot exécute et termine l'instruction. Ce nouvel état sera détecté au prochain passage du `CommManager`. Une série de mise à jour subséquentes suit cet événement et est présentée à la figure V.3. On remarque qu'une fois le `RobotAgent` informé de la complétion de l'instruction, il vérifie le résultat en demandant la position réelle du NanoWalker au `PosManager` et met à jour la commande. Les prochaines instructions envoyées au NanoWalker proviendront donc de cette commande révisée. Les mises à jour sur la progression de la requête sont ensuite propagées à travers l'architecture jusqu'à l'application client l'ayant initiée.

Plusieurs autres diagrammes de séquence pourraient être présentés, mais ils suivent tous l'évolution générale de la requête de déplacement.

2.5 Synchronisation des requêtes

L'architecture distribuée du logiciel rend problématique la synchronisation des différents *threads* d'exécution. Ces *threads* peuvent se retrouver sur différentes machines composants le système et une requête effectuée par un certain *thread* d'un processus demandeur vers un processus distant doit être envoyée sur le réseau. La réponse à cette requête est reçue après un temps indéterminé et par un *thread* du processus demandeur différent de celui qui l'a effectuée. Dans le cas où la réponse serait essentielle à la suite de l'exécution, c'est-à-dire que l'on souhaite avoir des requêtes «bloquantes», il est nécessaire d'établir un mécanisme de synchronisation permettant de suspendre l'exécution d'un *thread* tant que la réponse à une requête particulière n'est pas obtenue ou qu'un certain temps de *timeout* ne soit atteint.

Ceci requiert un usage judicieux de sémaphores de contrôle. La stratégie envisagée

se base sur le principe du *threadpool*. Une banque de sémaphores est créée. À chaque fois qu'un *thread* demandeur désire qu'un requête qu'il effectue soit bloquante, il demande un sémaphore libre à cette banque et le bloque. L'identificateur du sémaphore bloqué est associé à la requête envoyée. Ce même identificateur doit être inclus dans la réponse de sorte qu'une fois celle-ci obtenue par le *thread* du proxy, le bon sémaphore puisse être débloqué. Si un *timeout* survient, c'est que la réponse n'est pas arrivée à temps, le sémaphore est alors automatiquement débloqué. Par contre, cela ne signifie pas nécessairement que la réponse n'arrivera jamais. Si le sémaphore a été réassigné à une autre requête bloquante, il ne faut pas le débloquer. Le *thread* receveur doit donc préalablement s'assurer que la requête est encore en attente avant de procéder au déblocage du sémaphore, sans quoi, une incohérence dans le système serait introduite.

2.6 Contrôle des robots

La communication avec un robot est toujours initiée par le système de contrôle. Sauf exception, il ne communique qu'avec un seul robot à la fois et, tour à tour, il parcourt séquentiellement (en *round-robin*) l'ensemble des robots. Un quantum de temps pour la communication avec un robot donné est alloué et celle-ci continue tant qu'il n'est pas écoulé. Après quoi, le système passe au robot suivant. La valeur de ce quantum est fonction du nombre de robot à contrôler. On souhaite s'assurer que chaque robot sera interrogé à une fréquence minimale, par exemple, une fois par seconde. Plus le nombre de robots est grand, plus le quantum de temps alloué par robot doit être petit pour maintenir la contrainte fixée. La valeur pratique de la fréquence d'interrogation minimale devra être trouvée expérimentalement. Les paramètres qui l'influencent sont notamment la vitesse du changement de température des robots et la quantité de données STM que le robot peut emmagasiner

avant de devoir les transmettre par infrarouge.

Pour chaque message envoyé par le système de contrôle à un NanoWalker particulier, le robot a l'obligation d'émettre une réponse à l'intérieur d'un délai préétabli, sans quoi, il sera considéré comme inopérationnel. Un robot ne prend jamais la décision d'envoyer de l'information au système de contrôle, l'ordre explicite de transmettre un message doit lui être donné. La réponse qu'il émet varie selon le type de message qu'il reçoit du système de contrôle, mais on distingue trois sortes de réponses :

1. Retour de l'information demandée ;
2. Accusé de réception avec promesse d'exécution de l'instruction reçue (`ACK`) ;
3. Erreur sur la communication (`NACK`).

L'initiation de la communication s'effectue toujours de la même façon, selon l'algorithme suivant :

```
Tant que le quantum de temps alloué n'est pas dépassé
   Demander l'état du NW
      Si l'état == OK et READY
         Envoyer la prochaine instruction de la liste d'instruction
         Attendre la réponse
         Si timeout
            Propager l'erreur de communication
      Sinon
         Si l'état != OK
            Propager l'erreur aux couches supérieures
```

La première étape consiste toujours à interroger le robot sur son état. La réponse est analysée afin de s'assurer que le robot est apte à recevoir une instruction. Dans le cas où ce test s'avère vérifié, la prochaine instruction est envoyée et on attend l'accusé de réception. On boucle ensuite jusqu'à l'écoulement du quantum de temps.

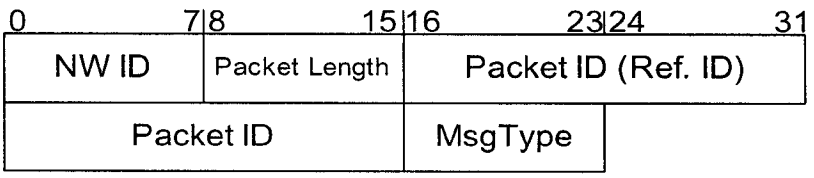


Figure 2.7 Entête des messages échangés avec les NanoWalkers

Le traitement des messages et l'analyse de leur contenu induit inévitablement des délais qu'il faut minimiser. L'analyse de l'état se résume à un simple branchement sur la valeur d'un champ du message reçu. Le nombre d'instructions comprises par un robot est amené à changer à mesure que les efforts de développement progresseront et que de nouveaux algorithmes seront développés. Pour l'instant, la section 2.6.1 présente les instructions prévues en plus amples détails.

2.6.1 Jeu d'instructions

La figure 2.7 schématise le format de l'entête commune à tous les messages échangés entre le système de contrôle et un robot.

Ces 56 bits sont structurés de la façon suivante :

NW ID :	8 bits. Numéro d'identification unique associé à un NanoWalker. Lors de l'envoi d'un message par le système de contrôle, ce champ spécifie le robot destinataire. Lors de l'envoi d'un message par un robot, ce champ spécifie le robot émetteur. Note : l'identificateur 255 est réservé pour un message de type *broadcast* destiné à tous les robots.
Packet Length :	8 bits. Taille totale du message (en octets) incluant l'entête.
Packet ID :	32 bits. Numéro d'identification du message envoyé à un NanoWalker. Ce champ est utilisé comme numéro de référence lors de la réponse du NanoWalker.
Msg Type :	8 bits. Type de message. Ce champ permet d'identifier la nature du message transmis et, par conséquent, de connaître la nature des données qu'il contient.

L'analyse des messages dépend de leur nature. Certains n'ont aucun paramètre et le champ `Msg Type` à lui seul dicte la marche à suivre, d'autres possèdent un ou plusieurs paramètres devant être traités. La longueur des messages est donc variable et dépend du nombre et de la nature des paramètres.

2.6.1.1 État

Le tout premier message envoyé lors de l'établissement de la communication avec un NanoWalker est une interrogation sur son état. Le NanoWalker interrogé retourne immédiatement son état sous la forme d'un message de 112 bits illustré à la figure 2.8. Mis à part les bits de l'entête, on retrouve les champs suivants :

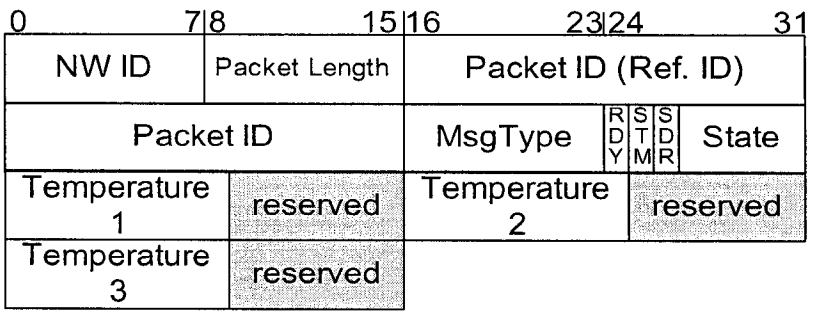


Figure 2.8 Format d'un message d'état d'un NanoWalker

RDY :	Ready. 1 bit. Indique si le robot est prêt à recevoir une nouvelle instruction. 0 si le robot est présentement occupé 1 s'il est prêt.
STM :	1 bit. Indique si le STM du robot est présentement en cours d'utilisation. 1 signifie que le robot utilise son STM 0 signifie que le robot utilise ses pattes piézoélectriques.
SDR :	STM Data Ready. 1 bit. Indique qu'un bloc de données provenant du STM est prêt à être transféré au système de contrôle et que ce dernier devrait en demander la transmission.
State :	5 bits. État actuel du robot parmi les 2^5 états possibles.
Temperature X :	9 bits. Valeur du capteur de température no. X du NanoWalker en complément à 2 entre $-55^{\circ}C$ et $+125^{\circ}C$ avec une résolution au demi-degré près.

Les valeurs de température des capteurs sont sur neuf bits. Mais puisque la SRAM sur le circuit *Flex* est configurée en mots de 16-bits, sept bits par valeurs sont donc inutilisés.

2.6.1.2 Déplacement

Pour contrôler le déplacement des robots et définir un bon format d'instruction, il importe de comprendre la méthode par laquelle les déplacements leurs sont in-

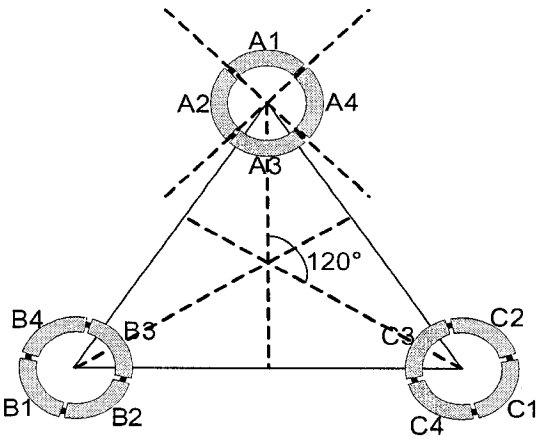


Figure 2.9 Disposition des pattes piézoélectriques et de leurs électrodes sur le NanoWalker

culqués. Chaque NanoWalker repose sur trois pattes piézoélectriques, chacune possédant quatre électrodes selon la configuration présentée à la figure 2.9. En appliquant une forte tension électrique, une déformation du tube piézoélectrique est induite. Cette réponse mécanique est directement proportionnelle aux valeurs de tensions appliquées sur les électrodes.[42] L'obtention d'un déplacement est rendue possible grâce à l'application d'une série d'impulsions électriques sur les électrodes des pattes selon une séquence prédéterminée. Bien que théoriquement la tension pouvant être appliquée à une électrode se situe dans un intervalle continu, le circuit actuel du NanoWalker ne permet que l'application de deux tensions discrètes : soient $+150V$ et $-150V$. Ces deux valeurs discrètes peuvent être représentées par un 1 et un 0 respectivement. La figure 2.10 présente un exemple d'impulsions envoyées à une électrode d'une patte et sa représentation binaire. Un mouvement général du NanoWalker est obtenu en bouclant un certain nombre de fois, et à une certaine fréquence, sur une série d'impulsions précise.

Les déformations rapides des pattes piézoélectriques ainsi produites font vibrer le robot et celui-ci se déplace en sautant littéralement d'un endroit à un autre.

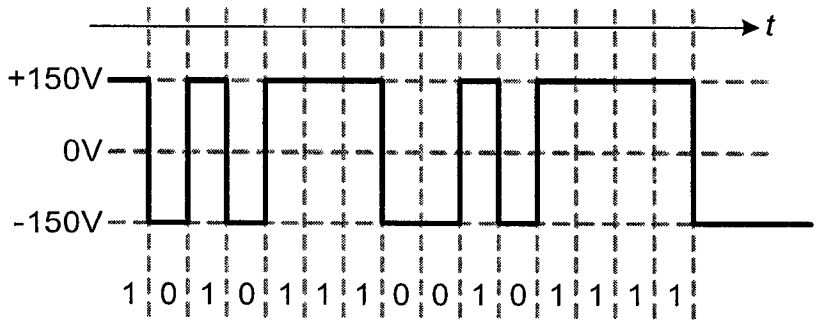


Figure 2.10 Exemple de série d'impulsions à envoyer à une électrode d'une patte

Chacune des électrodes des pattes peut recevoir une série d'impulsions différente et chacune peut être excitée à une fréquence différente.

Trois types de déplacements sont possibles pour le NanoWalker : 1) rectilignes, 2) rotations et 3) une combinaison des deux. Seules les séquences d'impulsions appliquées aux pattes, la fréquence d'application par pattes ainsi que le nombre de répétitions varient. On peut envisager qu'une seule instruction puisse combler tous les besoins. À l'aide d'une matrice 2×3, il est possible de définir n'importe quelle rotation et translation $2D$. En ajoutant des paramètres supplémentaires pour la vitesse de déplacement et la vitesse de rotation, une seule instruction définie l'ensemble des mouvements possibles.

$$mvt = \begin{bmatrix} \sin(\theta) & \cos(\theta) & \Delta x \\ \cos(\theta) & -\sin(\theta) & \Delta y \end{bmatrix} \tag{2.1}$$

Avec l'aide de Na-Mi Bae, stagiaire au Laboratoire, certaines séquences de déplacements élémentaires basées sur la méthode *push-slip*[42] ont été identifiées. Le tableau 2.7 présente la séquence théorique à appliquer aux électrodes pour obtenir un déplacement en ligne droite.

Tableau 2.7 Séquence de bits à appliquer aux électrodes des pattes pour obtenir un mouvement rectiligne

Mouvement rectiligne vers la patte A												
Patte	A				B				C			
Électrode	A1	A2	A3	A4	B1	B2	B3	B4	C1	C2	C3	C4
Bit 0	1	1	1	1	0	0	1	1	0	1	1	0
Bit 1	0	0	0	0	1	1	0	0	1	0	0	1

La principale décision à prendre est celle de l'emplacement où s'effectue la conversion de la matrice de déplacement en série d'impulsions électriques. Deux options sont envisageables : soit qu'elle s'effectue à même le DSP du robot ou soit qu'elle est faite dans le système de contrôle avant même l'envoi de l'instruction proprement dite au NanoWalker. Dans les deux cas, la conversion à effectuer est la même et l'implémentation est similaire. Comme première approche, la conversion avant l'envoi au robot est effectuée. Les valeurs de 12 bits sont stockées dans le SRAM et le rôle du DSP se limite à un simple transfert de données. Ceci permet de soulager le processeur embarqué du robot en minimisant sa charge de traitement. Par contre, cela a pour effet de limiter l'autonomie des robots. Dans le futur, si la capacité de traitement du DSP et la mémoire disponible le permet, la responsabilité de cette conversion pourra être transférée au robot.

Le format du message de déplacement (voir Figure 2.11) choisi est le suivant :

SeqLength :	8 bits. Longueur de la séquence de bits
Freq :	8 bits. Fréquence d'application de la série d'impulsions
NbRep :	16 bits. Nombre de répétitions à effectuer sur la séquence de bits.
SeqBits$X$:	16 bits. Valeur des bits à appliquer à chacune des 12 électrodes des pattes.

La longueur totale du message dépend de la longueur de la séquence de bits. Son minimum est de 104 bits alors que sa longueur maximale est de 525 octets. Même

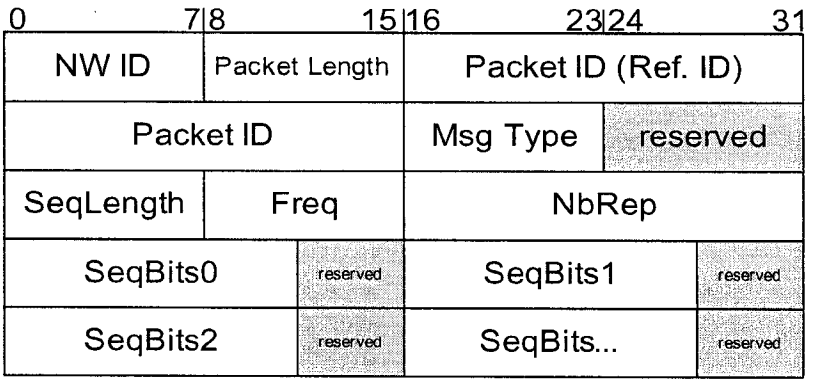


Figure 2.11 Format d'un message de déplacement

si la fréquence d'application sur chacune des pattes peut différer en théorie, en pratique, une seule est utilisée. La plus petite fréquence pour laquelle les autres sont toutes un facteur est celle à prendre. Ceci permet de stocker le même nombre de bits pour chacune des pattes dans la mémoire SRAM du robot configurée en mots de 16 bits. Ainsi, une séquence deux fois plus lente que la fréquence utilisée verra chacun de ses bits répétés deux fois afin d'obtenir le même effet que si elle était réellement deux fois plus lente.

2.6.1.3 Balayage du STM

Un scan du microscope à effet tunnel est commandé par plusieurs messages. Premièrement, tous les paramètres du balayage sont envoyés d'un bloc. Par la suite, une instruction ordonne au robot de démarrer ou d'interrompre le scan. Cette façon de faire permet de modifier les paramètres sans pour autant devoir arrêter le scan en cours. La configuration des paramètres se fait d'un seul coup plutôt qu'avec plusieurs messages très courts qui gaspilleraient davantage de bande passante.

La figure 2.12 illustre le message de modification des paramètres du PiezoScan. Une explication sommaire des différents champs est donnée ici, le lecteur est prié

de lire le chapitre 3 afin d'obtenir une meilleure compréhension du fonctionnement du microscope et de l'origine des différents paramètres.

Gain :	1 bit. Gain de l'amplificateur opérateur du circuit du PiezoScan du *Flex*.
ScanRes :	2 bits. Résolution de l'image. `00` => Résolution de 128 × 128 pixels `01` => Résolution de 256 × 256 pixels `10` => Résolution de 512 × 512 pixels `11` => Résolution de 1024 × 1024 pixels
Fast Vx Increment :	16 bits. Incrément de la tension V_x à ajouter à chaque déplacement du piézo sur l'axe rapide. (valeur déjà convertie pour le DAC)
Fast Vy Increment :	16 bits. Incrément de la tension V_y à ajouter à chaque déplacement du piézo sur l'axe rapide. (valeur déjà convertie pour le DAC)
Slow Vx Increment :	16 bits. Incrément de la tension V_x à ajouter à chaque déplacement du piézo sur l'axe lent. (valeur déjà convertie pour le DAC)
Slow Vy Increment :	16 bits. Incrément de la tension V_y à ajouter à chaque déplacement du piézo sur l'axe lent. (valeur déjà convertie pour le DAC)
Fast Vx Update Period :	16 bits. Nombre de période de mise à jour à sauter avant d'ajouter l'incrément V_x à l'axe rapide.
Fast Vy Update Period :	16 bits. Nombre de période de mise à jour à sauter avant d'ajouter l'incrément V_y à l'axe rapide.
Nb Fast Axis Updates :	16 bits. Nombre total de mises à jour sur l'axe rapide afin de balayer la taille totale du scan.
Vx Offset :	16 bits. Décalage du centre en X du balayage par rapport à la position de repos. (valeur convertie pour le DAC)
Vy Offset :	16 bits. Décalage du centre en Y du balayage par rapport à la position de repos. (valeur convertie pour le DAC)

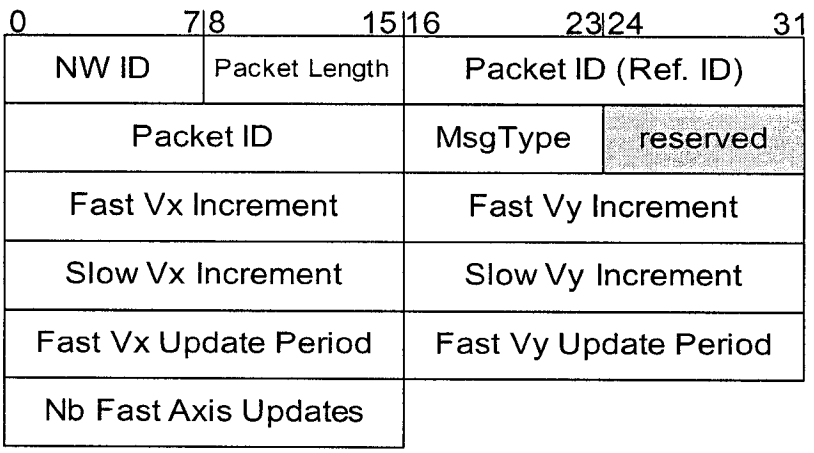


Figure 2.12 Format d'un message de configuration du balayage du piézo du STM

2.6.1.4 Tip Engage

Comme pour le balayage, un premier message de configuration est envoyé contenant les paramètres du contrôleur PID. Ensuite, la commande d'approche de la pointe STM jusqu'à l'obtention d'un courant tunnel est donnée. Encore une fois, le lecteur est référé au chapitre 3 pour les détails du STM.

PID Term$n$: 16 bits. Termes invariables précalculés de l'équation 3.34
$ln(SetPoint)$: 16 bits. Valeur du logarithme naturel du courant tunnel à maintenir. (la référence du PID)

2.6.1.5 Retrieve STM data

Le DSP du robot ne possède pas assez de puissance pour analyser les résultats d'un scan STM. Et, la mémoire dont il dispose n'est pas suffisante pour stocker une image complète. Les résultats doivent donc être transmis bout par bout au système de contrôle. Un mécanisme de double tampons (*double buffering*) permet

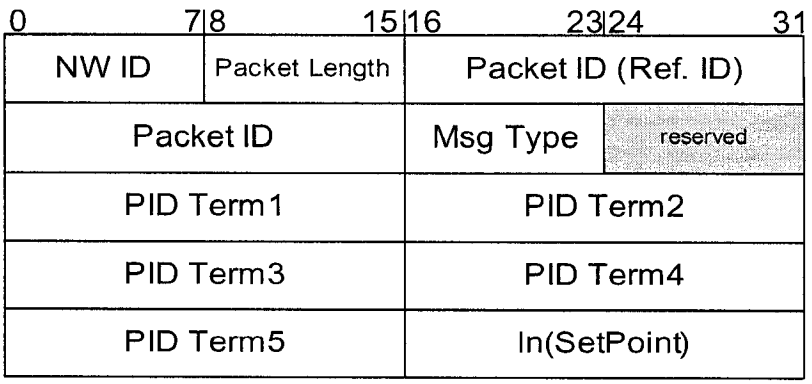


Figure 2.13 Format d'un message de configuration du contrôleur PID du STM

de remplir une section de mémoire pendant que l'autre attend d'être vidée. Lorsqu'une zone tampon de données est pleine, le message d'état du robot indique au `CommManager` de demander une transmission des données accumulées. Cette demande devrait venir avant l'envoi de toute nouvelle instruction afin d'éviter de trop retarder la transmission et que les deux tampons se retrouvent simultanément pleins, les données contenues se verraient alors écrasées.

Plusieurs autres messages sont à définir, par exemple des messages pour reprogrammer la mémoire FLASH via l'IR, pour gérer l'algorithme d'approche IAPA décrit au chapitre 4 et pour des algorithme futurs tels que la recherche d'atomes précis ou l'assemblage moléculaire.

Certains algorithmes doivent obligatoirement être implémentés à même le DSP, notamment les fonctionnalités de comptage d'atomes et de suivi d'un atome particulier. En effet, la latence inhérente à la communication infrarouge rend impossible la compensation en temps réel des effets de dérive thermique *thermal drift* qui déplacent la pointe STM à la fois verticalement et horizontalement. L'amplitude de cette dérive est fonction des matériaux utilisés dans l'assemblage du microscope et de la vitesse de variation de température, mais une dérive d'un diamètre atomique

par seconde n'est pas rare pour un STM opéré à température de la pièce[41]. Dans le cas présent où les NanoWalkers chauffent dans un environnement froid, le PID devra être particulièrement efficace pour minimiser la dérive thermique.

2.7 Format des messages réseaux

La section 2.3 énonce que les niveaux `agent`, `NanOS` et `application` communiquent par échange de messages sur le réseaux. Peu importe le type de protocole réseau utilisé, le format interne des données reste identique. Chaque message est constitué de deux sections : la première, de longueur fixe, est commune à chaque type de messages alors que la deuxième, de longueur variable, est fonction de la nature du message. La structure des messages est la suivante :

```
struct netMsgStruct
{
   unsigned short msgType;     // 2 bytes
   unsigned short srcId;       // 2 bytes
   unsigned short timestamp;   // 2 bytes
   unsigned short reserved;    // 2 bytes
   void*          pParameters; // 4 bytes (start of variable nb. of bytes)
};
```

Le champ `msgType` indique la nature et la fonction du message envoyé. Le champ `srcId` spécifie l'identificateur d'application, tel qu'assigné par le `NanOS` au moment de la connection, ayant émis le message. Le champ `timestamp` se veut une marque temporelle du moment où l'information fut originalement émise. Dans le cas d'un retour d'information du NanoWalker, ce champ devrait contenir une valeur de temps qui soit le plus près possible du temps où l'information était valide dans le robot physique. Ainsi, elle devrait être générée dès la première interprétation du

message infrarouge reçu du NanoWalker. Le champ `pParameters` sert à indiquer le début d'une région de longueur variable où sont stockés les paramètres particuliers au message transmis.

La présence du pointeur `void* pParameters` est trompeuse puisque celui-ci ne doit pas être utilisé comme un réel pointeur. En effet, un message à transmettre par réseau (ou tout autre médium de communication) est traité comme une série d'octets dans un espace mémoire continu. Lors de la création du message réseau, il est donc important de prévoir d'avance la taille des données variables afin de créer une zone mémoire continue suffisamment grande pour les contenir. Ainsi, le code suivant ne peut être utilisé pour créer cette zone.

```
netMsgStruct msg;
msg.pParameters = (void*) new unsigned char[50];
```

Il faut plutôt privilégier le code ci-dessous.

```
netMsgStruct* pMsg;
// Calculate the total length (-4 because of the size of void*)
unsigned short msgTotalLen = sizeof(netMsgStruct) - 4;
totalLen += SizeOfVariableZone;
pMsg = (netMsgStruct*) new unsigned char[msgTotalLen];
```

Sur réception d'une message, la méthode virtuelle `ProcessMsg()` du `nStreamOwner` possédant le canal de communication est appelée. En C++, cette méthode a l'allure générale suivante :

```
void ProcessMsg(const unsigned char* pData, const unsigned short& length)
```

```
{
   // Cast pData into communication structure format to be able
   // to get known parameters
   const netMsgStruct* const pMsg =
         reinterpret_cast<const netMsgStruct* const>(pData);

   // Position a pointer on the first parameter
   // use 'const' to make sure that the data will not be modified
   const unsigned char* pParam =
         reinterpret_cast<const unsigned char*>(&(pMsg->pParameters));
   unsigned short dataLength = length - (sizeof(netMsgStruct)-4);

   switch(pMsg->msgType)
   {
   case SOME_MSG_TYPE:
      // Retrieve parameters from pParam
      HandleMsgType(pMsg->srcId, parameters...);
      break;

   default:
      break;
   }
}
```

2.8 Conclusion

Le robot NanoWalker se fait attendre et évolue constamment, mais le développement logiciel de la plate-forme doit progresser. Plusieurs modules sont codés et l'intégration est entamée. La taille et la complexité de cette architecture distribuée augmente rapidement à mesure que de nouvelles fonctionnalités sont à supporter. Une vision d'ensemble est primordiale dans sa conception, sans quoi plusieurs des modules développés nécessiteront des efforts titanesques pour procéder à leur intégration. L'utilisation des patrons de conception facilite son maintient et favorise sa robustesse.

L'architecture établie permet le développement d'une solution unifiée pour un sys-

tème de contrôle d'une flotte de NanoWalkers. Les deux principaux aspects à contrôler sont tous deux personnifiés par un agent. L'`EnvAgent` permet un contrôle précis sur l'environnement afin de garantir la survie des robots, tandis que le `RobotAgent` a le rôle d'assurer une communication efficace et la coordination de la flotte de robots.

Il est impensable d'attendre une flotte complète avant de tester son contrôle. L'idée d'un simulateur fonctionnel de NanoWalker est donc apparue. Le but de ce simulateur est de tester la plate-forme de support d'une façon purement virtuelle sans nécessiter la présence de NanoWalkers physiques. L'architecture ne devrait cependant pas être «consciente» de l'absence de NanoWalkers. Sylvain Boissé[10] fut mandaté à l'automne 2002 et hiver 2003 pour développer une simulation fonctionnelle de la communication des NanoWalkers. Ceci fournit un outil facilement modifiable pour valider le protocole de communication et tester la réaction du système à certains stimuli, tels qu'une erreur de transmission ou la «mort» d'un robot. Implanté sur une machine indépendante, le simulateur communique avec la plate-forme via un lien infrarouge exactement comme dans le système réel.

Le développement parallèle de ce simulateur permettrait de tester la coordination d'une flotte entière de robots sans risquer de faire une erreur fatale aux répercussions financières et temporelles énormes. Une simulation physique réaliste est même envisageable.

CHAPITRE 3

CONTRÔLE DU MICROSCOPE À EFFET TUNNEL

D'un point de vue informatique, les STM sont traditionnellement contrôlés par un processeur 32-bits capables d'opérations à points flottants. Celles-ci sont particulièrement importantes lors des calculs sur le courant tunnel. Le DSP TMS320C25-50 ne possède cependant qu'un bus de données 16-bits et n'est capable d'effectuer que des opérations «entières».

L'approche préconisée pour le contrôle du STM consiste à réduire au maximum le traitement numérique à bord du processeur embarqué et de le transférer au système de contrôle externe présenté au chapitre 2. Les calculs inévitables dans le DSP devront être effectués en arithmétique à virgule fixe (*fixed-point arithmetic*). Les notions théoriques nécessaires sont brièvement introduites à la section 3.1.

Le reste du chapitre est consacré au développement du contrôle du STM, la description des prototypes, les tests effectués et les résultats obtenus.

3.1 Arithmétique en virgule fixe

L'arithmétique en virgule fixe n'utilise que des nombres entiers pour représenter des valeurs réelles. Elle est particulièrement utile lorsque l'on doit faire des calculs mathématiques sur des valeurs réelles à l'aide de microprocesseurs ne supportant que des types entiers tels que `int` ou `short`.

Connaissant la représentation binaire d'un nombre, on peut décider arbitrairement

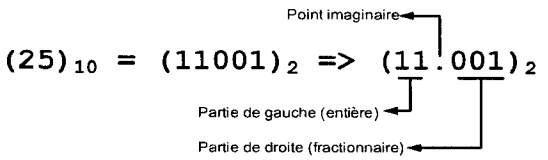


Figure 3.1 Nombre selon une représentation virgule fixe

de le considérer comme deux parties de part et d'autre d'une virgule imaginaire comme à la figure 3.1. La valeur réelle des bits doit donc être pondérées en fonction de leur position par rapport à cette virgule imaginaire. Ainsi, dans l'exemple de la figure 3.1, $(11.001)_2$ se traduit par deux bits indiquant la partie entière et les trois bits de droite indiquant la partie décimale. On note cette représentation à virgule fixe par «FP2.3». La valeur réelle codée est $3,125$ bien que d'un point de vue strictement logiciel, la valeur stockée ne possède pas de virgule imaginaire et est donc $3,125 \times 2^3 = 25$.

D'une façon plus formelle, un nombre virgule fixe se représente selon l'équation 3.1[14]. Seule la valeur Q est gardée dans la mémoire de l'ordinateur, les autres étant purement conceptuelles. Les valeurs de F et de B sont habituellement fixées à $1,0$ et 0 respectivement, de sorte que la valeur réelle puisse être retrouvée par la simple multiplication d'une puissance de deux avec la valeur Q.

$$V \approx \widetilde{V} = SQ + B \tag{3.1}$$

Tableau 3.1 Intervalle des valeurs selon la représentation point fixe utilisée, a) non signé b) signée

Représentation	Intervalle	Résolution
a) $p.q$	$\left[0, 2^p - \frac{1}{2^q}\right]$	$\frac{1}{2^q}$
b) $sp.q$	$\left[-2^p, +2^p - \frac{1}{2^q}\right]$	$\frac{1}{2^q}$

où

V = valeur réelle
$\widetilde{V}$ = valeur réelle représentée par le nombre virgule fixe
Q = valeur entière codant Vstockée en mémoire
S = facteur d'échelle ($S = F \times 2^E$)
E = position de la virgule imaginaire à partir du LSB
F = pente du facteur d'échelle
B = décalage (*offset*)

Tant les nombres signés que non signés peuvent être représentés. Dans le deuxième cas, la valeur stockée en mémoire suit la norme «complément à 2». La notation est alors $p.q$ dans le cas d'un nombre non signée et $sp.q$ dans le cas d'un nombre signé, où p est le nombre de bits représentant la partie entière, q est le nombre de bits représentant la partie fractionnaire et s est le bit de signe.

L'intervalle des valeurs pouvant être représentées dépend directement du nombre de bits disponibles pour encoder les valeurs réelles. Selon le tableau 3.1, un nombre de 16 bits avec une représentation `FPs9.6` se situe dans l'intervalle discret [-512, 511,984375]. La résolution à l'intérieur de cet intervalle est $0,015625$.

La difficulté de l'arithmétique virgule fixe ne se situe pas au niveau de la représentation, mais bien au niveau des opérations. En effet, l'utilisateur de cette arithmétique doit toujours avoir en tête la précision résultante d'une opération afin d'éviter les erreurs telles que les débordements (*overflow*) ou de *wrap-around* dues à un nombre de bits insuffisant pour contenir le résultat de l'opération effectuée. Il est facile de s'y perdre si on ne fait pas attention. Le tableau 3.2 résume les précisions résultantes des différentes opérations mathématiques de base.

L'addition de deux nombres à virgule fixe n'est possible que si les deux ont le même nombre de décimales. Puisque les valeurs stockées dans la mémoire de l'ordinateur ne sont en réalité que des nombres entiers, il revient au programmeur de s'assurer, par des décalages de bits si nécessaire, qu'il n'additionne que des valeurs présentant la virgule virtuelle au même endroit. Sans quoi, l'opération, bien que possible d'un point de vue logiciel, ne produit pas un résultat cohérent.

Le résultat de l'addition de deux nombres est de même précision mais nécessite un bit de plus dans la partie réelle pour contenir la retenue s'il y a lieu. La soustraction fonctionne selon le même principe, excepté que dans le cas de deux nombres non signés, le bit supplémentaire de la retenue ne sera jamais présent. Cependant, une erreur de *wrap-around* peut survenir lorsque le résultat de l'opération devrait être négatif. C'est-à-dire qu'un résultat négatif ne pouvant être représenté, le passage à zéro boucle à la valeur maximale. Le quantité obtenue sera donc supérieure à la quantité originale, ce qui est, bien évidemment, insensé.

Lors d'une multiplication, le résultat possède un format composé par l'addition des nombres de bits des deux opérandes pour la partie réelle et fractionnaire respectivement.

Une division s'accompagne d'une perte de précision au niveau des décimales, le

Tableau 3.2 Précision résultante des opérations à virgule fixe

Opération	Résultat
Addition	$p_1.q + p_2.q = (\max(p_1, p_2) + 1).q$ (3.2) $sp_1.q + sp_2.q = s(\max(p_1, p_2) + 1).q$ (3.3)
Soustraction	$p_1.q - p_2.q = \max(p_1, p_2).q$ (3.4) $sp_1.q - sp_2.q = s(\max(p_1, p_2) + 1).q$ (3.5)
Multiplication	$p_1.q_1 \times p_2.q_2 = (p_1 + p_2).(q_1 + q_2)$ (3.6) $sp_1.q_1 \times sp_2.q_2 = s(p_1 + p_2).(q_1 + q_2)$ (3.7)
Division	$p_1.q_1 \div p_2.q_2 = (p_1 + q_2).(q_1 - q_2)$ (3.8) $sp_1.q_1 \div sp_2.q_2 = s(p_1 + q_2).(q_1 - q_2)$ (3.9)

nombre de bits pour la partie fractionnaire du résultat se retrouve en fait à être la soustraction entre le nombre de bits du numérateur et du dénominateur. Une astuce mathématique permet cependant de contourner ce problème. Il est possible de convertir une division $\frac{p_1.q_1}{p_2.q_2}$ en multiplication en inversant le deuxième terme de façon à obtenir

$$p_1.q_1 \times \frac{1}{p_2.q_2}$$

La valeur 1 étant une constante, il est possible de la représenter avec autant de précision que nécessaire. La perte de précision inhérente à la division se trouve ainsi atténuée et le résultat total de l'opération est plus précis. Illustrons ceci à l'aide d'un exemple.

On cherche à calculer le résultat du quotient

$$\frac{18,5}{11,25} = 1,64\overline{4}$$

et comparer le résultat obtenu avec et sans l'utilisation de l'inversion intermédiaire. Nous nous limitons à des termes ne dépassant pas 16 bits, mais aucune limite n'est fixée pour les calculs intermédiaires.

Commençons par calculer les différentes représentations virgules fixes pour chacun des termes utilisés.

$$\begin{aligned} 18.5 &= (10010.1)_2 \Rightarrow \texttt{FP5.1} \\ (100101)_2 &= 37 \end{aligned} \tag{3.10}$$

$$\begin{aligned} 11.25 &= (1011.01)_2 \Rightarrow \texttt{FP4.2} \\ (101101)_2 &= 45 \end{aligned} \tag{3.11}$$

Sans l'utilisation de l'inversion intermédiaire, on obtient directement

$$\begin{aligned} \left\lfloor \frac{37}{45} \right\rfloor &= 0 \\ \texttt{FP} = (5+2).(1-2) &= \texttt{FP7.-1} \\ \Rightarrow (00)_2 &= 0 \end{aligned} \tag{3.12}$$

La valeur obtenue est complètement erronée ! La perte de précision de la division est fatale. Avec l'utilisation de l'inversion intermédiaire, on calcule d'abord l'inverse du dénominateur. Pour cela, on utilise la constante 1 avec une précision arbitraire, prenons une précision de `FP1.15`, soit 16 bits.

$$\begin{aligned} 1 \text{ en } \texttt{FP1.15} \Rightarrow \texttt{0x8000} &= 32768 \\ \left\lfloor \frac{32768}{45} \right\rfloor &= 728 = (1011011000)_2 \\ \texttt{FP} = (1+2).(15-2) &= \texttt{FP3.13} \\ \Rightarrow (000.0001011011000)_2 &= 0,0888671875 \end{aligned} \tag{3.13}$$

La vraie valeur de l'inversion est $0,0\overline{88}$. L'erreur ici est d'environ $0,2\%$. La multiplication subséquente donne alors

$$\begin{aligned} 37 \times 728 = 26936 &= \texttt{0x6938} \\ \texttt{FP} = (5+3+1).(1+13) &= \texttt{FP9.14} \\ \Rightarrow (000000001.10100100111000)_2 &= 1,64404296875 \end{aligned} \tag{3.14}$$

L'erreur sur la vraie valeur est de $0,0004014756944$, soit presque $0,025\%$. On remarque que cette astuce mathématique a permis de grandement accroître la pré-

cision finale. Dans certains cas, le gain de précision est marginal mais on ne peut pas prendre le risque de reproduire l'exemple précédent, c'est pourquoi la division est toujours codée de la sorte dans le code du DSP.

3.1.1 Logarithme naturel

Le traitement du courant tunnel nécessite le calcul d'un logarithme naturel (ou népérien). Or, cette fonction mathématique continue n'est généralement pas implémentée dans un DSP à virgule fixe. Certains compilateurs permettent son utilisation et prennent en charge le passage en calcul en arithmétique à virgule fixe. Quoiqu'il en soit, l'évaluation d'un logarithme dans un DSP à virgule fixe n'est jamais exacte. Certaines approximations sont nécessaires, mais la précision atteignable peut néanmoins s'avérer suffisante pour les besoins de la situation. Plusieurs techniques d'approximation existent. L'algorithme de Borchardt propose notamment l'équation suivante[19] :

$$\ln(x) \approx \frac{6(x-1)}{x+1+4\sqrt{x}} \tag{3.15}$$

La série de Mercator[66], qui est en fait qu'une décomposition en série de Taylor autour de 1, donne :

$$\ln(1+x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} x^k = x - \frac{x^2}{2} + \frac{x^3}{3} + ... \tag{3.16}$$

pour $-1 < x < 1$. L'utilisation de cette série mène forcément à un compromis entre la précision et la rapidité de calcul. Plus le nombre de termes utilisés est grand, plus la précision résultante sera bonne, mais plus long sera le calcul.

Le logarithme est une fonction mathématique qui croît lentement, de cours segments peuvent même être approximés par une droite. Une tactique couramment

utilisée met à profit cette observation. Elle consiste à précalculer plusieurs valeurs de logarithmes, les mémoriser dans une table de recherche (*lookup table*) et procéder à une interpolation linéaire lors du calcul d'une valeur particulière. Considérant les valeurs a et b, où $a < b$, pour lesquelles les valeurs de $\ln(a)$ et $\ln(b)$ sont connues, l'équation d'interpolation devient alors :

$$\ln(x) \approx (x - a)\left(\frac{\ln(b) - \ln(a)}{b - a}\right) + \ln(a) \tag{3.17}$$

Le nombre de valeurs mémorisées dépend de la plage possible pour les valeurs dont il faut calculer le logarithme, ainsi que du degré de précision nécessaire. Par exemple[1], si l'on désire calculer $\ln(\texttt{0x3456})$ et qu'on ne dispose que d'un tableau possédant les valeurs de $\ln(\texttt{0x1000} \times i)$ où $i = 0...15$. De l'équation 3.17, on obtient

$$\ln(\texttt{0x3456}) \approx (\texttt{0x3456} - \texttt{0x3000}) \times \left(\frac{\ln(\texttt{0x4000}) - \ln(\texttt{0x3000})}{\texttt{0x4000} - \texttt{0x3000}}\right) + \ln(\texttt{0x3000}) \tag{3.18}$$

$$= 1110 \times \frac{9,70406 - 9,41638}{4096} + 9,41638$$

$$9,50286 \approx 9,49434$$

L'intégration à l'arithmétique virgule fixe repose la propriété du logarithme stipulant que

$$\ln\left(\frac{a}{b}\right) = \ln(a) - \ln(b) \tag{3.19}$$

Ainsi, en reprenant l'exemple précédant avec une représentation `FP5.11`,

$$\texttt{0x3456} => \frac{\texttt{0x3456}}{2^{11}} = 6,65199$$

[1]Cette exemple est adapté de celui trouvé dans [19]

donne naissance au calcul suivant :

$$\ln\left(\frac{\texttt{0x3456}}{2^{11}}\right) = \ln(\texttt{0x3456}) - \ln(2^{11}) \tag{3.20}$$

$$= \ln(\texttt{0x3456}) - 11 \times \ln(2) = 1,87824 \tag{3.21}$$

La valeur de ln(2) est une constante qu'on est capable de précalculer avec autant de précision que disponible dans le DSP.

3.2 Électronique

La figure 3.2 illustre de façon schématique le contrôle d'un STM. Le courant tunnel provient du saut des électrons de valence d'un matériau conducteur chargé vers une pointe effilée de Platinum/Iridium lorsque la distance entre ceux-ci atteint l'ordre du nanomètre. Pour ce faire, une différence de potentiel (*Bias Voltage*) de quelques millivolts à une dizaine de volts doit être appliquée entre ces deux éléments. Ceci induit un courant extrêmement faible (typiquement entre 0,01 nA et 50nA [13]) circulant à travers la tige conductrice. Bien que le phénomène exact nécessite la résolution de l'équation de Schröedinger, la relation 3.22 permet d'approximer le lien entre la distance et la valeur du courant tunnel induit.[65]

$$I_{tunnel} \propto e^{-A\sqrt{\phi}d} \tag{3.22}$$

où ϕ est le travail d'extraction (*work function*) résultant des deux électrodes (pointe et matériau), A est une constante d'environ $\frac{1}{\sqrt{eV}\text{Å}}$ et d est la distance séparant la pointe de la surface.

Ce courant, fortement sensible aux diverses sources de bruit électronique présentes dans l'environnement et sur le reste du circuit *Flex* (transmission IR, bus de don-

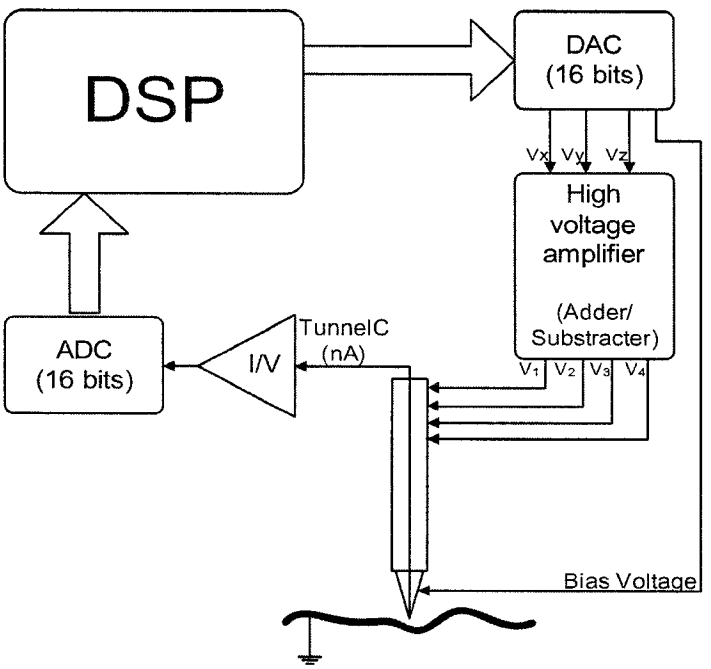
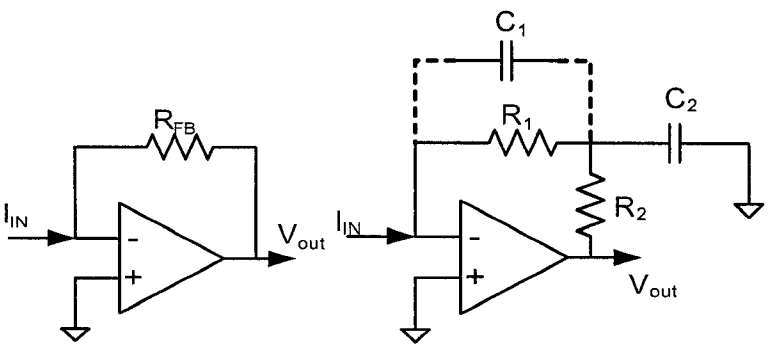


Figure 3.2 Parcours du courant tunnel dans un STM

nées, etc.), doit être rapidement converti en une quantité plus robuste, sans quoi le bruit devient vite dominant et noie le signal recherché.

La première étape est une conversion courant/tension réalisée à l'aide d'un amplificateur opérationnel branché en feedback négatif.(voir Figure 3.3(a)) En pratique cependant, la présence de capacité parasite vient réduire la réponse en fréquence de ce circuit, c'est pourquoi un réseau RC est souvent ajouté après la résistance R_{FB} de façon à obtenir une meilleure réponse fréquentielle sans pour autant ajouter de bruit thermique (aussi appelé bruit de Johnson).[13] (voir Figure 3.3(b)) L'équation 3.23 présente la valeur de tension obtenue en fonction du courant tunnel en entrée.

$$V_{out} = I_{tunnel}\left(R_1 + R_2\right)\frac{1 + i\omega R_2 C_2}{1 + i\omega R_1 C_1} \tag{3.23}$$



(a) Idéal (b) Large bande passante

Figure 3.3 Amplificateur opérationnel en feedback négatif

Un facteur de gain de 10^8 est typiquement utilisé dans les STM commerciaux, le NanoWalker ne fait pas exception. Le courant tunnel est ainsi converti en une tension de l'ordre du Volt.

La tension obtenue est acheminée à un convertisseur analogique-numérique (ADC) de 16 bits dont la plage dynamique de conversion est $-10V$ à $10V$. Ceci offre un pas de quantification avoisinant $300\mu V$ par division. L'équation 3.24 présente la relation entre la valeur du courant tunnel et la tension correspondante reçue par le DSP.

$$I_{tunnel} = V_{ADC} \times \frac{V_{max} - V_{min}}{2^{16}} \times \frac{1}{10^8} \tag{3.24}$$

$$= V_{ADC} \times \frac{20V}{65536 \times 10^8 \frac{V}{A}} \tag{3.25}$$

On sait que la valeur du courant tunnel varie de façon exponentielle à mesure que la distance entre la pointe et la surface diminue. Une relation linéaire entre les deux est déduite en calculant le logarithme naturel des deux expressions. L'équation 3.26 présente le résultat de l'opération.

$$\ln\left(I_{tunnel}\right) = \ln\left(V_{ADC} \times \frac{20}{2^{16} \times 10^8}\right) \tag{3.26}$$

$$= \ln\left(V_{ADC}\right) + \ln\left(\frac{20}{2^{16} \times 10^8}\right) \tag{3.27}$$

$$= \ln\left(V_{ADC}\right) - 26,515303359... \tag{3.28}$$

En étudiant la valeur du courant tunnel, il est possible d'obtenir une image réaliste de la densité d'électrons d'une surface.[49]. De là, de l'information sur la topographie de la surface imagée peut être tirée. En effet, le DSP implémente un contrôleur PID qui s'affaire à maintenir un courant tunnel constant. Si ce dernier augmente, le DSP sait que la pointe s'est rapprochée de la surface et réagit en rétractant le tube piézoélectrique la supportant. Inversement, si le courant diminue, le DSP commande l'allongement du piézo. Le DSP contrôle également les déplacements de la pointe dans le plan du matériau en agissant sur ce même piézo. En balayant la surface, le courant tunnel varie selon la topographie, les variations d'élongation appliquées au piézo sont sauvegardées et, par la suite, transférées au système de contrôle qui reconstitue une image de la surface. Il faut noter que l'image obtenue est celle de la réponse du PID aux variations du courant tunnel, une bonne calibration du contrôleur est donc une condition essentielle à l'obtention d'une image fidèle à la réalité.

Les commandes acheminées au piézo prennent la forme de trois valeurs de tensions sur 16 bits (V_x, V_y et V_z) transmises à un convertisseur numérique-analogique (DAC) qui les transforment en valeurs entre $-2,5V$ et $2,5V$. Elles traversent ensuite un étage d'additions/soustractions où elles sont également amplifiées et donnent lieu à quatre tensions appliquées sur chacune des quatre électrodes du piézo (voir Figure

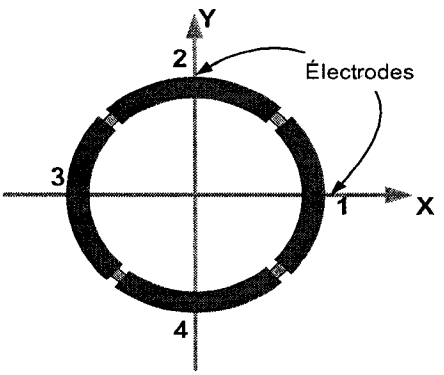


Figure 3.4 Disposition des électrodes sur le tube piézoélectrique du STM

3.4). Les équations 3.29 régissent cette dernière étape.

$$V_1 = -(V_x + V_z) \times G \tag{3.29a}$$

$$V_2 = -(V_y + V_z) \times G \tag{3.29b}$$

$$V_3 = +(V_x - V_z) \times G \tag{3.29c}$$

$$V_4 = +(V_y - V_z) \times G \tag{3.29d}$$

où G est le gain associé à l'étage d'amplification. Celui-ci peut prendre deux valeurs dans l'implémentation actuelle du NanoWalker : 56 lors du balayage d'une grande surface ($\pm 2\mu m$) ou $2,5$ pour une petite surface ($< 150nm$).

La tension maximale pouvant être sortie par l'étage d'amplification est de $138V$. Dans certaines situations, les valeurs de V_1, V_2, V_3 et V_4 se verront donc écrêtées par les caractéristiques physiques du circuit. L'impact concret est qu'à mesure que le piézo se déplace sur un axe, sa liberté de mouvements dans les deux autres axes se voit réduite.

La déformation réelle du piézo dépend des tensions appliquées sur ses électrodes. Les spécifications du manufacturier du tube piézoélectrique utilisé (un PZT-5A de

Staveley Sensors) stipule que la contribution de chaque électrode en XY est calculée par l'équation 3.30

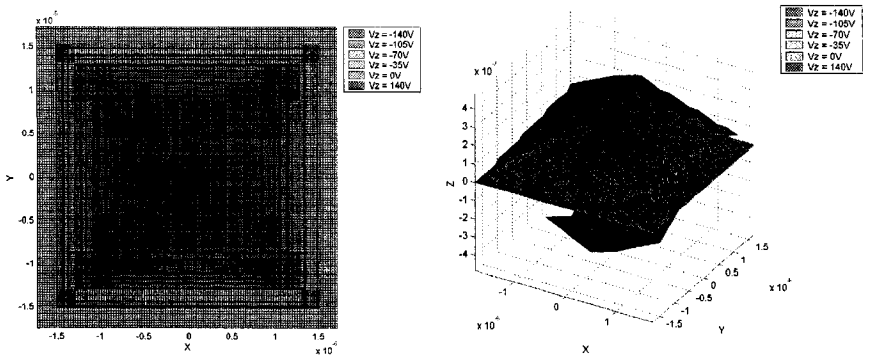
$$\Delta = -0,45 \times \frac{17,3\frac{\mu V}{m} \times (5,08mm)^2}{1,27mm \times 254\mu m} \tag{3.30}$$

De cette équation, la position en X et Y du piézo revient à :

$$X = \Delta \times (V_1 - V_3)$$
$$Y = \Delta \times (V_1 - V_4)$$

Avec Thomas Boitani FitzGerald et Marc-Antoine Fortin, des simulations `MATLab` ont permis de calculer que les déformations élastiques maximum du tube piézoélectrique sont de $\pm 1,7438\mu m$ dans les axes X et Y et de $\pm 484,40nm$ dans l'axe Z. Par contre, à cause des contraintes sur les valeurs maximales pouvant être appliquées sur les électrodes, on se retrouve dans une situation où, lorsque V_x et V_y sont maximisées, il n'y a plus aucune marge de manoeuvre sur V_z. Les surfaces présentées à la figure 3.5 illustrent la plage dynamique du piézo et la relation avec les tensions V_x, V_y et V_z applicables. On y voit que la déflexion maximale n'est atteignable que lorsque $V_z = 0$. Dès qu'une tension V_z est appliquée, ceci à pour effet d'écrêter la valeur maximal applicable à deux des quatre électrodes (une par axe) et réduire la portée horizontale.

Si l'on désire garantir à tout moment un débattement en Z de $\pm 100nm$, on doit assurer une marge de manoeuvre sur V_z de $35V$ au minimum. Les tensions V_x et V_y doivent donc être limitées à $\pm 105V$ pour éviter toute possibilité d'écrêtage.



(a) Vue du dessus (b) Vue en plongée

Figure 3.5 Plage dynamique du tube piézoélectrique du STM selon différentes valeurs de V_z

3.3 Design logiciel du DSP

D'un point de vue logiciel, l'interaction avec les composantes du circuit électronique se fait par l'entremise du DAC et de l'ADC. Le DAC7644 est un convertisseur numérique-analogique 16-bits à quatre sorties à bascules contrôlées par un signal nommé `LOADDACS`. L'écriture doit respecter le chronogramme de la figure 3.6. Puisque le temps de stabilisation de la sortie est de $10\mu s$, sa fréquence maximale d'utilisation est de $100kHz$. La linéarité est garantie sur 15 bits.

Le convertisseur analogique-numérique ADC976A reçoit en entrée une tension entre $-10V$ et $+10V$ et la convertit sur 16 bits. La conversion et la lecture de la valeur doit respecter le chronogramme de la figure 3.7. Une conversion nécessite $4\mu s$ environ, sa fréquence maximale d'utilisation est de $200kHz$.

Le tableau 3.3 résume les signaux nécessaires au contrôle du STM par le DSP. Les bus de données et d'adresses sont partagés entre le DAC, l'ADC et la mémoire de stockage. Ceux-ci ne peuvent donc pas être utilisés simultanément.

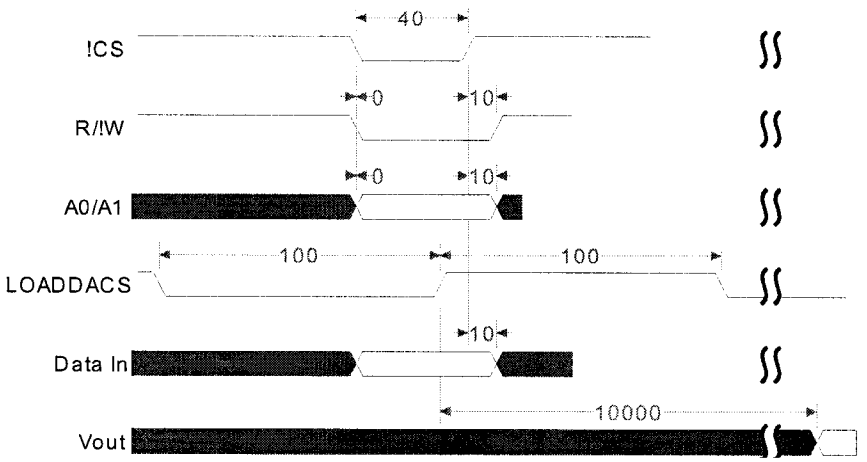


Figure 3.6 Chronogramme d'une écriture au DAC (temps en ns)

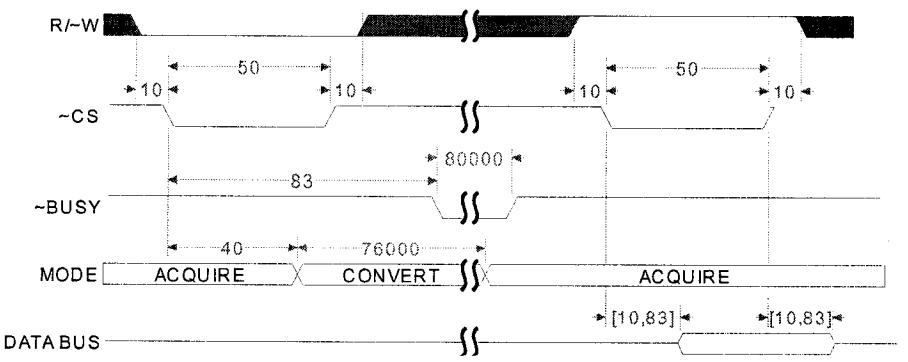


Figure 3.7 Chronogramme d'une lecture de l'ADC (temps en ns)

Tableau 3.3 Signaux de contrôle du circuit STM

Nom	Taille	Dir.	Description
DATA	16 bits	I/O	Bus de données partagé entre le DAC et l'ADC.
DACADDR	2 bits	O	Adressage d'un des quatre registres de conversion du DAC.
DACSel	1 bit	O	*Chip Select* (CS) du DAC. Actif bas.
DACRST	1 bit	O	*Reset* du DAC. Initialise les valeurs de sorties du DAC à une valeur prédéterminée par DACRSTSEL. Actif sur un front montant.
DACRSTSEL	1 bit	O	Sélectionne la valeur de sortie du DAC lors du prochain *reset*. 1 => 0x8000 0 => 0x0000
LOADDACS	1 bit	O	Contrôle la sortie des valeurs converties par le DAC. Sur réception de ce signal, les tensions analogiques deviennent «actives». Actif sur un front montant.
ADC_CS	1 bit	O	*Chip Select* (CS) de l'ADC. Actif bas.
RW	1 bit	O	*Read/Write*. Sélectionne le mode (lecture ou écriture) pour le DAC ou l'ADC. 1 => Lecture (*Read*) 0 => Écriture (*Write*)
ScanGain	1 bit	O	Sélectionne le gain d'amplification de l'étage d'additions/soustractions. 1 => *Small scan* 0 => *Big scan*

Tableau 3.4 Description des tâches du DSP nécessaires au contrôle du STM

Tâche	Description
PiezoScan	Gère le déplacement horizontal (en X-Y) du tube piézoélectrique supportant la pointe STM. Aucun retour d'information sur la position réelle du piézo n'est récolté. Une calibration précise du piézo est nécessaire pour déterminer sa déflexion maximum et son comportement.
Z-Feedback	Gère le déplacement vertical (en Z) du piézo pour répondre aux variations dans le courant tunnel afin d'épouser la topographie du matériau imagé. Cette boucle de contrôle fermée est implémentée à l'aide d'un PID dans le DSP.
Communication IR	Gère la réception des instructions du système de contrôle et le transfert de l'information sur le courant tunnel sauvegardée. Cette partie n'est pas spécifique à l'opération du microscope et est partagée avec l'ensemble des tâches du NanoWalker.

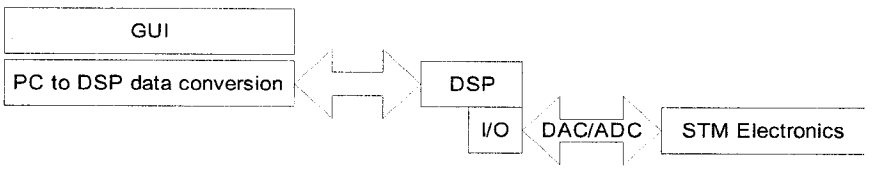


Figure 3.8 Décomposition du prototype logiciel pour le contrôle du STM

Une analyse de surface par le STM peut se diviser en trois grandes parties : le balayage du tube piézoélectrique (PiezoScan), la boucle de contrôle sur le courant tunnel menant à une rétroaction sur le piézo (Z-Feedback) et la sauvegarde et communication des données (via l'infrarouge). Le tableau 3.4 résume ces trois tâches parallèles. Chacune de ces parties est d'abord élaborée et développée d'une façon indépendante et, par la suite, intégrée les unes aux autres. Le prototype logiciel est en deux parties : l'interface utilisateur sur PC et la partie destinée au DSP. La figure 3.8 schématise cette décomposition. En plus de l'interface, le côté PC comprend le traitement des données avant l'envoi au DSP.

Un DSP n'est pas un environnement «multiprocessus», c'est-à-dire que les tâches qu'il exécute sont traitées séquentiellement et qu'il n'y a aucune préemption possible par une tâche plus prioritaire. Les mécanismes d'interruptions contournent cependant cette limitation. Le TMS320C25-50 possède un total de sept interruptions. Trois interruptions sont dédiés aux périphériques externes et quatre sont internes : deux pour l'interface de communication série, une pour l'unique *timer* et la dernière pour l'instruction logicielle `TRAP`.[63] Il est également possible qu'une interruption interrompe une autre interruption moins prioritaire, mais ceci ne devrait pas être confondu avec de la préemption. Le traitement d'une interruption devrait être aussi rapide que possible, le nombre d'opérations qu'elle devrait exécuter est donc limité. Typiquement, le traitement d'une sous-routine d'interruption (ISR) se limite à activer un indicateur (*flag*) sur lequel réagira le programme principal. Ainsi, un programme de DSP a généralement l'allure d'une boucle principale infinie qui appelle différentes fonctions selon la valeur de registres d'état modifiés par des événements internes ou externes.

L'ordonnancement et la décomposition des tâches est primordiale. Il ne faut pas que l'exécution d'une longue tâche vienne bloquer une courte tâche plus critique. Chaque tâche est donc fragmentée en plusieurs sections de code relativement courtes. Entre chaque fragment, le retour à la boucle principale vérifie si une autre tâche plus prioritaire doit venir interrompre la continuité de la première. Il fut décidé que la tâche la plus prioritaire serait la communication infrarouge car si l'on perd la possibilité de communiquer avec le robot, le résultat de toute autre tâche que celui-ci exécuterait serait également perdu. Ensuite vient le Z-Feedback et finalement, le PiezoScan.

3.4 Stratégie de tests

L'électronique du NanoWalker est un héritage signé MIT. Une fois le projet transféré à Polytechnique, un prototype du circuit *Flex* était déjà en cours de production. Des retards dans sa fabrication, ainsi que certaines erreurs décelées dans le design original, nous ont cependant poussé à envisager d'autres approches pour le développement et la validation des modules électroniques et logiciels du STM. En collaboration avec Marc-Antoine Fortin, étudiant à la maîtrise au Laboratoire, un prototype pour chacune des parties électroniques du STM fut fabriqué à l'aide des machines de prototypage rapide de circuits de la compagnie LPKF.

Ne disposant pas du premier prototype du circuit (une version grande-échelle que nous avons baptisé «MacroWalker») réalisé au MIT lors de la première itération de design, il fut décidé de d'abord tester la fonctionnalité à l'aide d'une carte d'entrées/sorties numériques contrôlée par un ordinateur. Plusieurs modèles de ces cartes existent actuellement sur le marché, National Instruments étant sûrement parmi les plus gros fabricants. Les vitesses atteignables par les plus performantes sont de l'ordre des MHz. Par contre, ceci implique de fonctionner en mode *Pattern I/O*, c'est-à-dire qu'un certain patron de bits est préalablement stocké dans une mémoire interne de la carte et, lorsque le signal de départ est donné, la carte boucle sur ce patron à une fréquence élevée. Aucune modification au patron n'est possible sans arrêter la sortie et charger un nouveau patron en mémoire. Dans un tel mode, il est impossible d'implémenter une boucle de contrôle où la valeur de la sortie est fonction d'une valeur lue en entrée. Dans le mode *Single line*, chaque bit de sortie est contrôlé directement par un appel de fonction du pilote de la carte. Ceci permet une plus grande flexibilité, mais c'est au détriment de la fréquence. En effet, la fréquence maximale atteignable dans ce mode diminue d'une façon drastique à cause des accès répétés au bus PCI et au processeur.

Vue la nature du problème, le mode *Single line* doit être utilisé. La carte d'entrées/sorties 8505 de la compagnie Sealevel ne possède que ce mode, ce qui en fait une carte très bon marché, quoique plutôt lente. Des tests maisons effectués sur un Pentium IV 1,7 GHz ont trouvés une période minimale d'environ $30\mu s$. Avec six groupes de huit bits, pour un total de 48 bits, pouvant être contrôlés individuellement, elle possède assez de lignes pour contrôler le circuit STM.

Conscient que l'utilisation d'une carte I/O n'est pas une solution viable à long terme, une attention particulière fut portée à la portabilité du code. Le code spécifique au matériel utilisé est encapsulé dans un ensemble de fonctions facilement modifiables. Le reste du code est ainsi plus générique et réutilisable. Un *thread* indépendant, baptisé *EmuDSP*, «émule» les fonctionnalités du DSP.

Ne disposant pas d'un STM commercial avec lequel valider notre propre STM, nous nous sommes rendus au département de Physique de l'université McGill. En consultant le professeur Peter Grütter, une stratégie de tests en trois temps fut adoptée. L'assemblage mécanique réalisé par Thomas Boitani FitzGerald doit d'abord être validé. Pour ce faire, on utilise un vieux STM de la compagnie NanoScope. La tête et l'électronique de ce STM sont d'abord utilisés avec notre base. La figure 3.9 illustre le montage. En plaçant l'échantillon sur notre piézo et en capturant une image complète avec leur logiciel et électronique, on s'assure que l'échantillon est bel et bien en contact avec notre base de sorte que la tension (*Bias Voltage*) commandée par leur contrôleur est bien appliquée. En utilisant leur piézo pour balayer la surface et lire les valeurs du courant tunnel, on ne peut obtenir une image que si l'assemblage mécanique est fonctionnel, sans pour autant faire appel à notre électronique de contrôle.

Dans un deuxième temps, avec le même montage mécanique, on maintient leur piézo immobile en XY et on utilise le nôtre pour effectuer le balayage. On utilise tou-

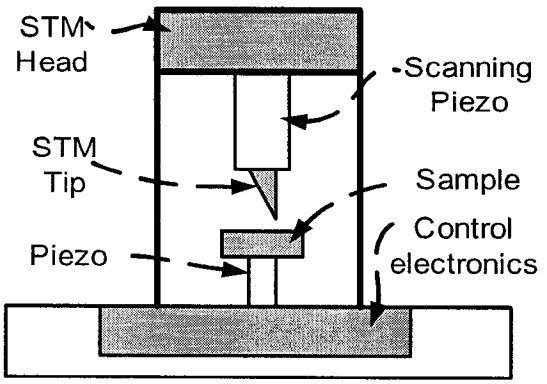


Figure 3.9 Montage du test d'assemblage mécanique du STM

jours leur électronique pour la lecture du courant tunnel et le Z-Feedback, mais on déplace l'échantillon par rapport à la pointe plutôt que l'inverse. Ceci ne constitue cependant pas un réel problème puisque plusieurs STM commerciaux fonctionnent de cette façon. L'image obtenue par un tel test n'a aucune signification réelle, autre que de nous apprendre que notre logique de balayage est bonne et que l'assemblage des microfils sur les électrodes du tube piézoélectrique est fonctionnel.

Comme dernier test, on inverse le tout premier. C'est-à-dire qu'on inverse la position de l'échantillon pour le fixer à leur piézo à la place de la pointe STM et on fixe une pointe à notre piézo. On utilise notre propre logique de balayage et de lecture du courant tunnel pour capturer une image. En utilisant le même type d'échantillon à travers tous les tests (typiquement de l'or sur du mica), on peut comparer les images obtenues entre elles pour le premier et dernier test. Le résultat, quoique différent, devrait présenter de grandes similarités.

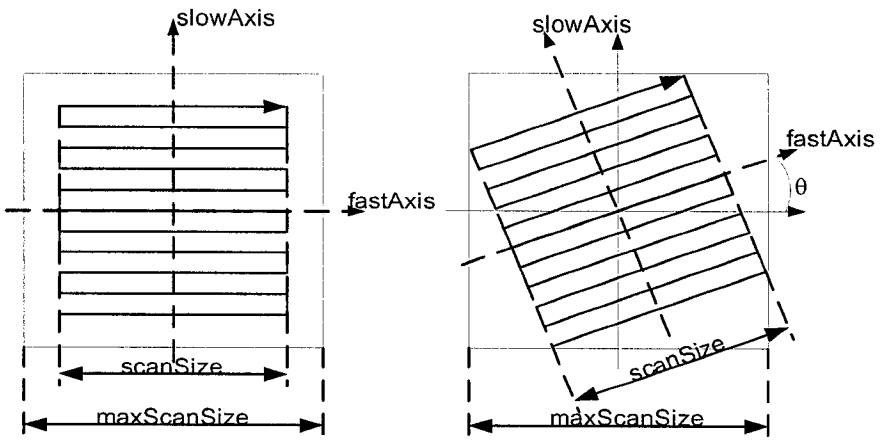
3.5 PiezoScan

Le type de balayage du tube piézoélectrique utilisé est couramment appelé «balayage TV», ou *raster scan* en anglais, c'est-à-dire que le balayage s'effectue selon deux axes : un axe rapide (*fastAxis*) et un axe lent (*slowAxis*). La position sur l'axe rapide est mise à jour à la fréquence de balayage tandis que la position sur l'axe lent n'est modifiée qu'à la fin d'une ligne de l'axe rapide. La figure 3.10(a) illustre ce type de balayage dans sa forme la plus simple. D'une façon plus générale, la direction principale du scan peut faire un angle θ avec l'orientation XY du piézo comme à la figure 3.10(b). La zone scannée est habituellement carrée. Le nombre de lignes sur l'axe lent est égal à la résolution désirée en pixels pour l'image résultante. Les valeurs de 128, 256 ou 512 sont courantes dans l'industrie. Sur l'axe rapide, la moyenne de plusieurs lectures est utilisée pour chaque pixel. Ce nombre dépend de la fréquence de balayage, mais avoisine habituellement 40[52]. Afin d'obtenir une plus grande précision sur les données, il est commun d'effectuer un aller-retour sur une même ligne avant de poursuivre le long de l'axe lent. Ce mode porte le nom de mode *Retrace*, par opposition au mode *Trace* où chaque ligne n'est parcourue qu'une seule fois. Le mode *Retrace* permet de comparer la topographie à l'aller et au retour d'une même ligne. Elles devraient théoriquement être identiques mais l'hystérésis et le fluage (*creep*) du piézo, sans compter le bruit dans le système, peuvent altérer les résultats.

Le tableau 3.5 présente une liste des différents paramètres régissant un scan du STM. En fonction des paramètres fournis tels que la taille de la surface à imager, l'angle et la fréquence du scan ainsi que le décalage du centre de l'image, une décomposition en incréments sur l'axe des X et Y de l'axe rapide et lent doit être calculée. Les équations 3.31 et 3.32 présentent le calcul des incréments en mode *Retrace*.

Tableau 3.5 Paramètres nécessaires au balayage du piézo

Type	Nom	Description
Fournis	`scanSize`	Taille de la surface imagée.
	`scanFreq`	Fréquence du scan.
	`scanAngle`	Angle de l'axe rapide par rapport à l'horizontale. (considéré par rapport aux électrodes 1 et 3 du piézo)
	`XOffset`	Décalage du centre du scan selon l'axe des X.
	`YOffset`	Décalage du centre du scan selon l'axe des Y.
	`scanRes`	Résolution en pixels de l'image de la surface. Correspond directement au nombre d'incréments sur l'axe lent.
	`scanGain`	Gain à appliquer aux tensions sortant du DAC. Gère la taille maximale du scan.
Calculés	`FastXIncr`	Composante sur l'axe des X du piézo de l'incrément à appliquer à l'axe rapide.
	`FastYIncr`	Composante sur l'axe des Y du piézo de l'incrément à appliquer à l'axe rapide.
	`SlowXIncr`	Composante sur l'axe des X du piézo de l'incrément à appliquer à l'axe lent.
	`SlowYIncr`	Composante sur l'axe des Y du piézo de l'incrément à appliquer à l'axe lent.
	`NbFastAxisUpdates`	Nombre de fois qu'il faut ajouter l'incrément rapide pour atteindre la taille du scan spécifiée.



(a) Balayage TV (b) Balayage avec un angle θ

Figure 3.10 Balayage X-Y du piézo du STM

$$fastAxisIncr = \frac{2 \times scanSize \times scanFreq}{\text{Fréquence du DAC}} \tag{3.31a}$$

$$fastXIncr = fastAxisIncr \times \cos(\theta) \tag{3.31b}$$

$$fastYIncr = fastAxisIncr \times \sin(\theta) \tag{3.31c}$$

$$nbFastAxisUpdates = \frac{scanSize}{fastAxisIncr} \tag{3.31d}$$

$$slowAxisIncr = \frac{scanSize}{scanRes} \tag{3.32a}$$

$$slowXIncr = slowAxisIncr \times \cos(\theta) \tag{3.32b}$$

$$slowYIncr = slowAxisIncr \times \sin(\theta) \tag{3.32c}$$

Ces valeurs sont de natures réelles (points flottants), elles ne peuvent donc pas être mémorisées telles quelles par le DSP. Mais puisque ces valeurs sont constantes à

l'intérieur d'un même scan, il est possible de les précalculer dans l'ordinateur de contrôle et de n'envoyer que le résultat des calculs, déjà converti en virgule fixe, au robot. Ceci permet une plus grande précision de calcul et libère de précieux cycles de traitement du DSP embarqué. Mieux encore, il ne sert à rien de mémoriser la mesure de l'incrément puisque celle-ci devra être convertie entre `0x0000` et `0xFFFF` pour interfacer avec le DAC. On peut donc immédiatement convertir les incréments en valeurs à virgule fixe sur 16 bits du DAC avec l'équation 3.33. Il faut noter qu'une tension de $0V$ équivaut à une sortie `0x8000` du DAC, ce qui correspondant à la position de repos du piézo. Lors du calcul d'une position par rapport à cette position de repos, il ne faut pas oublier d'ajouter le décalage de `0x8000` à la valeur obtenue par l'équation 3.33.

$$\begin{aligned} V_{DSP} &= \left\lfloor \frac{d_{PC}}{\text{Déflexion maximale du piézo}} \times \frac{\text{Sortie maximale du DAC}}{\text{Résolution du DAC}} \right\rfloor \\ &= \left\lfloor \frac{d_{PC}}{1,7438\mu m} \times \frac{2,5V}{\frac{2,5V-(-2,5V)}{2^{16} div.}} \right\rfloor \end{aligned} \tag{3.33}$$

Il se peut que l'une des valeurs d'incréments obtenue pour V_x et V_y soit égale à zéro à cause du manque de précision avec 16 bits. Ceci se produit généralement lorsqu'un angle de scan très proche d'un axe principal est demandé. Pour régler le problème, il suffit de garder un plus grand nombre de décimales binaires en multipliant successivement par deux la valeur réelle jusqu'à l'obtention d'une partie entière non nulle et de mémoriser le nombre de décimales gardées. Chaque décimale de plus correspond à une mise à jour qu'il faudra sauter avant d'incrémenter la position. Ce nombre de sauts de périodes est mémorisée dans une variable nommée `NbFastVxUpdatePeriod` (idem pour V_y). Un phénomène de crénelage (*aliasing*) apparaît alors comme à la figure 3.11. La trajectoire voulue a une forme en dents de scie puisque la fréquence de mise à jour sur les deux axes n'est pas la même. À cause de la non linéarité du LSB du DAC, la plus petite valeur d'incrément non

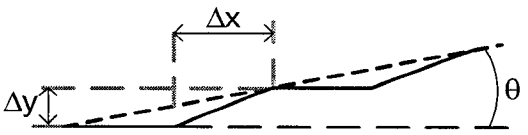


Figure 3.11 Phénomène de aliasing

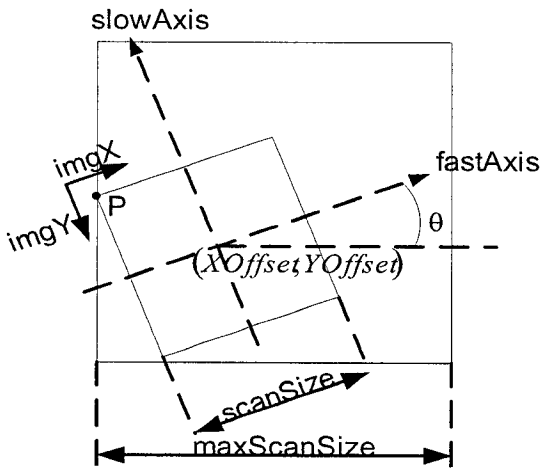


Figure 3.12 Coordonnées du début d'un scan STM

nul acceptable devrait être de deux.

Il est facile de se perdre à travers les multiples systèmes de coordonnées utilisés. L'origine d'une image (le point P sur la figure 3.12) est son coin supérieur-gauche et correspond également à la position du début du scan STM. Ses coordonnées sont :

$$\begin{pmatrix} XOffset - \dfrac{nbFastAxisUpdate}{2 \times NbFastVxUpdatePeriod} \times FastVxIncr - \dfrac{scanRes}{SlowVxIncr}, \\ YOffset - \dfrac{nbFastAxisUpdate}{2 \times NbFastVyUpdatePeriod} \times FastVyIncr - \dfrac{scanRes}{SlowVyIncr} \end{pmatrix}$$

Une fois les valeurs transférées au DSP, l'algorithme de balayage est assez direct. L'important est de maintenir un balayage aussi fluide que possible. Pour se faire, on privilégie les mises à jour petites et rapides du DAC. Un *timer* déclenche pé-

riodiquement une routine d'interruption se chargeant d'activer le signal LOADDACS. Sa période est fixée par la fréquence d'utilisation du DAC, soit à chaque $10\mu s$. Le calcul et l'écriture des nouvelles valeurs de V_x et V_y doivent donc être terminés avant la prochaine interruption du *timer*. Il est également important de noter qu'on ne doit écrire une valeur dans le DAC que si elle doit être «activée» au prochain LOADDACS puisque celui-ci est commun au quatre sorties du DAC7644.

L'algorithme de balayage continu en mode *Retrace* est le suivant :

```
Si un changement de ligne sur l'axe lent est signalé
  Si la direction verticale est présentement vers le bas
    Ajouter les incréments lents à Vx et Vy
    Incrémenter le compte des mises à jour sur l'axe lent
    Si le compte est égal à la résolution de l'image à prendre
      Changer de direction vers le haut.
  Sinon
    Soustraire les incréments lents à Vx et Vy
    Décrémenter le compte des mises à jour sur l'axe lent
    Si le compte est égal à zéro
      Changer de direction vers le bas.
  Écrire les valeurs Vx et Vy sur le DAC
Sinon
  Incrémenter le compte de période sautées depuis la dernière mise
  à jour pour Vx et Vy
  Si dans la direction «trace»
    Si le nombre de périodes à sauter == nombre de périodes sautées
    pour Vx
      Ajouter l'incrément rapide à Vx
      Réinitialiser le compte de périodes sautées à zéro.
      Écrire Vx sur le DAC
    Si le nombre de périodes à sauter == nombre de périodes sautées
    pour Vy
      Ajouter l'incrément rapide à Vy
      Réinitialiser le compte de périodes sautées à zéro.
      Écrire Vy sur le DAC
    Si Vx ou Vy a été modifié
      Incrémenter le compte de mises à jour sur l'axe rapide
```

```
    Si la ligne est complétée
      Changer de direction vers la direction «retrace»
Sinon
  Si le nombre de périodes à sauter == nombre de périodes sautées
  pour Vx
    Soustraire l'incrément rapide à Vx
    Réinitialiser le compte de périodes sautées à zéro.
    Écrire Vx sur le DAC
  Si le nombre de périodes à sauter == nombre de périodes sautées
  pour Vy
    Soustraire l'incrément rapide à Vy
    Réinitialiser le compte de périodes sautées à zéro.
    Écrire Vy sur le DAC
  Si Vx ou Vy a été modifié
    Décrémenter le compte de mises à jour sur l'axe rapide
    Si le compte des mises à jour sur l'axe rapide == zéro
      Changer de direction vers «trace»
      Si le changement de ligne sur l'axe lent est activé
        Signaler un changement de ligne sur l'axe lent.
```

3.6 Z-Feedback

Le Z-Feedback du microscope se décompose lui-même en deux : l'approche initiale et le contrôle PID. Tout microscope à effet tunnel doit dans un premier temps s'approcher de l'échantillon à observer et établir un courant tunnel avant de commencer à scanner. L'approche initiale ou *coarse approach* en anglais a comme objectif de trouver ce premier courant prêt de la référence du PID en approchant graduellement la pointe STM vers la surface du matériau. Une fois un courant tunnel adéquat observé, le DSP bascule en mode Z-Feedback proprement dit et le PID régularise la distance de sorte à garder le courant tunnel constant.

Lorsque le NanoWalker est en mouvement, la pointe du STM est relevée à une distance sécuritaire de plusieurs dizaines de microns. Vue la faible portée du tube piézoélectrique supportant la pointe, un deuxième système d'approche est néces-

saire pour combler cette distance initiale. Le design initial prévu par le MIT avait la forme d'un *inchworm actuator*, un système de pinces capable de descendre ou monter le piézo. Par contre, ce design fut par la suite éliminé à cause d'une trop grande complexité mécanique. À ce jour, le système de remplacement n'est pas encore connu. L'algorithme d'approche peut néanmoins être élaboré.

```
Centrer piézo en X-Y (Appliquer Vx=Vy=0x8000)
Appliquer Bias Voltage
Centrer piézo en Z (Appliquer Vz=0x8000)
Lire courant tunnel
Tant que Coarse Approach n'est pas au
max de son allongement ET mode == Tip Engage
    Centrer piézo en Z
    Allonger Coarse Approach d'un cran
    Lire courant tunnel
    Tant que tunnelC == 0 OU piézo pas rendu au max de l'allongement
        Incrémenter Vz d'un cran
        Lire courant tunnel
        Si tunnelC != 0
            Si allongement du piézo < seuil d'élongation tolérée
                Passer en mode ZFeedback
```

Le seuil d'élongation tolérée est arbitraire. Il sert à garantir que le piézo ne soit pas complètement allongé avant de passer en mode de Z-Feedback où il ne pourrait alors aucunement compenser pour les variations topographiques. Soixante-dix pourcents semble une valeur acceptable pour la première implémentation.

L'équation du contrôleur PID telle que fournie par Thomas Boitani FitzGerald[11] a la forme suivante :

$$\begin{aligned} V_{z_{i+1}} = V_{z_i} &+ err_i\,(K_p + K_i T) + err_{i-1}\,(-K_p) \\ &+ \ln\left(I_{tunnel_i}\right)\left(\frac{-K_d}{T}\right) \\ &+ \ln\left(I_{tunnel_{i-1}}\right)\left(\frac{2K_d}{T}\right) \\ &+ \ln\left(I_{tunnel_{i-2}}\right)\left(\frac{-K_d}{T}\right) \end{aligned} \tag{3.34}$$

où l'erreur $err_i = \ln(I_{tunnel_i}) - ln(I_{setPoint})$, K_p, K_i et K_d sont fixés par l'utilisateur et T est la période d'échantillonnage propre au système. Les termes entre parenthèses sont entièrement déterminés par des valeurs connues avant d'envoyer au robot la commande de scanner. Il est donc possible de procéder de la même façon que pour les paramètres du PiezoScan, c'est-à-dire de tout précalculer dans le PC et de convertir le résultat en virgule fixe pour utilisation directe par le DSP. La précision finale doit être celle de V_z à cause des propriétés d'addition de termes à virgule fixe.

3.7 Résultats

La première série de tests réalisée à l'Université McGill au courant du printemps 2003 a permis d'établir que le montage STM reliait correctement l'échantillon à la base de sorte que le *Bias Voltage* pouvait être appliqué. Par contre, de graves problèmes de bruit électroniques, vraisemblablement dus à des boucles de mise à la terre (*ground loops*) sont venus mettre un terme aux deuxième et troisième étapes de tests.

Ces problèmes de boucles surviennent lorsque les trois sources d'alimentation nécessaires sont branchées simultanément. En effet, trois sources d'alimentation dis-

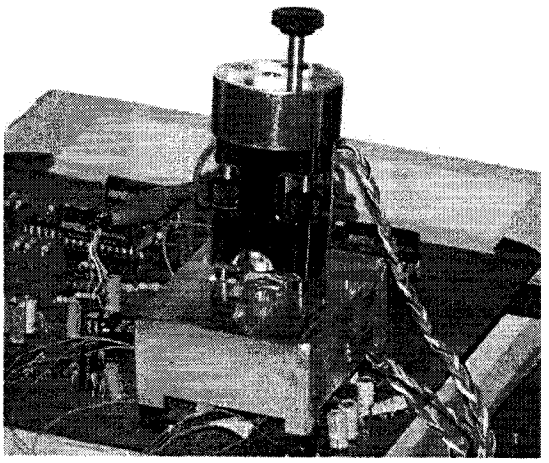


Figure 3.13 Montage utilisé lors des tests STM à McGill

tinctes étaient utilisées : le contrôleur NanoScope, l'ordinateur avec la carte I/O et le bloc d'alimention des DC-DC du circuit STM. Le fort bruit électronique engendré rend impossible l'obtention d'un courant tunnel. En forçant des tensions manuellement sur les électrodes du piézo, sans passer par la carte I/O, un mouvement en Z a pu être obtenu. Mais, il ne fut pas possible de valider de façon expérimentale l'ensemble du STM. Les tests de l'algorithme de balayage ont donc dû se limiter à des tests avec un analyseur logique.

Comme prévu, la carte d'entrées/sorties s'est révélée être beaucoup trop lente pour contrôler adéquatement le STM. Des délais non négligeables entre la sortie des différents groupes de huit bits viennent même introduire des *glitchs* sur les lignes de données. Ces délais peuvent atteindre $3\mu s$, ce qui est bien au-dessus du temps de maintien (*hold time*) de $100ns$ du DAC et de l'ADC. Par exemple, lors d'un changement de ligne du balayage, l'adressage du DAC survient $3\mu s$ après le changement de données sur le bus et l'adresse demeure $3\mu s$ de plus à la fin, causant ainsi l'affectation d'une valeur erronée au DAC.

Pis encore, on remarque qu'à chaque $15,6ms$ environ, un délai plus long qu'à l'ha-

bitude (jusqu'à plusieurs centaines de millisecondes) se produit. Cette périodicité semble due au quantum de temps que Windows alloue au processus avant de forcer l'appel de l'ordonnanceur. L'augmentation de la priorité du processus et du *thread* du EmuDSP ne change en rien ce délai. Il est donc totalement impossible de penser avoir un bon contrôle sur le STM avec une telle carte d'entrée/sorties.

3.8 eZdsp

Le Laboratoire s'est donc doté d'une carte de développement pour DSP : un module eZdspF2812 de la compagnie Spectrum Digital. Ce DSP, le F2812 de TI, bien plus récent que le TMS320C25-50 possède certaines similarités avec lui, notamment le fait d'être 16-bits sans instruction à virgule flottante. La carte de développement l'opère à une fréquence de $30MHz$, bien qu'il puisse fonctionner jusqu'à $150Mhz$. Il possède un port d'interface externe respectant la temporisation ISA, ce qui en fait une interface parfaite pour communiquer avec le TIR2000 qui fonctionne également selon cette norme. Divers paramètres permettent de configurer le nombre de cycles de délai entre les différents signaux de contrôle ISA : $\overline{CS}$, $R/\overline{W}$, l'adresse et les données. Ce port ne peut cependant pas être utilisé pour adresser le DAC et l'ADC. La figure 3.15 montre la différence entre les signaux d'un bus ISA et les signaux du DAC et de l'ADC. L'arrivée du $\overline{CS}$ par rapport à $R/\overline{W}$ y est inversée. Dans le circuit du NanoWalker, le CPLD a la responsabilité de convertir les signaux d'adressage du DSP en un format respectant les chronogrammes de chacune des composantes. Ne disposant pas d'un tel mécanisme, des ports GPIO sont donc utilisés pour interfacer avec le DAC et l'ADC. Aucune RAM externe n'est nécessaire puisque le F2812 possède une mémoire incorporée (*on-chip memory*) de 18K mots de 16 bits.

La communication avec le DSP est complètement différente de celle avec le EmuDSP puisqu'il ne situe pas dans le même espace mémoire. Le F2812 supporte une inter-

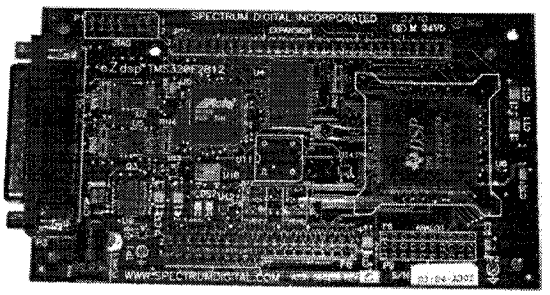
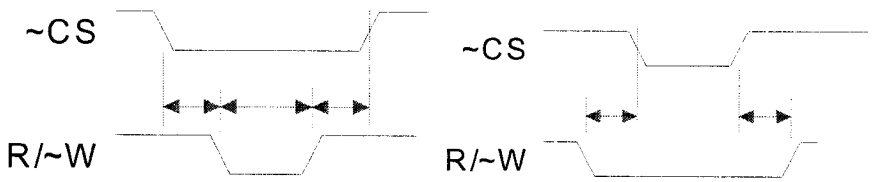


Figure 3.14 Carte de développement eZdspF2812 de Spectrum Digital



(a) TIR2000 (b) DAC/ADC

Figure 3.15 Différence dans les chronogrammes pour le TIR2000 et le DAC/ADC

face de communication sérielle (SCI) à $3,3V$ selon le standard $RS232$. En attendant le fonctionnement du pilote infrarouge, cette communication lente à 19200 bps est utilisée. Tout ordinateur personnel est équipé d'un tel port série, mais fonctionne avec une différence de potentielle de $12V$. Un petit circuit de conversion de niveaux électriques basé sur un `MAX232` de la compagnie Maxim est donc nécessaire.

Les circuits d'alimentation et de contrôle du STM sont refaits et les modules de PiezoScan et de Z-Feedback sont portés sur le DSP. La fréquence d'utilisation du DAC est accélérée à sa fréquence maximale, soit $100KHz$. Un *timer* génère une interruption périodique à chaque $10\mu s$. Sur cette interruption, le signal `LOADDACS` est activé. À $30MHz$, une période de $10\mu s$ ne représente que 300 cycles. Dans ce cours lapse de temps entre deux interruptions successives, on doit être en mesure de calculer la position future du piézo et d'écrire les valeurs V_X et V_Y dans le DAC. La

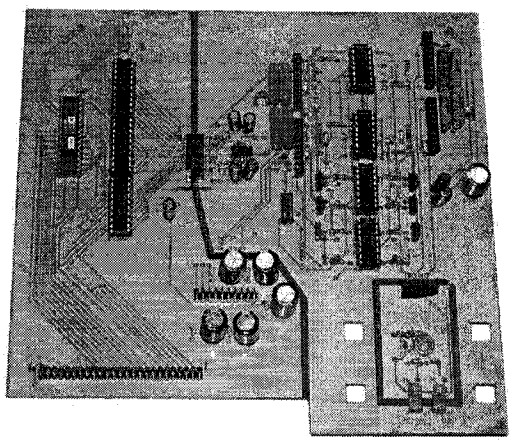


Figure 3.16 Révision du circuit STM

figure VI.1 de l'annexe VI démontre que c'est possible, mais que le temps restant est plutôt mince. Compilé en mode *release*, il reste $5,4\mu s$ entre la fin de l'écriture sur le DAC et le prochain `LOADDACS`.

3.8.1 Communication infrarouge

Vue le peu de temps disponible pour un traitement autre que le balayage du piézo, une crainte pour l'intégration de la communication infrarouge est soulevée. Avec l'aide d'Eric L'Heureux, stagiaire au Laboratoire, un prototype du circuit de transmission IR du robot est construit pour implémenter la communication sur la carte de développement eZdspF2812. Parallèlement, Serge Ngakeng est mandaté pour développer un pilote Linux pour la carte infrarouge Actisys IR2000B/L.

L'allure général d'une trame IrDA 1.1 est reproduite à la figure 3.18. Le TIR2000 se charge «d'épurer» la trame reçue afin de ne placer que les données utiles dans son FIFO de réception. La taille maximale d'une trame est de 2048 octets. La réception et la transmission fonctionnement entièrement sur des interruptions. En réception,

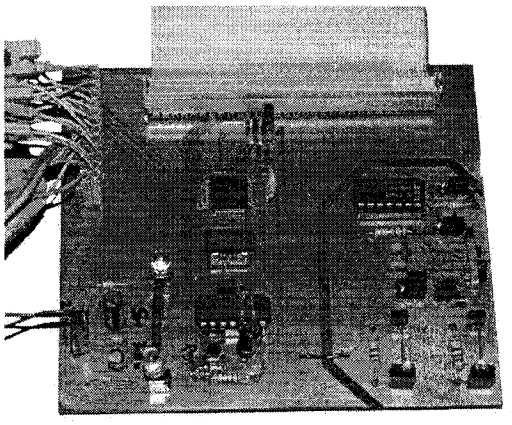


Figure 3.17 Prototype du circuit infrarouge

Preambule	Start Flag	Frame Data	CRC-32	Stop Flag

Figure 3.18 Trame IrDA 1.1

le TIR2000 possède un FIFO interne de 64 octets pour contenir des données reçues en attendant que le processeur maître viennent les récupérer. Lorsque ce FIFO dépasse un certain seuil préconfiguré, une ligne d'interruption est activée. Le seuil utilisé lors de ces premiers tests est de 56 octets.

Lorsqu'une interruption survient, le DSP sait que le FIFO contient au minimum 56 octets qu'il peut lire sans craindre de créer une erreur de perte de données (*buffer underrun*). À l'inverse, le DSP doit réagir rapidement à cet événement afin d'éviter que le FIFO ne déborde et lance une erreur de *buffer overrun*. Dans un cas comme dans l'autre, le TIR devrait être réinitialisé avant d'être à nouveau apte à recevoir ou envoyer des données.

Le TIR2000 n'active qu'une seule ligne d'interruption pour l'ensemble des événements. L'identification de la cause se fait par l'analyse du registre `IIR` (*Interrupt Identification Register*) à l'adresse `0x2`. Les 16 adresses de registres du TIR se par-

tagent d'ailleurs diverses fonctions selon le mode dans lequel il est configuré ou s'il s'agit d'une lecture ou d'une écriture.

Le DSP retire donc du FIFO le nombre d'octets correspondant au seuil à chaque fois que l'interruption `IIR[0]` arrive, plaçant les octets nouvellement reçus à la suite des octets déjà lus. Par contre, la fin d'une trame n'arrive pas forcément sur le seuil du FIFO. Lorsque le `EOF` (*End of Frame*) est détecté par le TIR, il active l'interruption `IIR[7]` et le bit `LSR[5]`. Le DSP passe alors dans le mode de lecture octet par octet. Un octet est lu à la fois jusqu'à ce que l'indicateur `LastByte` (bit 2 du registre `IIR` et bit 2 du registre `LSR`) se lève. La réception est alors terminée, les quatre derniers octets sont la valeur du CRC de la trame et ne font donc pas partie intégrale du message.

Pour tester le mécanisme de réception avant même que ne soit achevé le mécanisme de transmission du côté PC, un ordinateur portable doté d'un port IR et un émetteur USB furent utilisée. Les paquets échangés lors d'un transfert de fichier entre l'ordinateur et le portable suivent le protocole FIR. En plaçant le récepteur HSDL-3602 du circuit prototype entre les deux stations, la communication peut être interceptée. Un analyseur logique déclenché par l'arrivée de l'interruption du FIFO affiche les signaux de contrôle échangés entre le DSP et le TIR. En transférant un fichier dont le contenu est connu d'avance, il est possible de valider la réception non corrompue des données. Un long fichier ASCII ne contenant que des 0 (valeur hexadécimale `0x30`) est utilisé.

La figure VI.2 présente l'allure des signaux de contrôle du TIR lors de la réception continue des données IR. On y voit que sur l'activation du IRQ, de multiples lectures du FIFO d'entrée sont effectuées. S'ensuit une pause jusqu'à l'arrivée du prochain IRQ. La présence de cette pause assure que la lecture du FIFO est plus rapide que son remplissage, condition essentielle pour une communication efficace.

La durée totale pour lire les 56 octets est de $56,5\mu s$. Le code utilisé durant ces tests était, cependant, compilé en mode *debug*, c'est-à-dire qu'aucune optimisation n'était effectuée. L'optimisation du code mène généralement à des gains de performances significatifs.

La figure VI.3 présente la réception d'une fin de trame. On y voit clairement le passage du mode de lecture par seuil du FIFO au mode octet par octet sur le bus d'adresse. En effet, en mode octet par octet, chaque lecture du FIFO (l'adresse `0x0`) est séparée d'une interrogation du bit 5 du registre `LSR` (à l'adresse `0x5`). Si l'on effectue un zoom sur la fin d'une trame (voir Figure VI.4, on peut voir que les lectures arrêtent lorsque la valeur du registre `LSR` est de `0xE`, c'est-à-dire que le bit `LastByte` est à 1. Les quatre derniers octets lus du FIFO (`0x89`, `0x10`, `0x8F` et `0xEE`) sont la valeur du CRC et doivent être retirés du message final.

En mode de transmission, le fonctionnement du TIR est similaire. À mesure que les données sont transmises, le FIFO de sortie se vide. Lorsqu'il franchit un certain seuil, l'IRQ *TX FIFO below threshold* (IIR[1]) est activé. Un remplissage en rafale (*burst*) du FIFO est effectué pour s'assurer qu'une erreur de *buffer underrun* ne survienne pas. Cette suite d'actions continue jusqu'à ce que la fin de la trame soit envoyée. L'indication de la fin de la trame se fait soit automatiquement en fonction de la longueur des données transmises, soit en spécifiant explicitement la fin par l'écriture du bit `EOT` (`ACREG[0]`) juste avant d'écrire le dernier octet de la trame.

La majorité des messages envoyés par le NanoWalker seront courts, à l'exception des messages de données STM qui pourraient atteindre quelques centaines d'octets. Le mode de terminaison *Set-EOT bit* semble le plus flexible et est celui choisi.

Comme pour la réception, deux avenues sont possibles pour gérer les IRQ. Soit qu'un haut seuil du TX FIFO cause l'arrivée peu fréquente d'interruptions, mais

qu'à chaque fois, le nombre d'octets à lire soit grand ou, à l'inverse, qu'un petit seuil cause plusieurs interruptions plus rapides à traiter. L'influence sur le PiezoScan et les autres fonctionnalité du DSP est particulièrement d'intérêt.

La figure VI.5 montre clairement que le remplissage en rafale du TX FIFO lors de l'utilisation d'un seuil de 48 octets monopolise le bus de données pendant environ $39\mu s$. Pendant ce temps, les mises à jour du PiezoScan sont retardées et un total de quatre périodes sont manquées. Les `LOADDACS` sont quand même émis puisqu'ils sont synchronisés sur une interruption du *timer*.

L'effet d'une diminution du seuil à 8 octets (voir Figure VI.6) est notable, le temps nécessaire pour remplir le FIFO passe à environ $10\mu s$, mais revient à toutes les $15\mu s$ à peu près, soit beaucoup plus rapidement que les $120\mu s$ lorsqu'un seuil de 48 octets est utilisé. Par contre, on remarque toujours que les mises à jour du PiezoScan souffrent de la transmission infrarouge, l'écriture de V_x et V_y n'étant pas capable d'être effectuée à tout coup avant l'arrivée du prochains `LOADDACS`.

Plus amples tests devront être faits sur l'intégration de la communication IR et du contrôle du STM puisqu'il s'agit ici des deux tâches aux contraintes temporelles les plus strictes. Diminuer la fréquence de balayage du piézo à $5KHz$ ou moins est l'option la plus simple pour respecter ces contraintes, mais une meilleure segmentation et optimisation du code est à étudier.

Il ne faut pas non plus oublier que le DSP du NanoWalker fonctionne à une fréquence de 18MHz supérieur à la fréquence du F2812 sur la carte eZdspF2812. La présence du CPLD comme intermédiaire entre les composants, le partage du même bus de données et d'adresses modifieront également l'allure des chronogrammes obtenus de l'analyseur logique. S'attarder sur l'intégration du F2812 pourrait s'avérer inutile pour l'atteinte de l'objectif du projet.

Au printemps 2004, le «MacroWalker», le circuit prototype du NanoWalker fabriqué par le MIT, est arrivé au Laboratoire. Les tests des algorithmes et code du STM et IR ont délaissé le F2812 pour migrer vers ce circuit qui présente, lui aussi, des différences majeures par rapport au design final du NanoWalker, mais qui lui est néanmoins beaucoup plus fidèle.

CHAPITRE 4

POSITIONNEMENT ATOMIQUE

Le but du projet NanoWalker est d'effectuer des opérations au niveau atomique mais, pour y arriver, il faut d'abord être capable de se positionner à cette échelle. Ce chapitre présente les efforts mis en œuvre pour relever ce défi de taille. Le fruit de ce travail fut d'ailleurs le sujet de deux articles de conférences (voir annexe II).

Diverses technologies existantes permettent d'obtenir un positionnement de précision. L'interférométrie au laser est l'une des plus connue et des plus couramment utilisée. Elle fait appel aux patrons d'interférences produits par un laser réfléchi sur un objet distant lorsque recombiné avec le laser d'origine. L'analyse du patron d'interférence permet de déterminer le changement de phase subi par la lumière et ainsi de calculer la distance qu'elle a parcourue. La précision atteignable est de l'ordre du nanomètre[15], mais diverses raisons rendent cette technologie inapplicable dans le cadre du projet NanoWalker. En effet, pour positionner un objet, celui-ci doit réfléchir le laser incident vers l'émetteur. Un miroir parfaitement perpendiculaire doit donc y être fixé. Étant un robot mobile, le NanoWalker ne peut garantir la perpendicularité à tout moment. Pis encore, une ligne visuelle directe doit être maintenue avec l'émetteur. Dans une flotte d'une centaine de robots, des obstructions visuelles momentanées sont une quasi-certitude. Certains robots se verraient donc, par moments, impossible à positionner. Cette situation est inacceptable. Le système de positionnement basé sur le PSD ne souffre pas de ces limitations puisqu'il ne fonctionne pas dans le plan des robots comme l'interférométrie au laser (voir Figure 4.1). Par contre, le PSD n'a qu'une précision de $\pm 75 \mu m$. À cette échelle, le positionnement atomique est encore bien loin. À titre d'exemple, un atome de

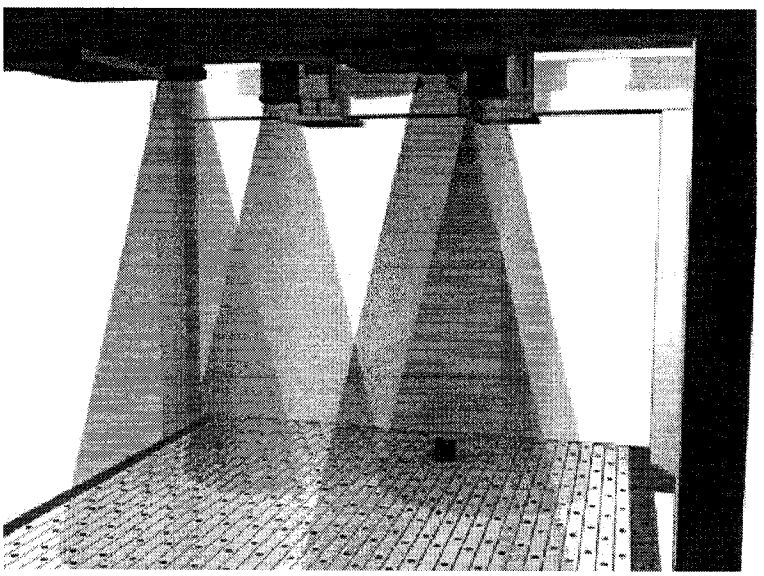


Figure 4.1 Aperçu du système de positionnement global. Le PSD ne fonctionne pas dans le plan des NanoWalkers

carbone possède un rayon atomique de $77,2pm$, soit près de 1 000 000 fois plus petit. D'autres systèmes de meilleure résolution doivent donc être jumelés au PSD pour espérer obtenir un positionnement atomique. La principale contrainte pour ce nouveau système est simple : aucune modification physique ne peut être apporté au NanoWalker. Il doit donc être basé sur la microscopie à effet tunnel. Le STM analyse des surfaces, le système de positionnement multi-stades développé doit donc reposer sur un principe d'analyse de caractéristiques topographiques préalablement gravées dans un matériau.

Au centre de ce système se trouve un matériau bien particulier sur lequel les atomes seront littéralement comptés. Le *Highly Oriented Pyrolytic Graphite* (HOPG)[29, 61, 31] est un matériau artificiel entièrement fait de carbone. L'agencement hexagonal dans le plan XY de ses atomes lui donne une surface plane très résistante à la traction. Par contre, son empilement en feuilles dans l'axe des Z fait de lui

un matériau très facile à «peler». Son principal intérêt se situe dans la régularité de sa structure atomique qui présente très peu de défauts. Lorsque clivé adéquatement, des zones peuvent se retrouver atomiquement planes sur plusieurs microns. La figure 4.2 illustre la structure atomique du HOPG. La distance entre deux atomes consécutifs est de $142pm$ tandis que la distance entre deux couches est de $334,8pm$. Toutes ces caractéristiques font du HOPG un matériau de prédilection pour les microscopes-sondes à balayage (SPM) en général.

Plusieurs ont suggéré l'utilisation du Si7x7 comme substrat au lieu du HOPG. Tout comme le HOPG, le silicium est conducteur et peut être imagé au STM, mais le silicium s'oxyde rapidement à l'air libre. Cet oxyde isolant perturbe la régularité de sa structure atomique et rend impossible l'obtention d'un courant tunnel. Le NanoWalker évolue dans une atmosphère d'hélium, mais des traces d'oxygène sont toujours présentes. Le HOPG s'oxyde aussi, mais très lentement et à une température très élevée[18], ceci ne constitue donc pas un réel problème dans les conditions d'utilisation actuelles. Un autre inconvénient du silicium est la grande quantité de défauts présents sur la surface de ses cristaux.

La figure 4.3 présente une image STM de la structure atomique du HOPG. Les atomes individuels sont clairement visibles. En fait, seulement la moitié des atomes réels sont observables. Comme l'a expliqué Hembacher[31], le STM ne permet de «voir» que les atomes de carbones possédant un voisin directement sous eux (les atomes en gris sur la figure 4.2). La densité de charge des autres atomes ne semble pas suffisante pour être imagée correctement. Ce n'est qu'avec un microscope à force atomique (AFM) qu'il a réussi à observer la présence de ces atomes cachés. Ce n'est cependant pas un problème puisqu'il y fut démontré qu'on peut utiliser la structure «partielle», mais régulière, du HOPG pour compter des atomes et ainsi référencer une position.[39] Deux axes peuvent être superposés à la structure atomique et utilisé comme référentiel afin d'établir un système de coordonnées. Le

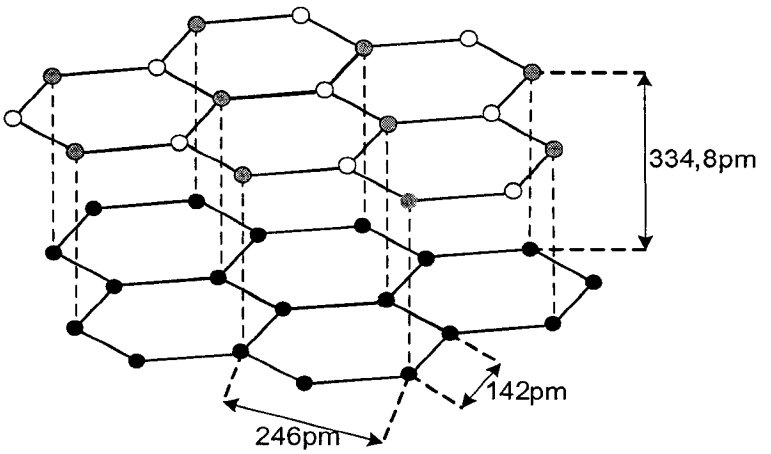


Figure 4.2 Structure atomique du HOPG

travail à faire est clair : décomposer progressivement une zone carrée de $150\mu m$ en plus petites surfaces de travail où le STM pourra compter des atomes. Plusieurs stratégies furent envisagées pour accomplir ceci. Deux ont été testées et une a été retenue.

4.1 Étapes de positionnement

Un positionnement en quatre étapes est élaboré. La première est celle du PSD. Durant l'été 2002, la précision espérée pour le PSD était de $\pm 15\mu m$, mais les tests de Eric Aboussouan[1] ont plus tard démontré que ce n'était pas réalisable. La précision réelle permet de positionner adéquatement un NanoWalker au-dessus d'un porte-échantillon du *Powerfloor*. La pointe de son STM se trouve alors dans une zone carrée de $150\mu m$ d'arête, l'atome recherché se situe quelque part à l'intérieur de ce carré. Mais comme expliqué au chapitre 3, la plage dynamique du piézo du STM du NanoWalker n'est cependant que de $\pm 1,7438\mu m$ en XY si l'on souhaite

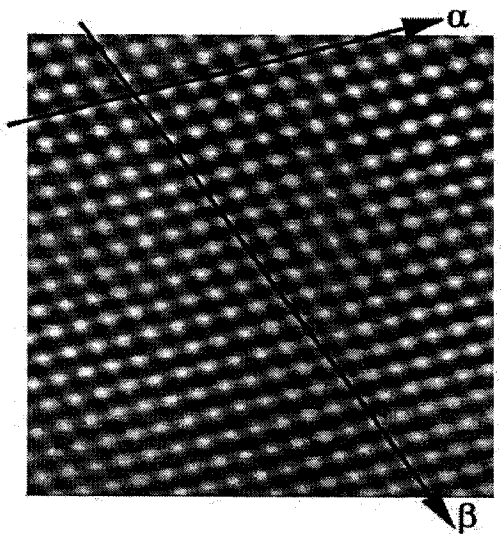


Figure 4.3 Image STM atomique du HOPG

préserver une certaine marge de manoeuvre sur l'axe des Z. L'algorithme IAPA[58], développé conjointement avec Pierre-Alain Dumas et Marc-Antoine Ducas et présenté en Nouvelle Orléans lors de la conférence *IEEE International Conference on Robotics & Automation 2004*, vise à palier aux lacunes du PSD et à placer la pointe du microscope à l'intérieur d'une grille d'une trentaine de microns de côté contenant l'atome cherché. Cette grille, dont les premières ébauches furent présentées à la conférence canadienne sur le Génie Électrique et Informatique 2003[59], constitue la troisième étape de positionnement. Son but est de fragmenter le HOPG en plusieurs zones de travail uniquement identifiables où le STM du robot peut travailler. La dimension des zones de travail est contrainte par la réalité du NanoWalker. La zone de travail et son code d'identification doivent être compris à l'intérieur de n'importe quel scan aléatoire du STM. Il faut cependant noter que les principes développés ici demeurent conceptuellement indépendants du projet NanoWalker. Celui-ci ne joue le rôle que de premier contexte d'implémentation. La généralisa-

tion du travail réalisé à tout appareil muni d'un STM, et même d'un SPM, est possible.

4.1.1 Code binaire

En partant de l'hypothèse que le STM est, principalement à cause des non linéarités de son piézo dans le plan XY, mieux adapté pour des mesures de profondeur que des mesures de largeur, les premiers essais de décomposition tentent de mettre ceci à profit. La surface de HOPG est fragmentée en une matrice de fenêtres de travail identifiées à l'aide de codes binaires. Un code est formé de bits rectangulaires gravés à différentes profondeurs dans le matériau. Quatre profondeurs sont nécessaires. Un bit de début, très profond, identifie le sens de lecture du code. Chaque bit est séparé par un «espace» gravé à une profondeur moyenne. Les «zéros» sont gravés à une profondeur se situant entre les espaces et le bit de début, tandis que les «uns» sont représentés par une zone non gravée du matériau (voir Figure 4.4). La figure 4.5 présente un exemple de fenêtre avec le code associé. Chaque fenêtre mesure au total $4,0\mu m$ et son code binaire est gravé sur chacune des faces de sorte que n'importe quel scan du STM puisse imager au minimum un code complet. L'analyse de l'image résultante permet de connaître la position du STM au dessus de la matrice de fenêtres totalisant $32\mu m$ (voir Figure 4.6).

La gravure de cette grille sur le HOPG, dont la largeur des lignes atteint parfois quelques dizaines de nanomètres seulement, nécessite une technologie de pointe. Le faisceau d'ions focalisés (FIB)[2, 45, 36] est choisi pour sa facilité et sa rapidité de prototypage à faible coût. En effet, bien connu du monde de l'électronique, le FIB permet de graver directement un échantillon sans avoir recours à un masque maître comme en lithographie conventionnelle. Son fonctionnement est relativement simple : des ions de gallium bombardent une surface, ce bombardement pulvérise

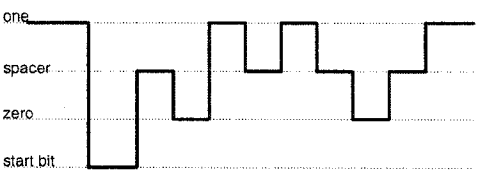


Figure 4.4 Section d'un code binaire

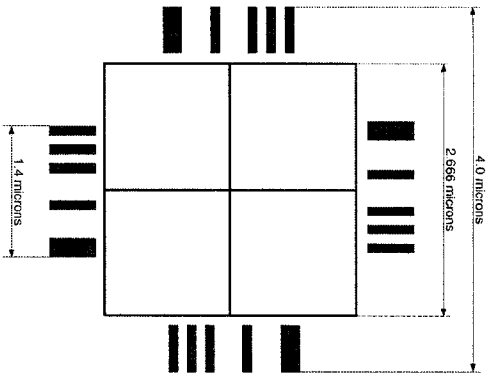


Figure 4.5 Fenêtre identifiée par son code binaire (en noir est le matériau gravé)

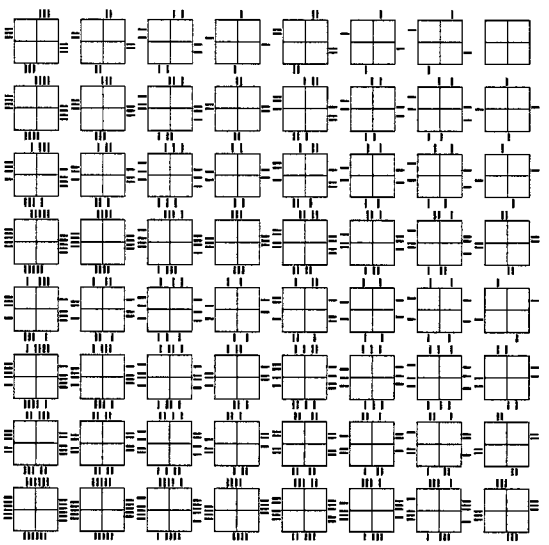


Figure 4.6 Matrice de 8×8 fenêtres binaires

littéralement le matériau frappé. Des ions secondaires, des atomes et des électrons secondaires sont délogés, captés et analysés résultant en une image du matériau. Cette technique d'imagerie est destructive puisqu'elle altère l'état de la surface. En jouant avec l'intensité du courant d'accélération des ions de Ga+ et avec la durée d'exposition, la quantité de matériau retiré, donc la profondeur de gravure, peut être contrôlée.

D'autres techniques de gravure, tel l'ultraviolet extrême (*Extreme UV*)[57] ou la gravure au plasma d'oxygène (*Oxygen Plasma Etching*)[37] permettent de graver du HOPG, mais aucune ne semble avoir été essayée pour des patrons possédant de longs segments droits. Une autre technique très prometteuse, la *nanoimprint lithography* (NIL)[30], permet de graver de très fines lignes sous les $10nm$. Pour ce faire, un moule précis est estampillé sur un polymère chauffé à haute température. En refroidissant, le polymère durcit et garde la forme négative du moule. Le polymère le plus couramment utilisé est le PMMA. N'étant pas conducteur, il ne pourrait pas être utilisée directement avec un STM. Une variante, le *step-and-flash imprint lithography* (S-FIL), utilise un photopolymère organosilicié durcissant lorsque éclairé à la lumière UV, ce qui permet un estampillage à température de la pièce. Cette nouvelle technologie a fait une montée rapide dans le domaine de la nanofabrication et plusieurs groupes de recherche s'y intéressent. À la connaissance des membres du laboratoire, aucun test sur le HOPG n'existe dans la communauté scientifique. Par contre, des tests ont démontré que le NIL est applicable sur du silicium. L'effort pour la création du masque initial est important, mais la production en série subséquente est faite à moindre coût que les autres techniques de lithographie. Ce facteur financier devra être considéré lorsque viendra le temps de graver plusieurs grilles sur chacun des cents échantillons de HOPG intégrés au *Powerfloor*.

Les gravures initiales du code binaire furent effectuées au centre de caractérisation microscopique des matériaux (CM^2) de l'École Polytechnique de Montréal à l'aide

d'un FIB Hitachi 2000A. Trois passes sont nécessaires pour graver une telle grille, une pour chaque profondeur. Pour tenter d'obtenir le meilleur rapport d'aspect possible, une technique de superposition des masques est utilisées. En premier, les bits de début sont gravés à un tiers de leur profondeur finale. Ensuite, la superposition des bits de début et des bits 0 sont gravés avec les mêmes paramètres. Ceci porte à deux tiers la profondeur des bits de début. Tous les masques sont gravés en même temps pour terminer la gravure. Dans le mode *vector scan controller*, un simple fichier bitmap de 512×512 pixels spécifie le patron à graver. La correspondance entre ce fichier et la zone gravée du matériau est fonction d'un facteur de grossissement de la machine. Trente-deux microns est la taille de fenêtre choisie pour englober complètement la grille. Chaque pixel de l'image représente donc $62,5nm$. L'utilisation d'une plus petite fenêtre améliorerait la résolution de la gravure, mais des erreurs supplémentaires d'alignement seraient induites, le Hitachi 2000A ne permettant pas de programmer une séquence de gravure complexe avec déplacements.

Les résultats obtenus de cette gravure sont décevants. À l'université de Montréal, au laboratoire de caractérisation de matériaux (LCM) du département de Chimie, la grille fut d'abord observée à l'aide d'un STM commercial, un Autoprobe CP de la compagnie Digital Instruments. Mais, des problèmes de bruits électroniques nous ont poussé à utiliser un autre STM de meilleure qualité, un Multimode de la même compagnie (voir Figure 4.8). Le profil des codes binaires n'est tout simplement pas visible (voir Figure 4.7). L'explication probable est que lors des trois étapes de gravure, malgré les précautions prises, les particules délogées se déposent dans les trous creusés précédemment et le profil se retrouve ainsi lissé.

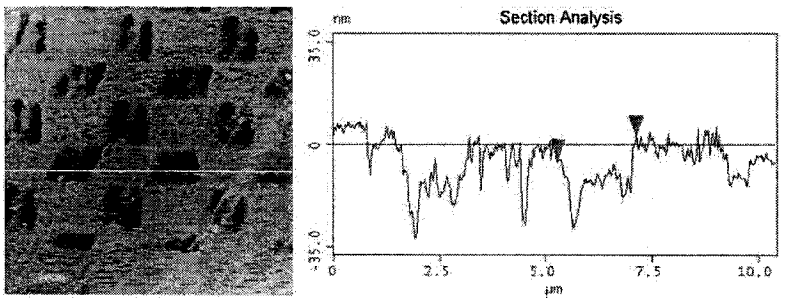
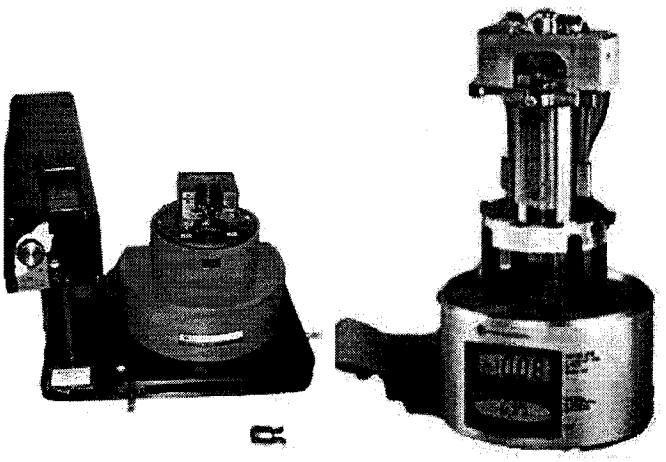


Figure 4.7 Vue de coupe d'un code binaire



(a) Autoprobe CP (b) Multimode

Figure 4.8 STM de la compagnie Digital Instruments (maintenant Veeco Instruments)

4.1.2 Patron de lignes

L'idée du code binaire est donc abandonnée au profit d'une technique d'encodage plus simple et plus robuste à l'étape de gravure. Un patron de lignes de largeurs variables a été conçu (voir Figure 4.9). Chaque ligne d'un ensemble horizontal ou vertical possède une largeur unique. Toute la grille est gravée à la même profondeur, éliminant ainsi le principal problème du code binaire. Chaque intersection est ainsi identifiable uniquement à partir de la largeur de ses deux lignes constitutrices. Les parties non gravées définissent les zones de travail du STM. La gravure de cette grille, surnommée *Fine Positioning Grid* (FPG), fut sous-traitée à FIB International Inc., une compagnie californienne. Cette compagnie prétend être capable d'obtenir une résolution de $7nm$ à l'intérieur d'une fenêtre de gravure de $30\mu m \times 30\mu m$. La profondeur de la gravure est fixée à $100nm$. Le fruit de leur travail, imagé à l'aide d'un microscope à balayage électronique (SEM), est présenté à la figure 4.10. Il faut noter que le fait d'avoir imagé au SEM la grille constitue une erreur. Quoique totalement non destructive, cette technique d'imagerie dépose une couche de quelques angströms de carbone sur le matériau imagé. Cette couche ne suit pas la structure atomique régulière du HOPG et rend ainsi impossible la dernière étape de positionnement, celle du comptage d'atomes. On est présentement encore à valider les algorithmes d'approche, cette mince couche de carbone ne pose donc aucun problème pour les tests en cours, mais cette erreur ne doit pas se répéter dans le futur.

Lorsqu'observé avec le Multimode en mode AFM, les résultats obtenus de cette deuxième version de la grille sont très prometteurs. L'étude du profil des images AFM avec Pierre-Alain Dumas[22] a permis d'établir que la mesure de la largeur des lignes est suffisamment précise pour identifier de façon unique une ligne de la grille. La figure 4.11 présente la section d'une ligne d'une largeur théorique de

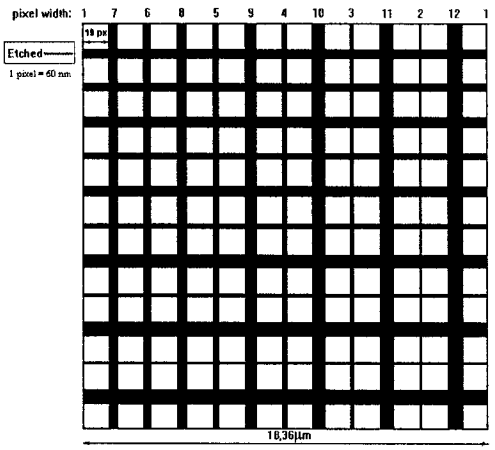


Figure 4.9 Dimension de la *Fine Positioning Grid*. La ligne la plus mince à une largeur de $60nm$, la plus large est de $720nm$. Chaque zone de travail est de $1,14\mu m$.

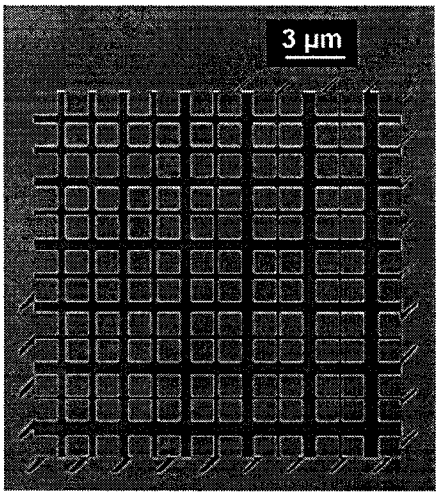


Figure 4.10 FPG imagée au SEM

$360nm$. La largeur expérimentale obtenue est de $358,55nm$. L'analyse manuelle de toutes les largeurs d'une rangée du quadrillage semble indiquer que le processus de détection de la position peut être automatisé et que les résultats seront fidèles aux attentes. Le tableau 4.1 présente les résultats de l'analyse des profils. Les largeurs ont été mesurées à partir du milieu des arrêtes plutôt que crêtes à crêtes.

On note cependant encore un problème de fabrication. En effet, la présence de boursouflures sur deux des quatre faces de la surface de travail (voir Figure 4.12) semble due à un problème de calibration du FIB Micrion 9500 utilisé par FIB International. Afin de s'assurer qu'il ne s'agissait pas d'un artéfact de pointe, trois pointes AFM différentes ont été utilisées et l'angle de l'échantillon a été varié afin de vérifier que les boursouflures perduraient. Le fait qu'elles se situent toujours du même côté de l'échantillon laisse croire qu'il ne s'agit pas d'un problème d'imagerie AFM, mais bien d'un de fabrication. FIB International est d'accord pour dire qu'un

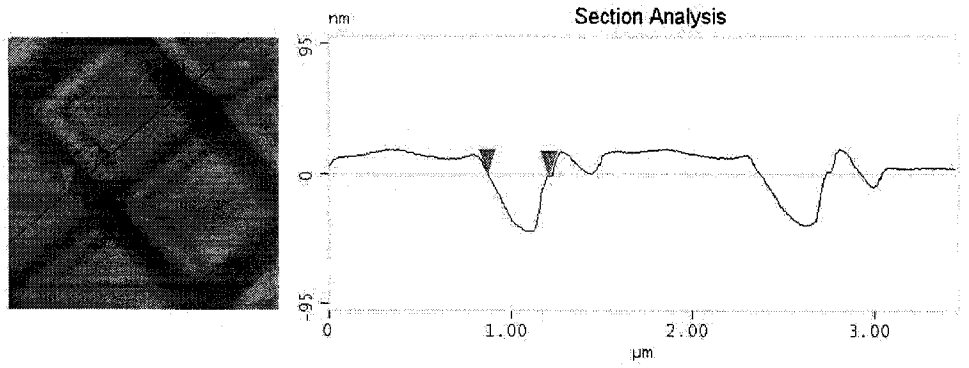
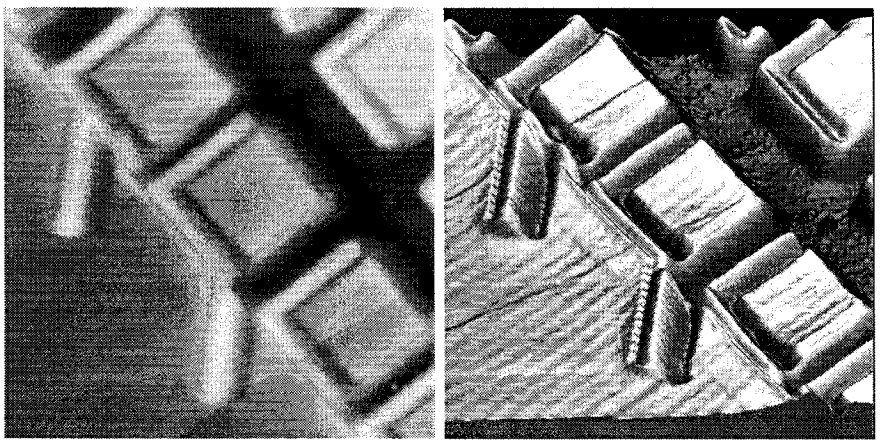


Figure 4.11 Vue de coupe d'une ligne de la FPG imagée avec un AFM (la largeur théorique de la ligne est $360nm$)

Tableau 4.1 Résultats d'analyse des profils d'une rangée de la FPG

Ligne	Mesure 1	Mesure 2	Mesure 3	Mesure 4	Moyenne	Erreur
60 nm	53,37	47,20	53,18	64,88	54,66	-5,34
120 nm	125,32	118,79	118,62	124,48	121,80	1,80
180 nm	194,72	194,38	194,69	201,82	196,40	16,40
240 nm	232,72	212,01	220,84	232,09	224,42	-15,59
300 nm	277,11	272,46	283,52	255,29	272,10	-27,91
360 nm	319,91	317,28	326,64	326,37	322,55	-37,45
420 nm	450,17	465,43	449,52	477,05	460,54	40,54
480 nm	464,50	445,16	451,46	453,78	453,73	-26,28
540 nm	491,80	511,47	500,39	514,63	504,57	-35,43
600 nm	560,27	558,16	569,89	543,51	557,96	-42,04
660 nm	697,70	711,00	706,23	696,30	702,81	42,81
720 nm	734,31	760,52	767,58	769,34	757,94	37,94



(a) Vue 2D (b) Vue 3D

Figure 4.12 Boursouflures sur deux des quatre côtés de la FPG

mauvais alignement du faisceau d'ions pourrait expliquer de telles boursouflures.

Le département de Physique de l'École Polytechnique de Montréal a, par la suite, procédé à l'installation d'un FIB de haute qualité (un Strata DB235M) dans leur salles blanches. Aurélien Masseboeuf[44], stagiaire français au Laboratoire, s'est penché sur l'amélioration de la gravure au FIB et sur la formation de points de référence atomique à l'intérieur même de la zone de travail. En effet, la zone de travail de $1,14\mu m$ contient à peu près 4000 structures hexagonales de HOPG. Pour référencer un atome unique, il faut savoir d'où commencer à compter. Les intersections elles-mêmes sont trop chaotiques au niveau atomique pour penser en utiliser le coin comme origine d'un système d'axes. La disposition d'atomes de bore est étudiée. Ces «gros» atomes, faciles à repérer, feraient ainsi office de balises pour diriger le STM.

4.2 Iterative Approach Positioning Algorithm

Parallèlement, un système d'appoint pour palier aux lacunes du PSD a dû être développé. Ceci a donné lieu à la *Coarse Positioning Grid* (CPG) et à l'*Iterative Approach Positioning Algorithm* (IAPA). La FPG est placée au centre de la CPG, une surface divisée en quatre quadrants. Chaque quadrant comporte des lignes de largeurs différentes. Toutes les lignes d'un même quadrant sont de même largeur comme présentée à la figure 4.13. L'idée derrière IAPA est de converger progressivement vers le centre de la CPG en se basant uniquement sur une mesure de la largeur des lignes d'un quadrant et le nombre d'itérations effectuées depuis le lancement de l'algorithme. On note trois objectifs :

1. Minimiser le nombre d'itérations nécessaires pour atteindre la FPG ;
2. Tolérer les erreurs de calibration, sur le déplacement ou sur la lecture de la largeur d'une ligne ;
3. Minimiser la complexité de l'algorithme afin de l'embarquer sur le DSP du robot.

La taille de la CPG est régie par l'incertitude du PSD. Dans le cas présent, elle est de $150\mu m$. Après la première étape de positionnement, le STM du robot se trouve quelque part à l'intérieur de la CPG, mais on ne peut dire où exactement. En effectuant un scan, le robot caractérise sa situation comme l'une des trois suivantes :

1. Le STM est au dessus de la FPG, l'algorithme IAPA se termine puisque l'objectif est atteint et le stade de positionnement fin prend le dessus ;
2. L'analyse du profil retourne le quadrant dans lequel se trouve le STM. Ce dernier doit être déplacé dans la direction du quadrant opposé d'une distance d_i fonction de l'itération courante ;
3. L'analyse du profil est incapable de déterminer le quadrant dans lequel se trouve le STM. Une sous-routine de gestion d'erreurs est lancée.

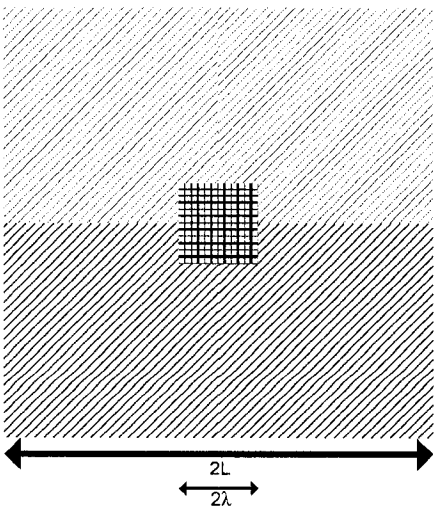


Figure 4.13 Dimension de la *Coarse Positioning Grid*

Si l'on considère pour la CPG un côté de longueur $2L$, une FPG avec un côté de longueur 2λ et un déplacement sans erreur, le nombre d'itérations maximal théorique pour atteindre la FPG est de

$$n = \left\lceil \log_2 \left(\frac{L}{\lambda} \right) \right\rceil \tag{4.1}$$

Ce résultat est fondé sur le même principe qu'une recherche dichotomique. À l'itération 0, le STM peut se trouver n'importe où sur la surface $4L^2$ de la CPG. Dans le cas moyen, la distance à parcourir pour atteindre le centre de la CPG est la moitié de la diagonale d'un quadrant, soit $\frac{\sqrt{2}L}{2}$. Après une itération, le STM est certain de se trouver dans une surface de $\frac{4L^2}{4} = L^2$ centrée sur la CPG. Après deux itérations, la surface est de $\frac{L^2}{4}$, et ainsi de suite. D'une forme générale, après n itérations, on veut obtenir

$$\frac{4L^2}{2^{2(n-1)}} = \lambda^2 \tag{4.2}$$

soit la superficie de la FPG. La distance à parcourir à chaque itération est donc

$$d_i = \frac{\sqrt{2}L}{2^i} \tag{4.3}$$

où i=1,2... est le numéro de l'itération. Un exemple d'évolution de l'algorithme est présenté à la figure 4.14. Trois itérations sont nécessaires pour atteindre la FPG, ce qui correspond au maximum théorique pour une grille avec un rapport de longueurs $\frac{L}{\lambda} = 5$. Des simulations MATLab ont permis d'étudier l'évolution du nombre d'itérations en fonction de la taille de la CPG. La figure 4.15 présente le résultat de 1000 essais de l'algorithme IAPA à partir de positions initiales aléatoires. On y voit que le nombre moyen d'itérations (représenté par un cercle) se situe sous la valeur théorique représentée ici avant l'application du plafond par une ligne pointillée. La valeur maximale du nombre d'itérations pour les 1000 essais est illustrée par un **x**. La largeur de la FPG utilisée pour l'ensemble des simulations est de $9,18\mu m$, soit la dimension actuelle de la FPG.

Les résultats présentés sont idéaux mais, en se déplaçant, le robot peut dévier. En considérant une erreur de e_d sur la distance et une erreur angulaire de e_φ sur l'angle lors du déplacement, la position finale du STM peut être n'importe où à l'intérieur d'un arc de cercle comme illustré à la figure 4.16. L'effet de ces erreurs sur le nombre d'itérations est présenté à la figure 4.17. Bien que le nombre moyen d'itérations nécessaires demeure sous la valeur théorique, la valeur maximale augmente.

Des erreurs d'identification de la position peuvent également se produire. Les causes les plus probables sont une erreur de lecture STM ou un mauvais alignement lors de la gravure des quadrants de la CPG. En effet, une surface de la taille du CPG ne peut être gravée en une seule passe de FIB tout en maintenant une résolution adéquate. Elle doit être fragmentée en plusieurs petites fenêtres de gravure. L'erreur d'alignement peut parfois atteindre jusqu'à $2\mu m$. Lorsque confronté à l'impossibi-

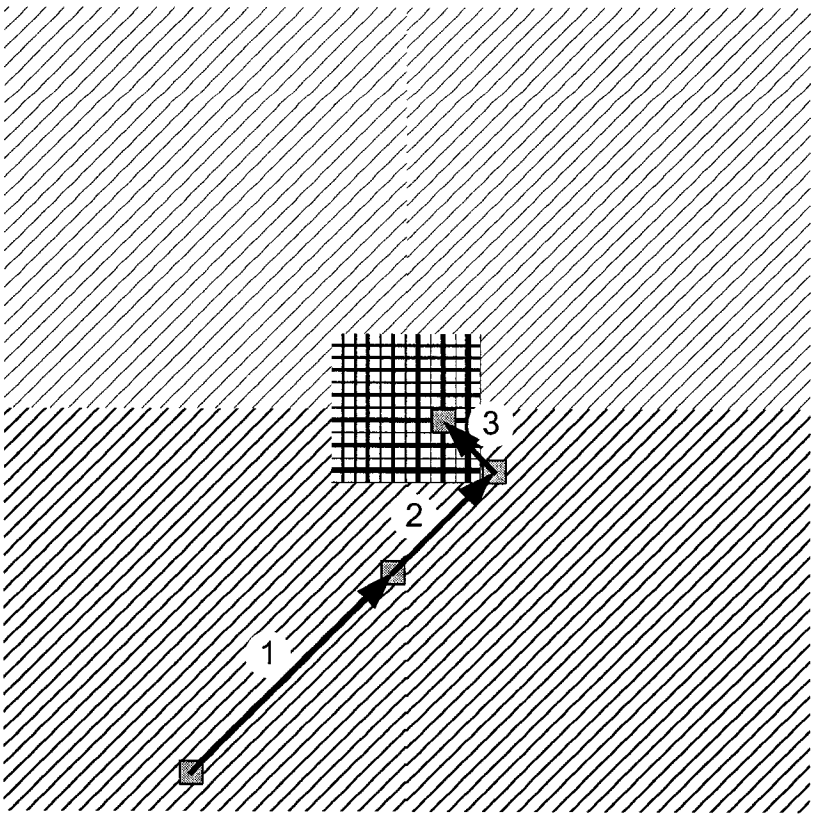


Figure 4.14 Exemple d'évolution de l'algorithme IAPA. Le carré représente la surface que le STM peut imager après chaque itération.

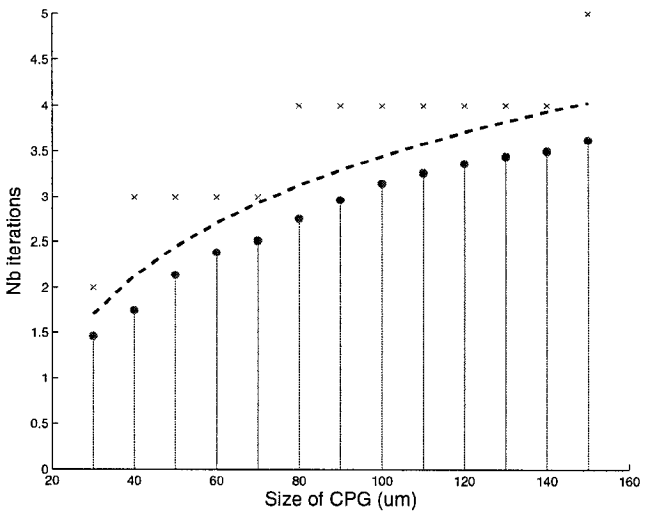


Figure 4.15 Simulation de l'algorithme IAPA pour différentes taille de CPG. La taille de la FPG est fixe à $9,18\mu m$. La ligne pointillée représente le nombre d'itérations théorique avant le plafond, le cercle représente la moyenne de 1000 essais et le x représente le nombre maximal d'itérations.

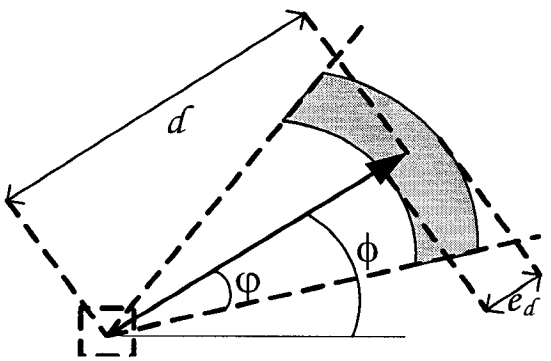
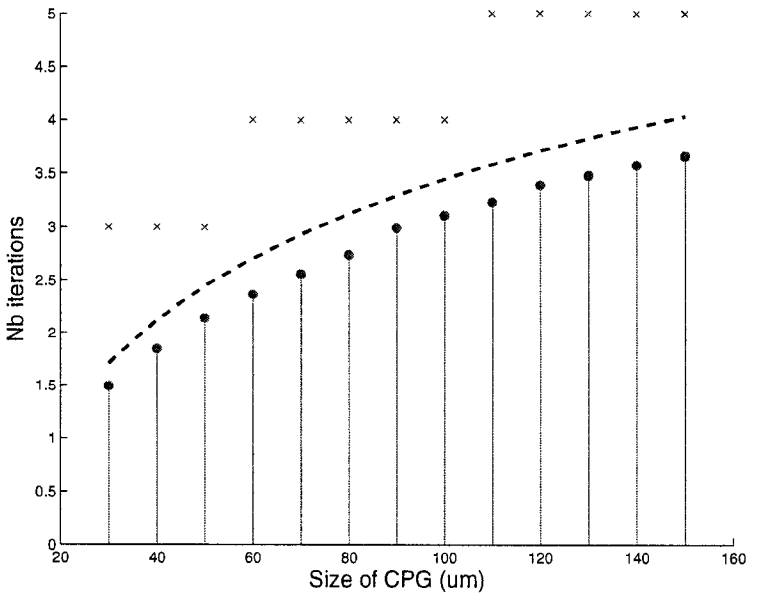
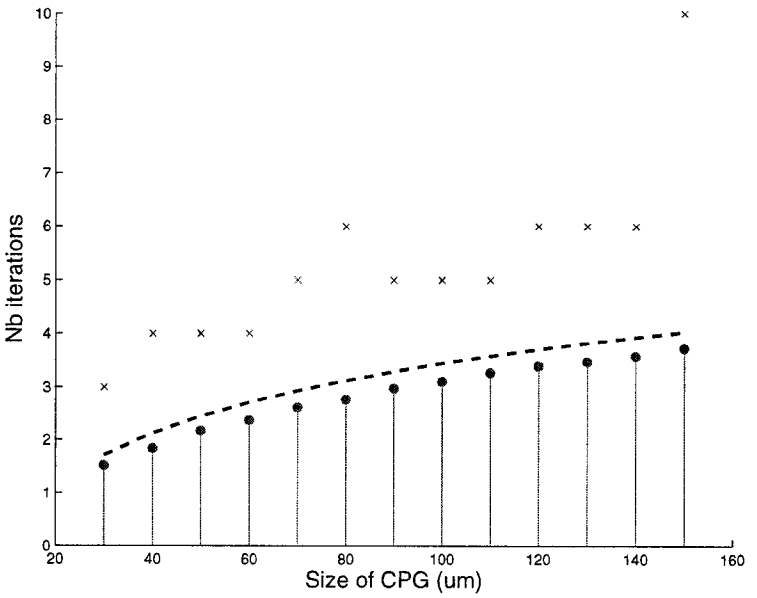


Figure 4.16 Incertitude reliée au déplacement du robot lors de l'algorithme IAPA. Sur un déplacement d'une distance d à un angle ϕ, l'erreur est de $\pm e_d$ et $\pm\varphi$.



(a) $e_d = \pm 1,5\mu m$, $e_\varphi = \pm 1,08^\circ$



(b) $e_d = \pm 3\mu m$, $e_\varphi = \pm 1,08^\circ$

Figure 4.17 Variation de l'algorithme IAPA selon différentes erreurs sur les déplacements.

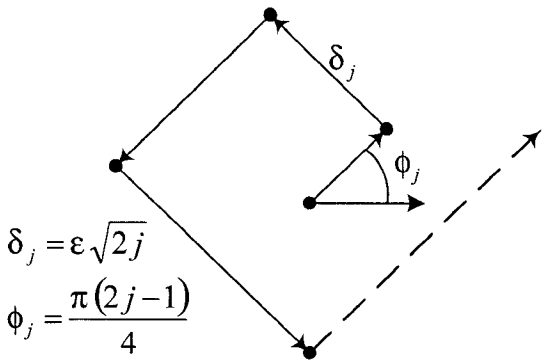


Figure 4.18 Sous-routine d'erreur de l'algorithme IAPA. j est le nombre d'erreurs d'idenfication successives, ϵ est une constante égale à deux fois l'erreur d'alignement.

lité d'identifier sa position, IAPA se lance dans une routine visant à déplacer le STM selon une spirale grandissante (voir Figure 4.18). La longueur du déplacement augmente progressivement à mesure que le nombre d'erreurs consécutives se répètent. Cette spirale a pour objectif de déplacer le STM légèrement et de reprendre un scan afin de lever l'indétermination. L'effet sur l'algorithme peut être important. Dans les pires cas, celui-ci peut diverger et ne jamais atteindre la FPG. Cette situation se caractérise par un nombre d'erreurs d'identification successives élevé. Lorsque détectée, on doit mettre fin à l'algorithme, se repositionner avec le PSD et espérer avoir une meilleure situation initiale. La figure 4.19 présente le résultat obtenu par IAPA lorsqu'une erreur d'alignement des quadrants de $1\mu m$ vient s'ajouter à l'erreur sur les déplacements.

4.3 Conclusion

L'utilisation efficace de l'algorithme IAPA et de la FPG passe par un contrôle accru des fonctionnalités d'un STM, ou même d'un SPM. En effet, l'utilisation d'un AFM

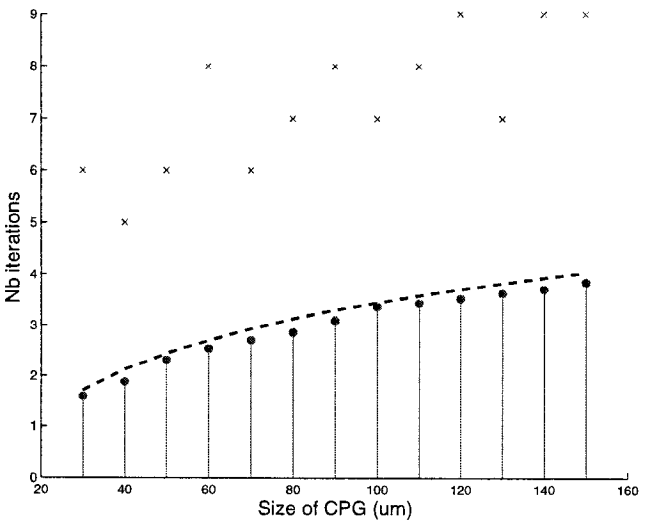
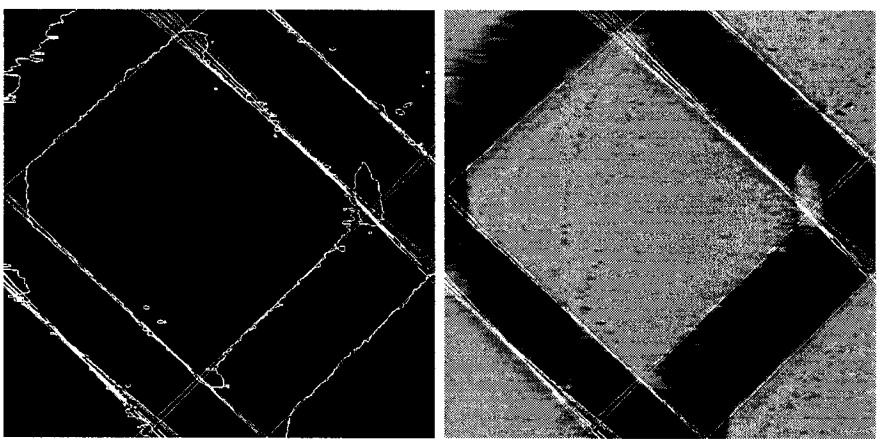


Figure 4.19 Algorithme IAPA avec une erreur d'alignement des quadrants de $1\mu m$ en plus des erreurs sur le déplacement ($e_d = \pm 1,5\mu m$, $e_\varphi = \pm 1,08^\circ$)

est tout à fait acceptable lorsque l'on cherche à atteindre une précision de positionnement de quelques dizaines de nanomètres, ce n'est que lorsque l'on recherche une précision atomique que le STM devient réellement nécessaire. Par contre, peu importe l'instrument utilisé, capturer une image complète de 512 × 512 pixels et en effectuer l'analyse par traitement d'images n'est pas optimal, ni même désiré. Frédéric Nguyenphat-Therrien, à l'hiver 2003, fut mandaté d'étudier les techniques de traitement d'images pour extraire la largeur des lignes de la FPG à partir d'un scan STM réel. En combinant les techniques de seuillage binaire (*thresholding*), de détection d'arrêtes selon l'algorithme de Canny[12] et d'extraction de lignes selon la transformée de Hough[47, 7], des résultats encourageant on été obtenus, mais à un coût de calcul trop élevé pour penser embarquer ce traitement dans le robot. Dans quelques années, un changement du processeur du robot pourrait permettre un tel traitement, mais il faudrait toujours capturer une image complète de la surface avant de l'analyser. Cette longue étape aurait avantage à être évitée. Un exemple

(a) Image originale (b) Après seuillage binaire (c) Après détection d'arrêtes (Canny)



(d) Après extraction des lignes (Hough) (e) Superposition sur image originale

Figure 4.20 Évolution du processus d'extraction des lignes d'une image de la FPG

d'évolution de l'algorithme d'extraction est présenté à la figure 4.20.

Comme alternative, il est envisagé de se baser que sur l'analyse de quelques profils horizontaux et verticaux de la surface plutôt qu'une image complète. Afin de maximiser la lisibilité d'une brusque transition verticale au SPM, il est recommandé d'effectuer un balayage à un angle de 90° de la transition. Dans le cas de la FPG, un balayage à un angle de 45 degrés maximise la lisibilité des deux séries de lignes constitutrices. Puisqu'on connaît la position du robot avec une incertitude angulaire de $1,08^{\circ}$ selon les calculs de Eric Aboussouan, on peut indiquer au robot son

orientation initiale par rapport à la grille sur laquelle il se trouve et ainsi tâcher de maximiser la qualité des profils obtenus. L'analyse des profils avec des outils comme la transformée de Fourier sans passer par la capture et l'envoi via IR d'une image complète permettrait d'accélérer grandement l'exécution de l'algorithme.

Pierre-Alain Dumas, Marc-Antoine Ducas et Aurélien Masseboeuf étudient actuellement une variante au patron de lignes afin de se baser sur une variation de la fréquence des lignes plutôt qu'une variation de largeur. Selon les premiers tests effectués, cette approche semble plus robuste et le niveau de confiance dans l'identification des quadrants d'autant plus élevé.

CONCLUSION

La nanorobotique est en pleine expansion. L'intégration des disciplines scientifiques est à la base de son succès et, par le fait, de sa complexité. L'impact d'une décision peut se ressentir dans une foule de domaines qui semblent, *a priori*, indépendants. L'informatique qui vise à contrôler l'électronique, la mécanique et la physique d'un projet doit être consciente de cette réalité. Une vision d'ensemble est primordiale. Le projet NanoWalker ne fait pas exception.

Dans le présent mémoire, l'établissement des bases du système informatique nécessite des incursions dans plusieurs autres domaines du génie. En cherchant à positionner la pointe d'un STM monté sur un robot miniature, une architecture distribuée est développée. Son but est d'assurer la survie des robots, de contrôler et de coordonner leurs actions. À l'aide d'un premier système de positionnement infrarouge, une précision de $\pm 75 \mu m$ est atteignable.

Basé sur la microscopie à effet tunnel, un second système de positionnement plus précis est nécessaire. La «grille atomique» présentée au chapitre 4 utilise un patron de lignes pour décomposer une large surface en une multitude de zones de travail de $1,14 \mu m$. L'algorithme IAPA permet de converger vers le centre de la *Coarse Positioning Grid* jusqu'à une *Fine Positioning Grid* en un nombre d'itérations proportionnel au logarithme du rapport de longueur des grilles.

Le contrôle du microscope à effet tunnel dans le DSP est abordé au chapitre 3. On y développe l'algorithme de balayage du piézo et la boucle de contrôle sur la valeur du courant tunnel. L'intégration de la communication infrarouge est également débutée. L'étude des signaux à l'analyseur logique identifie cependant un problème : le traitement des tâches associées à la communication est trop long et nuit au

balayage du piézo. Ceci porte à croire que le travail d'intégration et d'optimisation dans le DSP réel sera critique si le NanoWalker est pour fonctionner un jour.

Certains aspects du robot sont toujours flous et incertains, ce qui a comme conséquences de retarder l'implémentation et la validation d'autres modules. Par exemple, tant qu'une solution mécanique au problème d'approche grossière *(coarse approach)* du STM n'est pas trouvée, le circuit électronique du NanoWalker ne peut être finalisé. Ceci a une incidence directe sur le contrôle logiciel et sur les tests du STM lui-même. Les problèmes de boucles de mises à la terre empêchent d'obtenir un courant tunnel et sont venus mettre un terme aux tests à l'université McGill.

D'un point de vue logiciel, l'avancement est satisfaisant. Certains problèmes futurs pourraient néanmoins survenir. Ceux-ci nécessiteront une étude approfondie. Par exemple, le débit de la communication infrarouge de $4Mbps$ entre les robots et le système de contrôle pourrait s'avérer insuffisant lorsqu'un grand nombre de robots est présent. Considérons une image «standard» provenant d'un SPM. La taille de l'information utile pour une image de 512 × 512 pixels avec une profondeur de 8 bits/pixel est de 2097152 bits, et ce, sans compter les bits utilisés dans l'entête du fichier image. On peut s'attendre à une fréquence de balayage STM avoisinant les $5Hz$. On obtient donc un volume de données utiles de 20480 bits par seconde. Lorsque l'on considère une flotte de 100 robots *a priori* équivalents, la bande passante totale devrait être divisée de façon égale entre eux. La bande passante disponible par robot est donc de $40kbps$. À elles seules, les données provenant du STM occuperait plus de 50% de la bande passante totale théorique disponible

par robot.

$$(512 \times 512)\frac{pixels}{img} \times 8\frac{bits}{pixels} = 2097152\frac{bits}{img} \tag{4.4}$$

$$\frac{512 lignes}{5Hz} = 102,4\frac{s}{img} \tag{4.5}$$

$$\frac{2097152 bits}{102,4s} = 20480 bps \tag{4.6}$$

$$\frac{20480 bps}{40 kpbs} = 0,512 \tag{4.7}$$

Plusieurs facteurs viennent réduire la performance réelle du médium de communication. Les messages de synchronisation échangés avec les différents robots, les entêtes présentes dans chacun de ces messages, ainsi que les latences inévitables associées à la lecture et à l'envoi de messages viennent tous diminuer l'efficacité de la communication.

Une solution potentielle consisterait à adapter dynamiquement les temps de transmission alloués à chaque robot pour transmettre les données recueillies. Les robots effectuant des opérations générant un volume de données plus imposant se verraient allouer un plus grand temps pour transférer leur données. Lorsqu'un certain seuil critique de robots effectuant des «opérations gourmandes» serait dépassé, les nouvelles opérations gourmandes seraient retardées à cause du manque de ressources. On pourrait également compresser les données avant de les transmettre via l'IR. Ceci imposerait cependant une surcharge supplémentaire au DSP. Prévoyant déjà un DSP très sollicité, il serait important de s'assurer que cela ne nuira pas aux autres tâches déjà prises en charge.

Un autre problème potentiel concerne le déplacement des robots. Chaque déplacement induira inévitablement des vibrations sur le reste du *Powerfloor*. Il faudra caractériser ces vibrations et prévoir leurs effets sur les mesures STM prises à proxi-

mité. Une solution simple pour éviter tout problème serait d'interdire le déplacement d'un NanoWalker lorsqu'un autre NanoWalker utilise son STM. Ceci aurait cependant l'inconvénient de réduire le parallélisme des opérations. L'optimisation des séquences de déplacement et de scans STM deviendrait par le fait même un véritable casse-tête.

Mais avant même de penser à l'optimisation d'une flotte de robots, il faut s'attarder sur leurs déplacements. Le *Powerfloor* est parsemé de zones interdites. Aucune patte du NanoWalker ne doit entrer en contact avec les porte-échantillons présents un peu partout sur le *Powerfloor*. Ceux-ci constitue des obstacles fixes qu'il est possible de cartographier. À cela s'ajoute les obstacles mobiles que constituent les autres robots. La coordination des déplacements pour s'assurer qu'aucune collision n'aura lieu s'avérera un problème complexe qui nécessitera une étude poussée des principes d'optimisation de trajectoire dans un environnement dynamique[24].

Une simplification du problème est néanmoins possible, mais le coût attaché est important. On pourrait ne permettre que le déplacement d'un seul robot à la fois et ainsi considérer le reste des robots comme étant eux aussi des obstacles fixes. Mais puisqu'on sait que la communication avec un robot ne se produit que pendant un quantum de temps fonction du nombre de robot, la fenêtre de temps disponible pour déplacer les robots raccourcirait à mesure que le nombre de robots, et, par conséquent, la complexité du chemin à faire, augmenterait. Le parallélisme des opérations serait ainsi grandement affecté.

Pour permettre le déplacement de plusieurs unités simultanément, il faut avoir confiance en leur aptitude à réaliser le déplacement ordonné conformément aux paramètres prescrits. Ceci est d'autant plus vrai qu'on ne peut qu'estimer la position d'un robot mobile à partir de la dernière commande transmise et du temps depuis son dernier positionnement avec le PSD. S'il s'avérait que le robot dévie

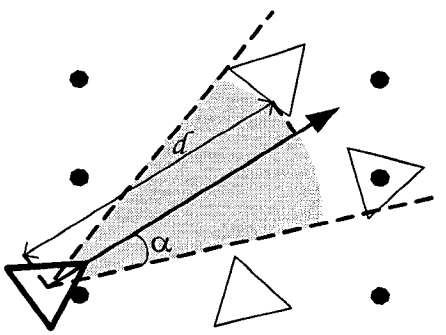


Figure 4.21 Cône d'erreur sur le déplacement d'un robot, α est l'erreur angulaire maximale, d est la distance maximale possible pour le déplacement rectiligne.

de sa trajectoire, le système ne détectera pas cette incohérence avant la prochaine séquence de positionnement. Durant tout ce temps, le risque de collision augmente. Plus le nombre de robots est grand, plus la tolérance aux erreurs diminue et les déplacements possibles avant de repositionner les robots doivent être petits.

À ce jour, le déplacement est encore erratique, un sérieux effort de développement doit être mis de l'avant pour dégager les séquences de bits optimales pour chaque forme de déplacement. Une fois cette information connue, celle-ci devra s'adaptée à chaque robot.

Chaque robot doit être calibré individuellement car les tolérances d'assemblage mécanique sur les pattes et les LEDs de positionnement sont de l'ordre du millimètre sont plus grandes que la précision nanométrique que l'on tente d'atteindre. Une série d'exercices de calibration doit donc être réalisée pour obtenir des données statistiques sur le comportement propre à chaque robot. De ces expériences, un cône représentant l'angle maximal du dispersement possible pour chaque déplacement doit être déduit. Lors de la fragmentation de la commande de déplacement en multiples déplacements unitaires rectilignes, le cône d'erreur devra être superposé

à la configuration actuelle du *Powerfloor* et la distance maximale possible pour le déplacement unitaire sera directement obtenue par l'identification du premier obstacle contenu dans le cône. La figure 4.21 présente une illustration du principe d'erreur de mouvement.

Ce ne sont pas les défis qui manquent dans le projet NanoWalker. Les risques sont grands, mais les bénéfices potentiels le sont tout autant...

RÉFÉRENCES

[1] Eric Aboussouan. Système de positionnement à l'échelle globale d'une flotte de nanorobots : Application du filtre de Kalman et d'autres techniques à l'estimation des positions retournées par un «Position Sensing Device» (PSD). Projet de fins d'études, École Polytechnique de Montréal, Avril 2004.

[2] I. Adesida, E. Kratschmer, E. D. Wolf, A. Muray, and M. Isaacson. Ion-beam lithography at nanometer dimensions. *Journal of Vacuum Science & Technology*, 3 :45–49, 1985.

[3] Stephen T. Albin. *The Art of Software Architecture : Design Methods and Techniques.* John Wiley & Sons, 2003.

[4] Ferdy Hanssen And. Real-time communication protocols : An overview, October 2003.

[5] Walder André, Jacques-A. Delafosse, and Sylvain Martel. Walking-Die : Using MEMS and SOC for a miniature robot designed for nanoscale operations. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE) 2003*, pages 1827–1830. IEEE, May 2003.

[6] Siaka Baro. DAQ Manager sans contrôles ActiveX. Projet de fins d'études, École Polytechnique de Montréal, Avril 2004.

[7] Luc Baron. Genetic algorithm for line extraction. Rapport technique, École Polytechnique de Montréal, Département de mécanique, août 1998.

[8] Guido Baumann. Development and testing of a positioning system for a miniature robot called nanowalker. Master's thesis, Universitat Karlsruhe (TH), Germany, 2003.

[9] Barry W. Boehm and PPhilip N. Papaccio. Understanding and controlling software costs. *IEEE Trans. Softw. Eng.*, 14(10) :1462–1477, 1988.

[10] Sylvain Boissé. Réalisation d'un simulateur de NanoWalker. Projet de fins d'études, École Polytechnique de Montréal, Avril 2003.

[11] Thomas FitzGerald Boitani. Design and control of a scanning tunnelling microscope for an autonomous micro-robot called "nanowalker". Master's thesis, Universitá "La Sapienza" di Roma, 2003.

[12] John F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6) :679–698, November 1986.

[13] C. Julian Chen. *Introduction to Scanning Tunneling Microscopy.* Oxford University Press, 1993.

[14] John E. Ciolfi. Fixed-Point Blockset for Use with Simulink : User's Guide Version 3, June 2001.

[15] Ondrej Cíp and Frantisek Petru. A scale-linearization method for precise laser interferometry. *Measurement Science and Technology*, 11(2) :133–141, 2000.

[16] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems : Concepts and Design.* Addison-Wesley, 2001.

[17] CRS Holdings Inc. Alloy data - carpenter super invar 32-5. http ://carpenter.idesinc.com/datasheet.asp ?i=3&e=179&c=TechArt&VIEW=PRINTER, 2004.

[18] N J Curson, R J Wilson, L A Silva, W Allison, and G A C Jones. Studying the kinetics of graphite oxidation using a scanning tunnelling microscope - an undergraduate laboratory experiment. *European Journal of Physics*, 20(6) :453–460, 1999.

[19] Scott Dattalo. Logarithms. http ://www.dattalo.com/technical/theory/logs.html, 1999.

[20] Diane Wilson and Thyra Rauch and Joeann Paige. Prototyping and the software development cycle. http ://www.firelily.com/opinions/cycle.html, 1992.

[21] E.K. Schweizer D.M. Eigler. Positioning single atoms with a scanning tunneling microscope. *Nature*, 344 :524–526, 1990.

[22] Pierre-Alain Dumas. Validation et réalisation d'une grille de référence et conception d'algorithmes de traitement de signal en vue d'obtenir un positionnement de précision nanométrique. Projet d'études supérieures, École Polytechnique de Montréal, Décembre 2003.

[23] Moufid Eyitayo. Room Manager : Système de contrôle de la chambre de refroidissement pour le projet NanoWalker. Projet de fins d'études, École Polytechnique de Montréal, Avril 2004.

[24] Thierry Fraichard. Trajectory Planning in Dynamic Workspace : a 'State-Time Space' Approach. Rapport de recherche 3545, Institut national de recherche en information et automatique, 655, avenue de l'Europe, 38330 Montbonnot St-Martin (France), Octobre 1998.

[25] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns : elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc., 1995.

[26] S. Gautsch, T. Akiyama, N.F. de Rooij, U. Staufer, Ph. Niedermann, L. Howald, D. Müller, A. Tonin, H.-R. Hidber, W.T. Pike, and M.H. Hecht. Atomic force microscope for planetary applications. In *Solid-State Sensor and Actuator Workshop*, pages 267–270, June 2000.

[27] S. Gautsch, T. Akiyama, R. Imer, N.F. de Rooij, U. Staufer, Ph. Niedermann, L. Howald, D. Brändlin, A. Tonin, H.-R. Hidber, and W.T. Pike. Measurement of quartz particles by means of an atomic force microscope for planetary exploration. *Surface and Interface Analysis*, 163(33), 2002.

[28] G.Binning and H.Rohrer. Scanning tunneling microscopy. *IBM Journal of Research and Development*, 30(4) :355–369, 1986.

[29] B Gopalakrishnan and SV Subramanyam. Many phases of carbon. *Resonance*, pages 10–19, December 2002.

[30] L Jay Guo. Recent progress in nanoimprint technology and its applications. *Journal of Physics D : Applied Physics*, 37(11) :R123–R141, 2004.

[31] Stefan Hembacher, Franz J. Giessibl, Jochen Mannhart, and Calvin F. Quate. Revealing the hidden atom in graphite by low-temperature atomic force microscopy. *PNAS*, 100(22) :12539–12542, 2003.

[32] Information Sciences Institute. Rfc793 : Transmission control protocol. WWW, September 1980.

[33] Jürgen Müller. STM Project. http ://www.e-basteln.de/index.htm.

[34] Joe Kerkes. Real-time ethernet, February 2001.

[35] Guillaume Langlois. Système de positionnement mésométrique par détection d'impulsions lumineuses pour le robot Nanowalker. Projet de fins d'études, École Polytechnique de Montréal, Avril 2003.

[36] Jiali Li, Derek Stein, Ciaran McMullan, Daniel Branton, Michael J. Aziz, and Jene Golovchenko. Ion-beam sculpting at nanometre length scales. *Nature*, 412 :166–169, 2001.

[37] Xuekun Lu, Hui Huang, Nikolay Nemchuk, and Rodney S. Ruoff. Patterning of highly oriented pyrolytic graphite by oxygen plasma etching. *Applied Physics Letters*, 75(2) :193–195, July 1999.

[38] Haritz Macicior. Contrôle de systèmes nanorobotiques. Master's thesis, École Polytechnique de Montréal, 2004.

[39] M.Aketagawa, K.Takada, Y.Minao, Y.Oka, and J.Lee. Tracking and stepping control of the tip position of a scanning tunnelling microscope by referring to atomic points and arrays on a regular crystalline surface. *Review of Scientific Instruments*, 70(4) :2053–2059, April 1999.

[40] S. Martel, J-B Mathieu, O. Felfoul, H. Macicior, G. Beaudoin, G. Soulez, and al. Adapting MRI Systems to Propel and Guide Microdevices in the Human Blood Circulatory System. In *26th Conference of the IEEE Engineering in Medicine and Biology Society.* IEEE, 2004.

[41] Sylvain Martel, Stefan Riebel, Torsten Koker, Mark Sherwood, and Ian Hunter. Large-scale nanorobotic factory automation based on the nanowalker technology. In *Proceedings of the 8th IEEE International conference on Emerging Technologies and Factory Automation*, volume 2, pages 591–597, October 2001.

[42] Sylvain Martel, Anant Saraswat, Arthur Michel, and Ian Hunter. Preliminary evaluation and experimentation of the push-slip method for achieving micrometer and sub-micrometer step sizes with a miniature piezo-actuated three-legged robot operating under high normal forces. In *Proceedings of SPIE : Microrobotics and Microassembly*, volume 4194, pages 141–148, November 2000.

[43] Sylvain Martel, Mark Sherwood, Chad Helm, William Garcia de Quevedo, Timothy Fofonoff, Robert Dyer, John Bevilacqua, Joshua Kaufman, Omar Roushdy, and Ian Hunter. Three-legged wireless miniature robots for mass-scale operations at the sub-atomic scale. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, volume 4, pages 3423–3428, 2001.

[44] Aurélien Masseboeuf. Validation et réalisation d'une grille de référence et conception d'algorithmes de traitement de signal en vue d'obtenir un positionnement de précision nanométrique. Projet de fin d'études, Institut National des Sciences Appliquées de Rennes, 2004.

[45] Shinji Matsui and Yukinori Ochiai. Focused ion beam applications to solid state devices. *Nanotechnology*, 7 :247–258, 1996.

[46] Ralph C. Merkle. It's a small, small, small, small world. *MIT Technology Review*, page 25, February/March 1997.

[47] N.Kiryati, H.Kalviainen, and S.Aloutinen. Randomized or probabilistic hough transform : unified performance evaluation. *Pattern Recognition Letters*, 21(13-14) :1157–1164, 2000.

[48] Philippe Ouimet. Design VHDL du CPLD et d'une solution de programmation dans le cadre du projet NsanoWalker. Projet de fins d'études, École Polytechnique de Montréal, Avril 2004.

[49] PageWise, Inc. A basic introduction to the analytical technique of Scanning Tunneling Microscopy. The mathematical and quantum mechanical concepts behind its inception. http ://mdmd.essortment.com/scanningtunneli_rsnc.htm, 2002.

[50] J. Postel. Rfc768 : User datagram protocol. WWW, August 1980.

[51] Roger S. Pressman. *Software Engineering : A Practitioner's Approach.* McGraw-Hill, fifth edition, 2000.

[52] S. M. Clark and D. R. Baselt and C. F. Spence and M. G. Youngquist and and J. D. Baldeschwieler. Hardware for digitally controlled scanned probe microscopes. *Review of Scientific Instruments*, 63(10) :4296–4307, October 1992.

[53] Andreas Schindler. Development of a cooling system for miniature robots (nanowalker). Master's thesis, Universitat Karlsruhe (TH), Germany, 2002.

[54] D. Schmidt and S. Vinoski. Object interconnections : Comparing alternative programming techniques for multi-threaded servers, 1996.

[55] Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects, Volume 2.* John Wiley & Sons, 2000.

[56] Douglas C. Schmidt. An architectural overview of the ace framework : A case-study of successful cross-platform systems software reuse. *USENIX Login Magazine, Tools special issue*, November 1998.

[57] H. H. Solak, C. David, J. Gobrecht, V. Golovkina, F. Cerrina, S. O. Kim, and P. F. Nealey. Sub-50 nm period patterns with euv interference lithography. *Microelectron. Eng.*, 67-68(1) :56–62, 2003.

[58] Dominic St-Jacques, Thomas Boitani, Pierre-Alain Dumas, Marc-Antoine Ducas, Marc-Antoine Fortin, and Sylvain Martel. Atomic-Scale Positioning Reference Grid System for Miniature Robots with Embedded Scanning Tunnelling Capability. In *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, pages 1339–1344. IEEE, April 2004.

[59] Dominic St-Jacques, Sylvain Martel, and Thomas Boitani. Nanoscale grid based positioning system for miniature instrumented robots. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE) 2003*, pages 1831–1834. IEEE, May 2003.

[60] Matt Stephens and Doug Rosenberg. *Extreme Programming Refactored : The Case Against XP.* Apress, 2003.

[61] Structure Probe, Inc. Highly ordered pyrolytic graphite - spi supplies. http ://www.2spi.com/catalog/new/hopgsub.shtml, 2003.

[62] Technical Manufacturing Corporation. CleanTop^{TM} II Optical Tops - 780 Series. http ://www.techmfg.com/products/opticaltops/780series.htm, 2004.

[63] Texas Instruments Inc. TMS320C2X User's Guide Rev. C, January 1993.

[64] Texas Instruments Inc. TIR2000 Data Manual : High-Speed Serial Infrared Controller With 64-Byte FIFO, June 1998.

[65] University of Leeds, School of Physics and Astronomy. Scanning Tunnelling Microscopy : Surface Topography Imaging. http ://www.stoner.leeds.ac.uk/techniques/stm.htm.

[66] Eric W. Weisstein. Mercator Series. From MathWorld–A Wolfram Web Resource. http ://mathworld.wolfram.com/MercatorSeries.html.

[67] Kwan-Po Wong and Cho-Li Wang. Push-pull messaging : A high-performance communication mechanism for commodity SMP clusters. In *International Conference on Parallel Processing*, pages 12–, 1999.

ANNEXE I

LISTE DES ÉTUDIANTS SUPERVISÉS

Nom : Constantin Fortier
Session : E2002
Projet : Projet de fin d'études (3 crédits)
Communication Infra Rouge : projet NANOWALKER
Description : Travail préliminaire en vue du développement d'un protocole de communication infrarouge avec l'optique d'obtenir la température des robots.
Supervision conjointe avec Simon McDougall

Nom : Frédérick Jubinville
Session : A2002
Projet : Projet de fin d'études (3 crédits)
Calibration du Powerfloor et de la lentille du PSD
Description : Caractérisation du PSD par l'exécution de tests de positionnement en utilisant le Protomat 95s/II de LPKF comme *XY stage*

Nom : Sylvain Boissé
Session : A2002 & H2003
Projet : Projet de fin d'études (6 crédits)
Réalisation d'un simulateur de NanoWalker
Description : Développement d'un simulateur fonctionnel des aspects de communication des robots NanoWalkers avec scénarios configurable.

Nom : Frédéric Nguyenphat-Therrien
Session : H2003
Projet : Projet de fin d'études (3 crédits)
Positionnement microscopique d'un nanorobot à l'aide de traitement et d'analyse d'images
Description : Analyse d'images STM de la grille atomique afin d'en extraire la largeur des sillons gravés. Utilisation de l'algorithme de Canny pour la détection des contours.

Nom : Serge-Olivier Chimi-Ngakeng
Session : E2003
Projet : Projet de fin d'études (6 crédits)

Description : Développement d'un pilote Linux pour la carte IR Actisys IR2000B/L

Nom : Eric L'Heureux
Session : E2003
Projet : Stage
Description : Conception et validation d'un circuit prototype pour la communication IR et implémentation avec le eZdspF2812.

Nom : Marc-Antoine Ducas
Session : E2003 & A2003 & H2004 & E2004
Projet : Projet de fin d'études (6 crédits) & stages
Système intermédiaire de positionnement du NanoWalker par algorithme itératif et grille d'approche
Description : Développement et perfectionnement de l'algorithme IAPA, participation à l'élaboration de l'architecture logicielle, aide à la supervision et la structure des travaux logiciels, etc.
Supervision conjointe avec Marc-Antoine Ducas

Nom : Pierre-Alain Dumas
Session : A2003
Projet : Projet d'études supérieures
Validation et réalisation d'une grille de référence et conception d'algorithmes de traitement de signal en vue d'obtenir un positionnement de précision nanométrique
Description :

Nom : Aurélien Masseboeuf
Session : H2004
Projet : Stage de fin de diplôme français
Projet Nanowalker : Étude du système de positionnement du robot à l'échelle micrométrique et atomique
Description : Amélioration des procédés de fabrication de la grille atomique au FIB. Tests sur du Silicium 7×7
Supervision conjointe avec Marc-Antoine Ducas et Pierre-Alain Dumas

Nom : Philippe Ouimet
Session : H2003 & H2004
Projet : Projet de fin d'études (6 crédits)
Design VHDL du CPLD et d'une solution de programmation dans le cadre du projet NanoWalker
Description : Étude et implémentation des responsabilité du CPLD du NanoWalker
Supervision conjointe avec Marc-Antoine Fortin

Nom : Moufid Eyitayo
Session : H2004
Projet : Projet de fin d'études (3 crédits)
RoomManager - Système de contrôle de la chambre de refroidissement pour le projet NanoWalker
Description : Développement du module d'interface logicielle avec le contrôleur PLC de la chambre de refroidissement via un lien RS232.
Supervision conjointe avec Haritz Macicior

Nom : Siaka Baro
Session : H2004
Projet : Projet de fin d'études (3 crédits)
DAQ Manager sans contrôles ActiveX
Description : Interface d'acquisition des données des modules SCXI National Instruments dissociée des interfaces graphiques ActiveX dans le but d'augmenter la modularité.
Supervision conjointe avec Marc-Antoine Ducas

Nom : José Pascual
Session : H2004
Projet : Projet de fin d'études (3 crédits)
Traduction de commandes en instructions de déplacement pour exécution par un NanoWalker
Description : À partir d'une instruction de déplacement, traduire en message compris par le NanoWalker selon le format de message prédéfinit.
Supervision conjointe avec Marc-Antoine Ducas

Nom : David Salamanca
Session : E2004
Projet : Stage
Description : Programmation du circuit prototype MacroWalker
Supervision conjointe avec Marc-Antoine Fortin

Nom : Marc Léger
Session : E2004
Projet : Stage de fin de diplôme français
Description : Programmation du circuit prototype MacroWalker
Supervision conjointe avec Marc-Antoine Fortin

Nom : Wael Sabra
Session : E2004
Projet : Projet de fin d'études
Description : Étude des algorithmes de déplacement pour un robot dans un environnement dynamique.

ANNEXE II

LISTE DES PUBLICATIONS

1. Présentation lors de la Conférence canadienne de Génie Électrique et Informatique (CCGEI 2003)

   St.-Jacques D., Martel S., and Boitani T. "Nanoscale grid based positioning system for miniature instrumented robots," Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE) 2003, Montréal, Canada, May 4-7, 2003

2. Présentation lors de la conférence *IEEE International Conference on Robotics and Automation* (ICRA2004)

   St-Jacques D., Boitani T., Dumas P-A., Ducas M-A., Fortin M-A., and Martel S., "Atomic-Scale Positioning Reference Grid System for Miniature Robots with Embedded Scanning Tunnelling Capability," Proceedings of the 2004 IEEE International Conference on Robotics & Automation, New Orleans, LA, pp. 1339-1344, April 2004

ANNEXE III

DESCRIPTION DES ÉLÉMENTS DE L'ARCHITECTURE

À partir des critères de design établis dans les sections 2.2 et 2.3, il est possible d'identifier certains des éléments constituant le noyau de l'architecture. Les éléments perçus comme essentiels pour chacun des niveaux logiciels sont détaillés dans la présente section.

III.1 Drivers

IRDriver	
Niveau :	Driver
OS :	Linux
Matériel :	Carte ISA ACTISYS IR2000B/L
Responsabilités :	– Fournir des fonctionnalités d'envoi et de réception de paquets infrarouge selon le standard FIR 1.1 – Aucun traitement autre que la validation de la réception ne doit être effectué sur les paquets reçus

PSDDriver	
Niveau :	Driver
OS :	Windows
Matériel :	National Instruments NI-4472 On-Trac Photonics PSM2-20
Responsabilités :	– Lire la tension émise par le PSD après un stade d'amplification analogique la menant entre $\pm 10V$ – Aucun traitement sur la tension obtenue n'est effectué

DAQDriver	
Niveau :	Driver
OS :	Windows
Matériel :	NI : SCXI-1000, SCXI-1112, SCXI-1530 Accéléromètres : Endevco 752A13, 7724 Thermocouples : Omega Type-T
Utilise :	Librairie NI-DAQ
Responsabilités :	– Recueillir les tensions générées par les différents senseurs apposés sur la chambre de refroidissement. Ces tensions sont acheminées vers un même module d'acquisition de NI. – Aucun traitement sur les tensions recueillies ne doit être effectué.

RS232Driver	
Niveau :	Driver
OS :	Windows
Matériel :	Chambre de refroidissement de Cryotronix
Utilise :	Fonctions de communication RS232
Responsabilités :	– Fournir les fonctionnalités d'envoi et de réception de messages selon le protocole RS232. – Aucun traitement sur les communications de la chambre ne doit être effectué.

GPIBDriver	
Niveau :	Driver
OS :	Windows
Matériel :	Bloc d'alimentation : Sorensen DCS12-250E NI PCI-GPIB+
Utilise :	Fonctions de communication GPIB
Responsabilités :	– Fournir les fonctionnalités d'envoi et de réception de message selon le protocole GPIB – Aucun traitement sur les communications du bloc d'alimentation ne doit être effectué.

III.2 Managers

CommManager	
Niveau :	Manager
OS :	Linux
Utilise :	IRDriver
Responsabilités :	– Fournir une interface indépendante du medium de communication utilisé pour échanger des paquets de données avec les NWs. – Gérer la validation des transmissions, la retransmission des paquets perdus et la réception des réponses des NWs.

PosManager	
Niveau :	Manager
OS :	Linux
Utilise :	PSDDriver
Responsabilités :	– Fournir une interface indépendante de la technologie pour l'acquisition de la position globale des NWs – Convertir les données obtenues du `driver` en données de positionnement pour un NW nominal. Le PosManager connaît les caractéristiques générales d'un NW lui permettant de transformer adéquatement les données du driver en données de positionnement. (*e.g.* Le NW possède deux LEDs IR théoriquement distante de $21,082mm$)

RoomManager	
Niveau :	Manager
OS :	Windows
Utilise :	RS232Driver
Responsabilités :	– Fournir une interface indépendante du matériel pour contrôler la chambre de refroidissement. – Convertir en messages compréhensibles par le contrôleur de la chambre de refroidissement les ordres reçus des niveaux supérieurs y étant adressé. – Convertir les messages reçus du contrôleur de la chambre de refroidissement en information sur son état.

DAQManager	
Niveau :	Manager
OS :	Windows
Utilise :	DAQDriver
Responsabilités :	- Convertir les tensions lues par le `DAQDriver` en information de température, de vibration, etc. Le `DAQManager` connaît la configuration du module d'acquisition NI afin de savoir la correspondance des tensions avec les paramètres physiques mesurés. - Fournir une interface simple pour transmettre l'information lue aux niveaux supérieurs.

PwrManager	
Niveau :	Manager
OS :	Windows
Utilise :	GPIBDriver
Responsabilités :	- Convertir en messages compréhensibles par le bloc d'alimentation les ordres reçus des niveaux supérieurs y étant adressé. - Convertir les messages reçus du bloc d'alimentation en information sur son état.

III.3 Agents

RobotAgent	
Niveau :	Agent
OS :	Linux
Utilise :	CommManager PosManager
Responsabilités :	– Fournir une interface pour le contrôle des NWs sous la forme d'une liste de commandes acceptées. Certaines commandes complexes pourront se traduire en plusieurs instructions individuelles pour les NWs. – Fournir une interface pour obtenir de l'information sur l'état des NWs (Température des différents senseurs d'un NW, données STM, position globale et atomique, ordres courants, etc.) – Maintenir un instantané (*snapshot*) de l'état actuel des NWs physiques. – Maintenir la liste des commandes à effectuer et en cours de traitement pour chacun des NWs. – Maintenir l'information de calibration relative à chaque NW. – Effectuer le pré-traitement des instructions à envoyer aux NW afin de tenir compte des caractéristiques physiques réelles propres à chaque NW. – Effectuer le post-traitement sur les données de positionnement obtenues du PosManager en fonction des données de calibration propres à chaque NW. – Calculer les trajectoires pour les déplacements des NWs afin d'éviter les collisions avec les autres NWs et les zones de travails du PowerFloor. – Convertir les instructions à envoyer aux NWs en messages compréhensibles par ceux-ci.

EnvAgent	
Niveau :	Agent
OS :	Windows
Utilise :	RoomManager DAQManager PwrManager
Responsabilités :	– Fournir une interface pour le contrôle de l'environnement des NWs (*i.e.* la chambre de refroidissement) – Fournir une interface pour obtenir de l'information sur l'état de l'environnement. – Maintenir l'état actuel de l'environnement. – Connaître la configuration physique des différents senseurs. – Régulariser la température de l'environnement.

III.4 NanOS

NWMonitor	
Niveau :	NanOS
OS :	Linux
Utilise :	RobotAgent EnvAgent
Responsabilités :	– Corréler l'état des NWs et les données provenant des deux agents d'assurer l'intégrité des systèmes de la plate-forme. – Propager l'information relatives aux erreurs à tous les éléments du système.

ClientAppInterface	
Niveau :	NanOS
OS :	Linux
Utilise :	RobotAgent EnvAgent
Responsabilités :	– Faire le pont entre les agents contrôlant les divers systèmes et les requêtes des applications clients. – Assurer l'intégrité et la sécurité du lien entre l'application client et la plate-forme. – Maintenir une liste des applications utilisant la plate-forme.

ANNEXE IV

DIAGRAMMES DE CLASSES

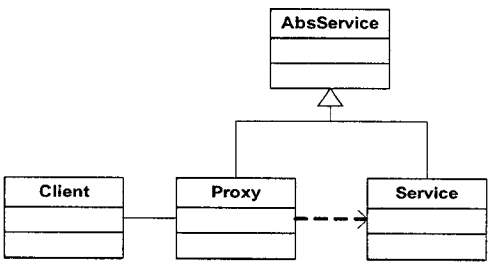


Figure IV.1 Structure du patron de conception *Proxy*

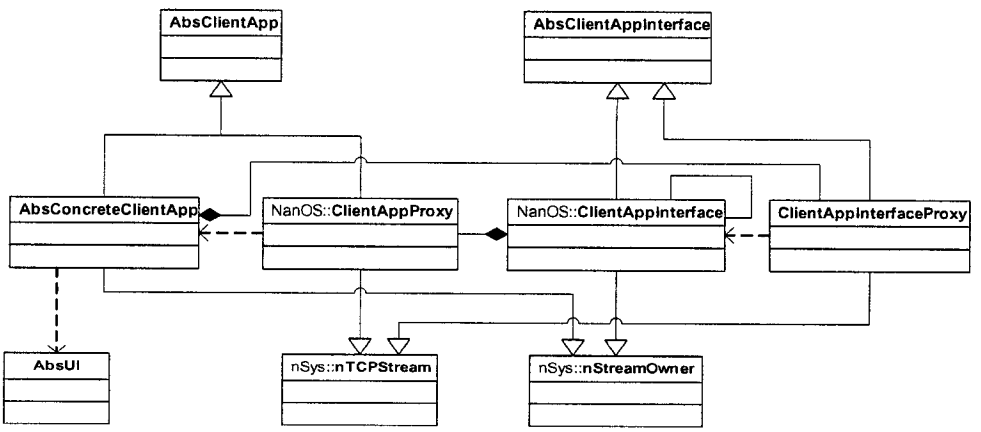


Figure IV.2 Classes de bases abstraites et communes aux autres paquetages

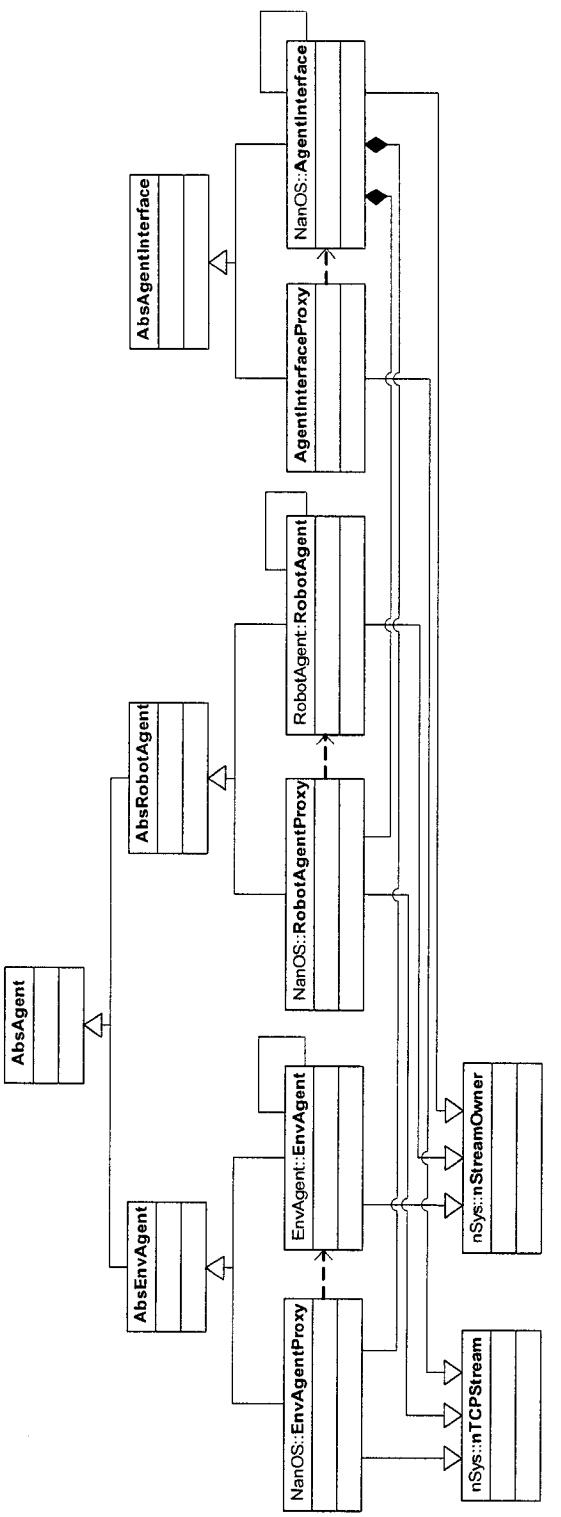


Figure IV.3 Classes de bases abstraites et communes aux autres paquetages

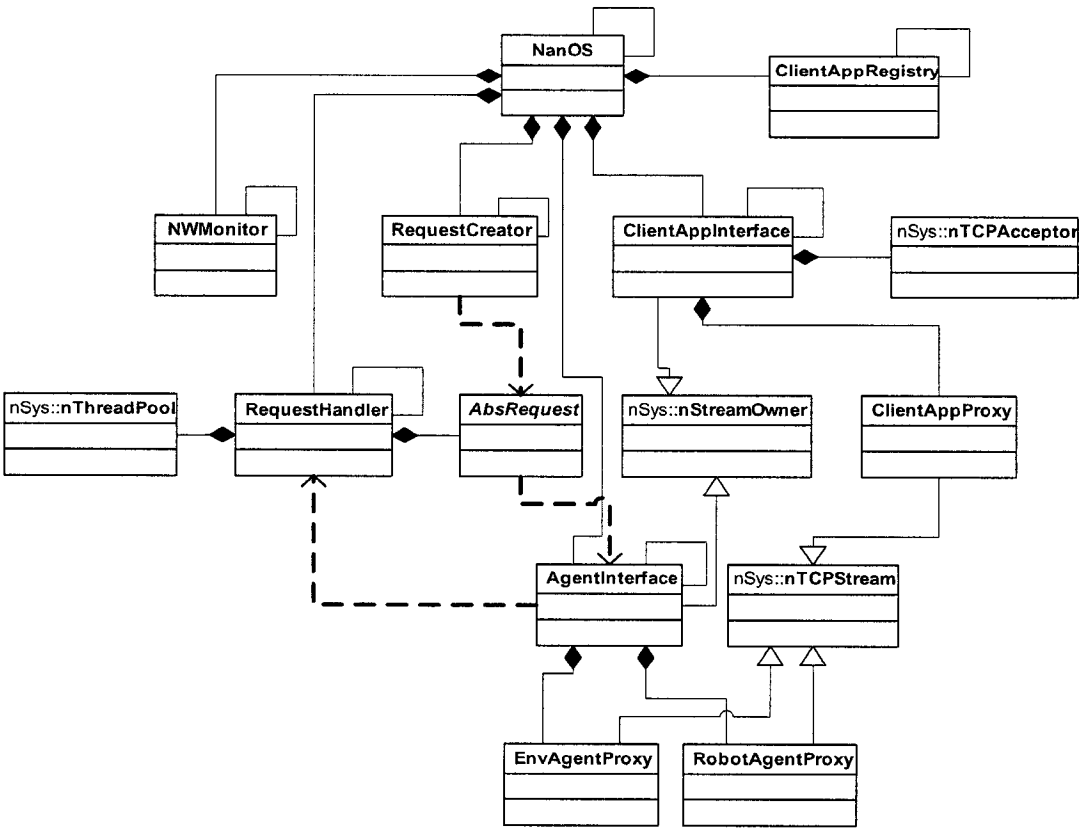


Figure IV.4 Classes de la couche `NanOS`

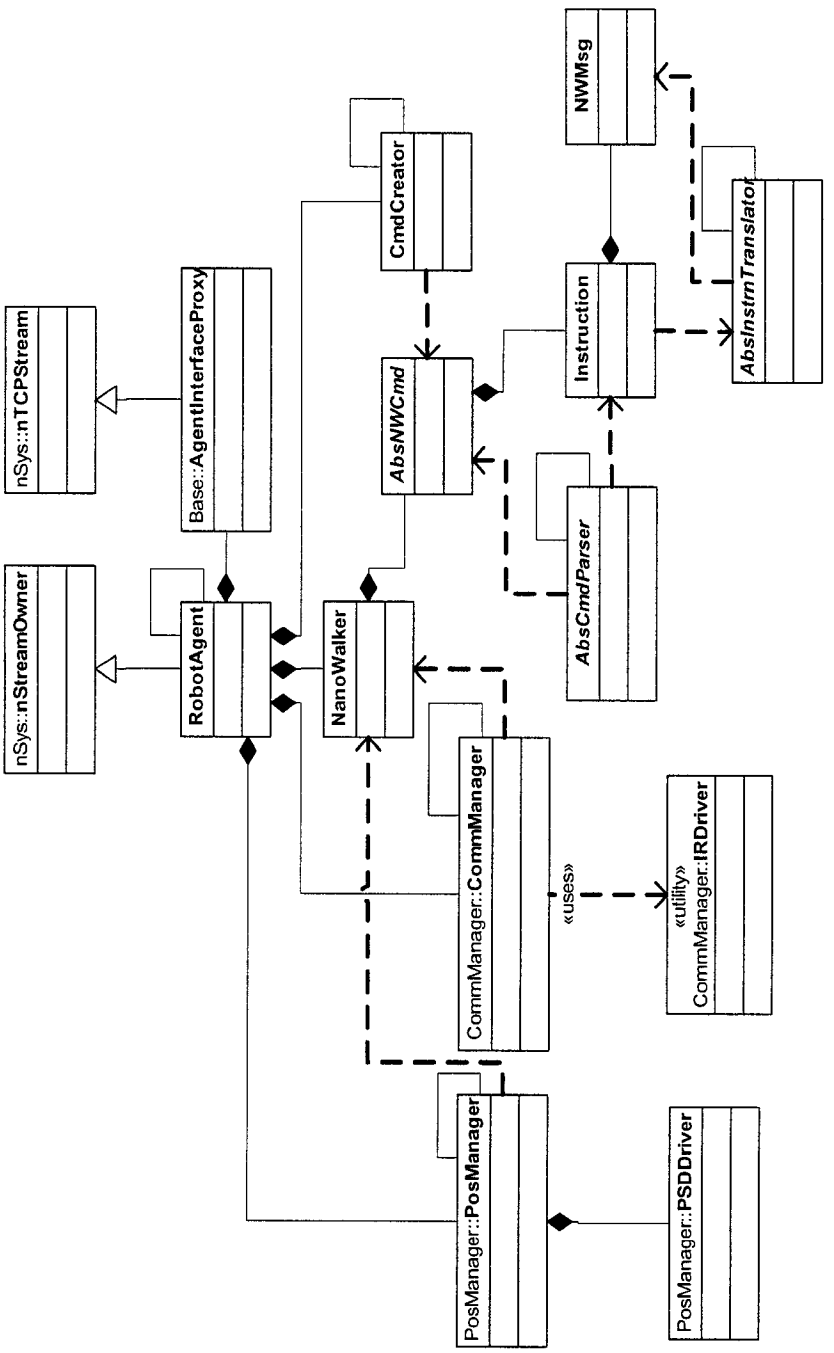


Figure IV.5 Diagramme de classes du RobotAgent

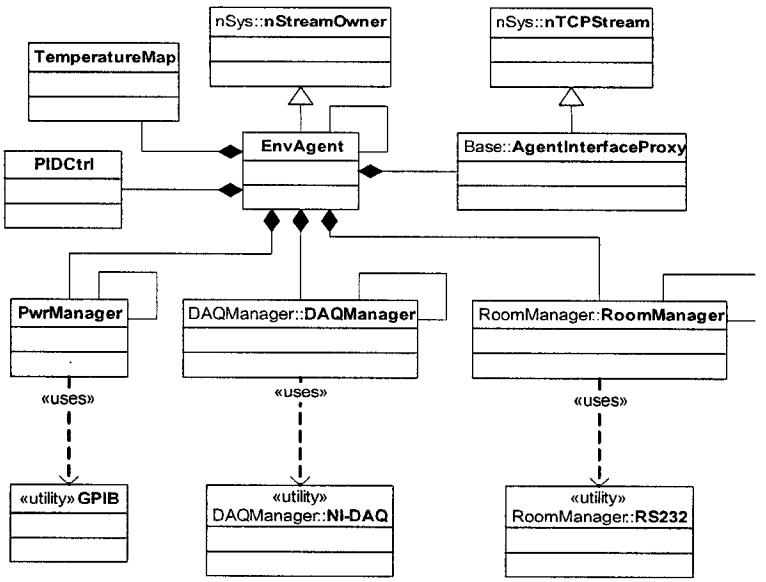


Figure IV.6 Diagramme de classes du `EnvAgent`

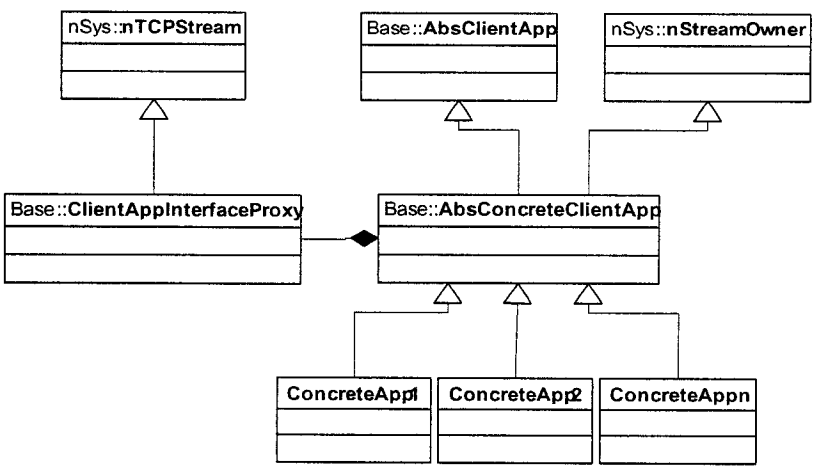


Figure IV.7 Classes de la couche `application`

ANNEXE V

DIAGRAMMES DE SÉQUENCES

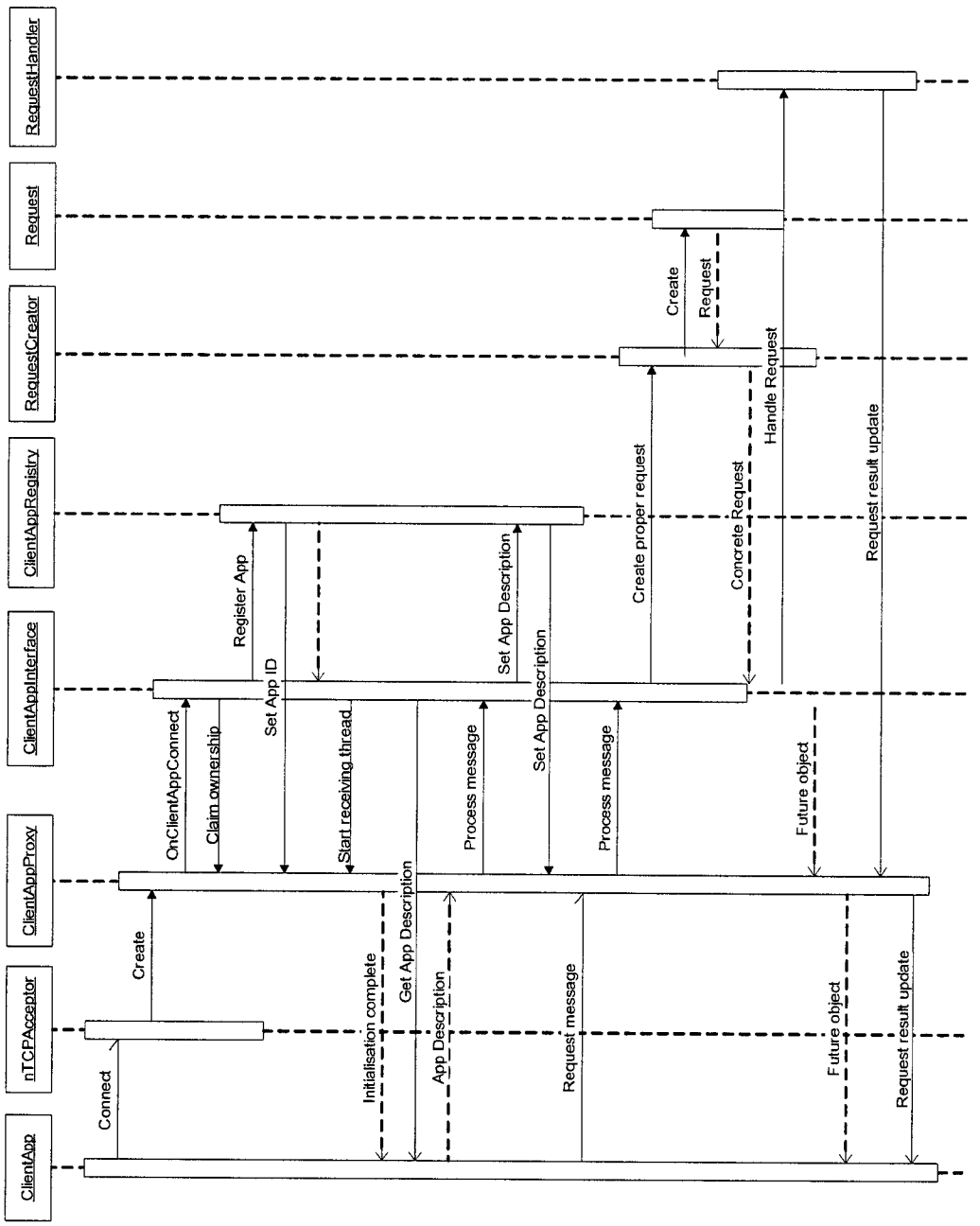


Figure V.1 Séquence d'initialisation d'une application client

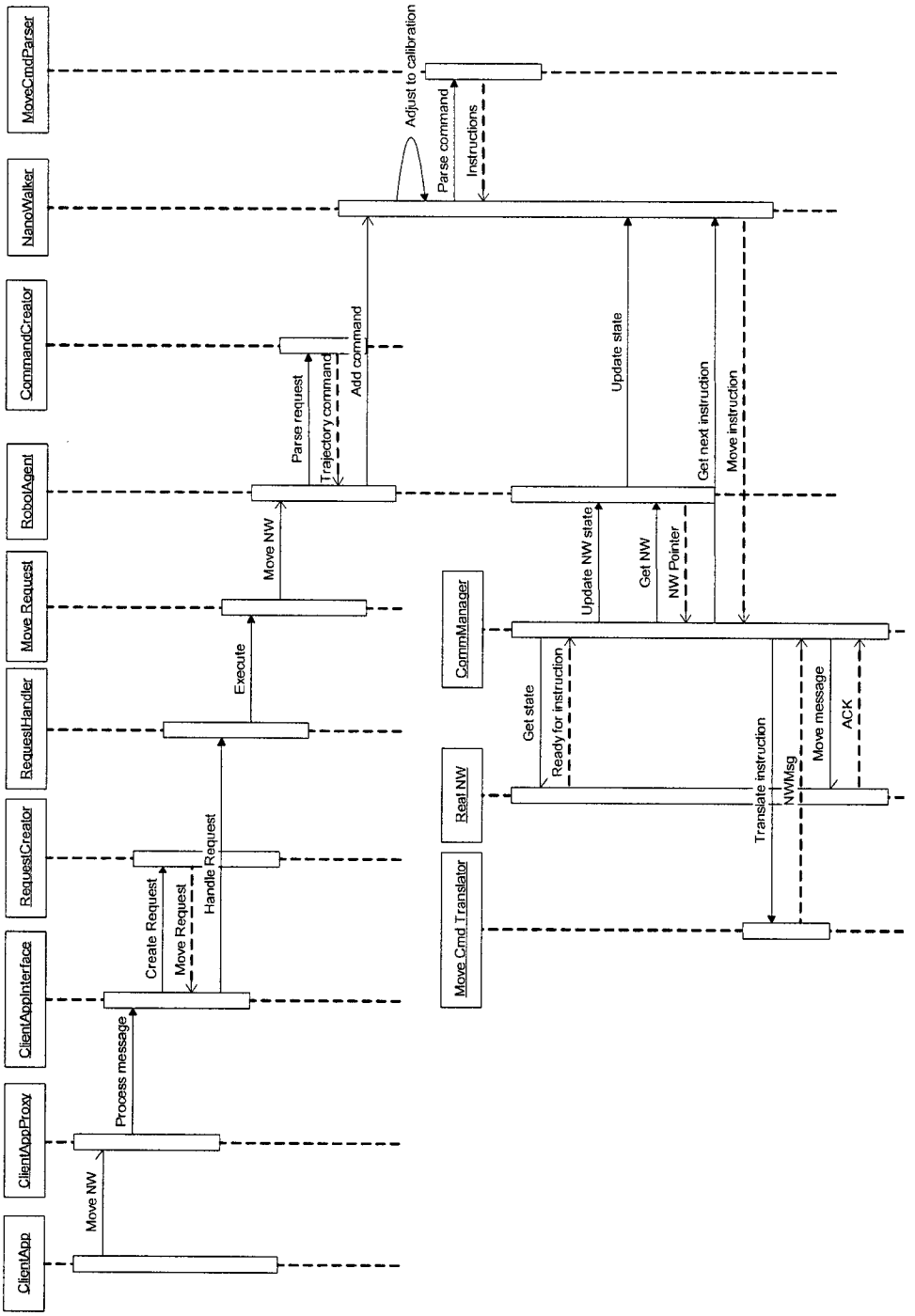


Figure V.2 Diagramme de séquence d'une requête de déplacement de robot

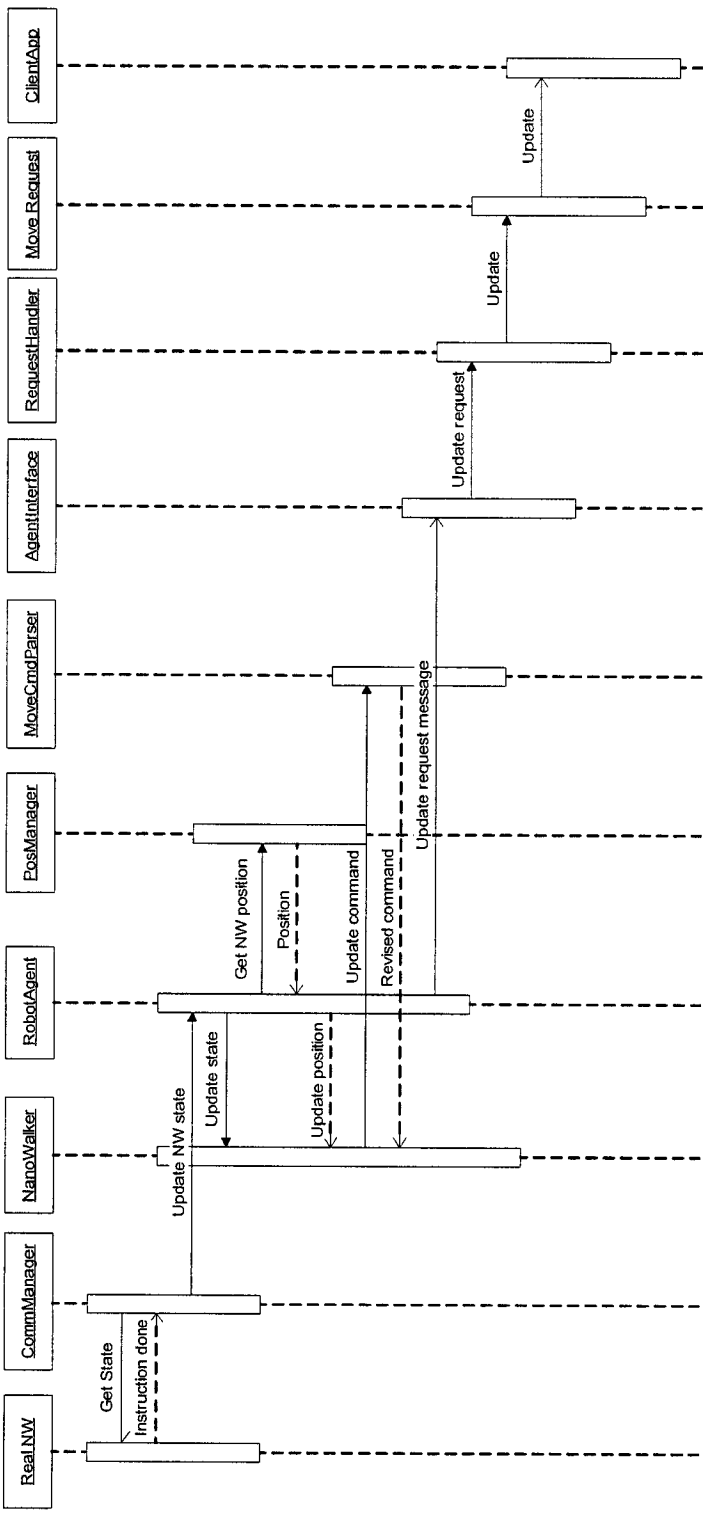


Figure V.3 Diagramme de séquence d'une requête de déplacement de robot (mise à jour)

ANNEXE VI

OBSERVATIONS À L'ANALYSEUR LOGIQUE

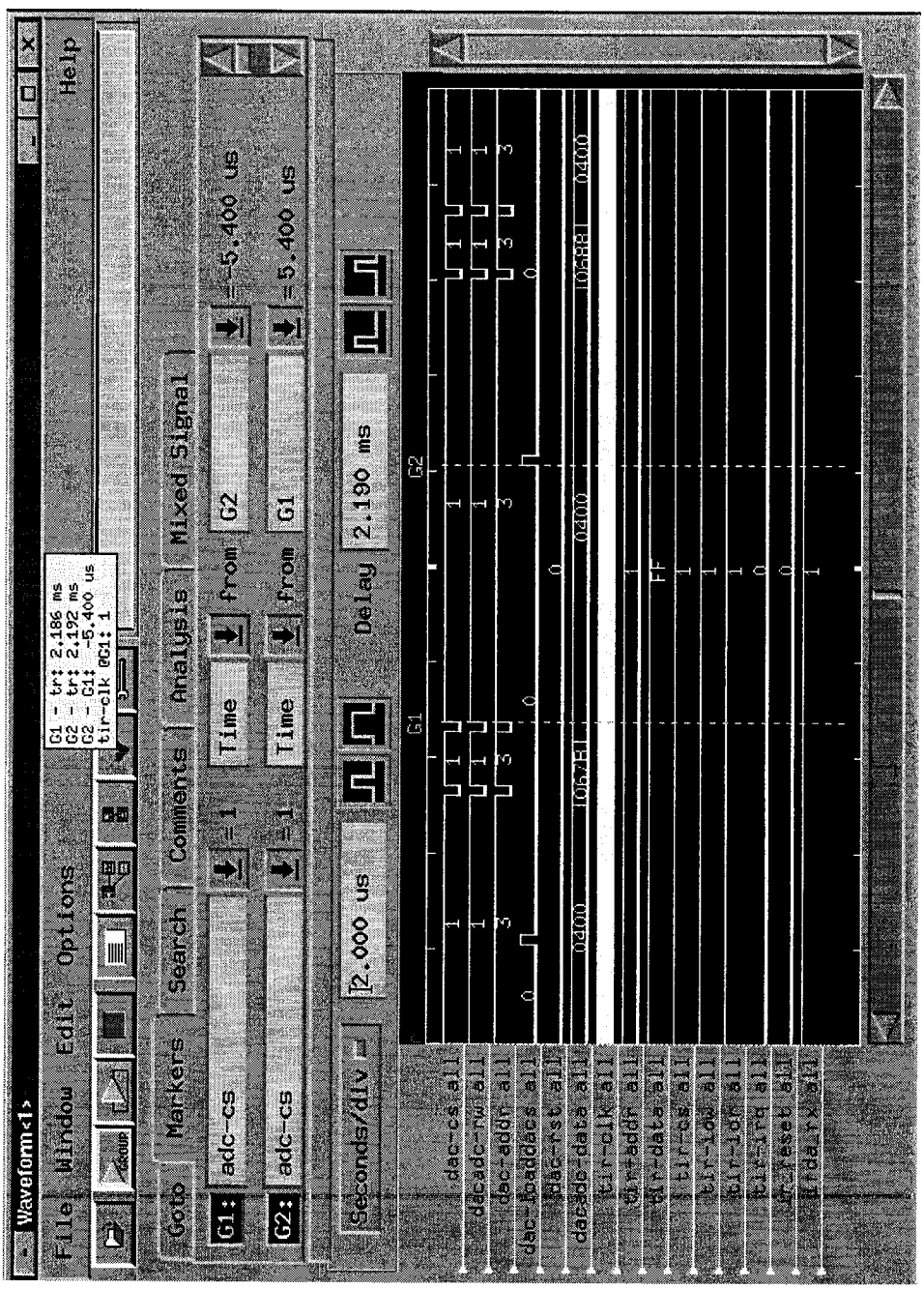


Figure VI.1 Aperçu global des signaux de contrôle du DAC par le eZdspF2812

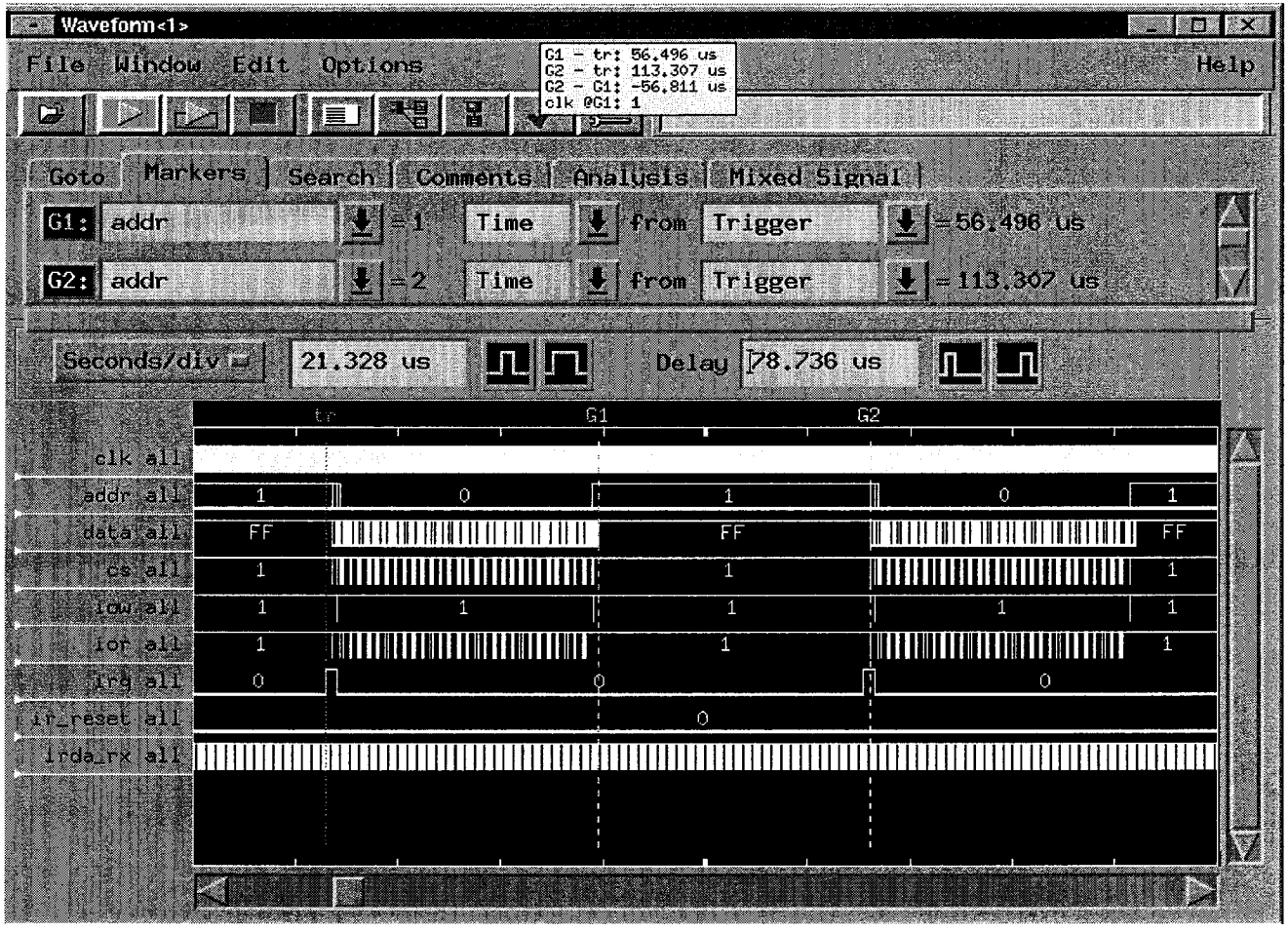


Figure VI.2 Aperçu global des signaux de réception de données IR avec un seuil de 56 octets

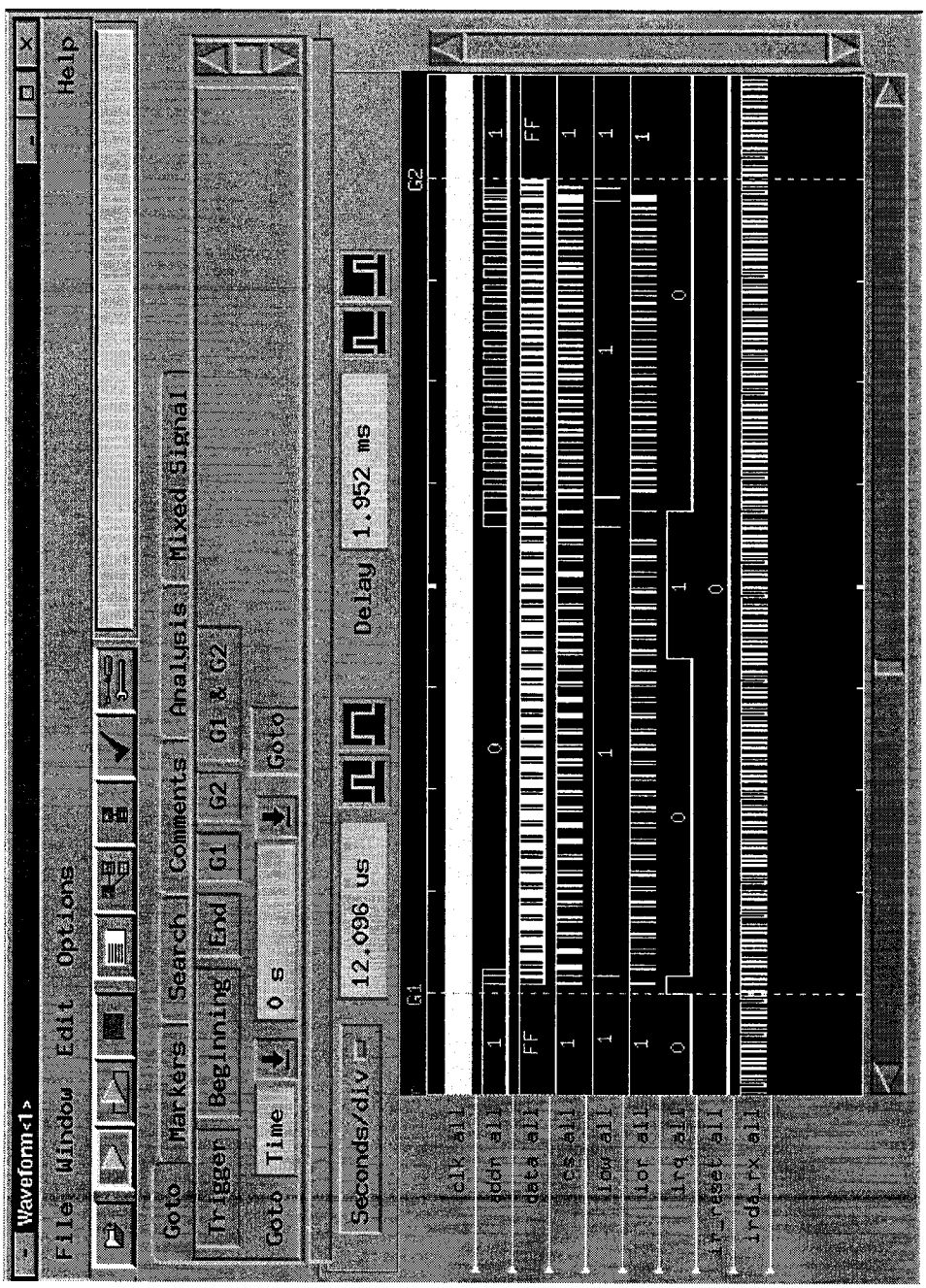


Figure VI.3 Zoom sur la fin de réception d'une trame IR

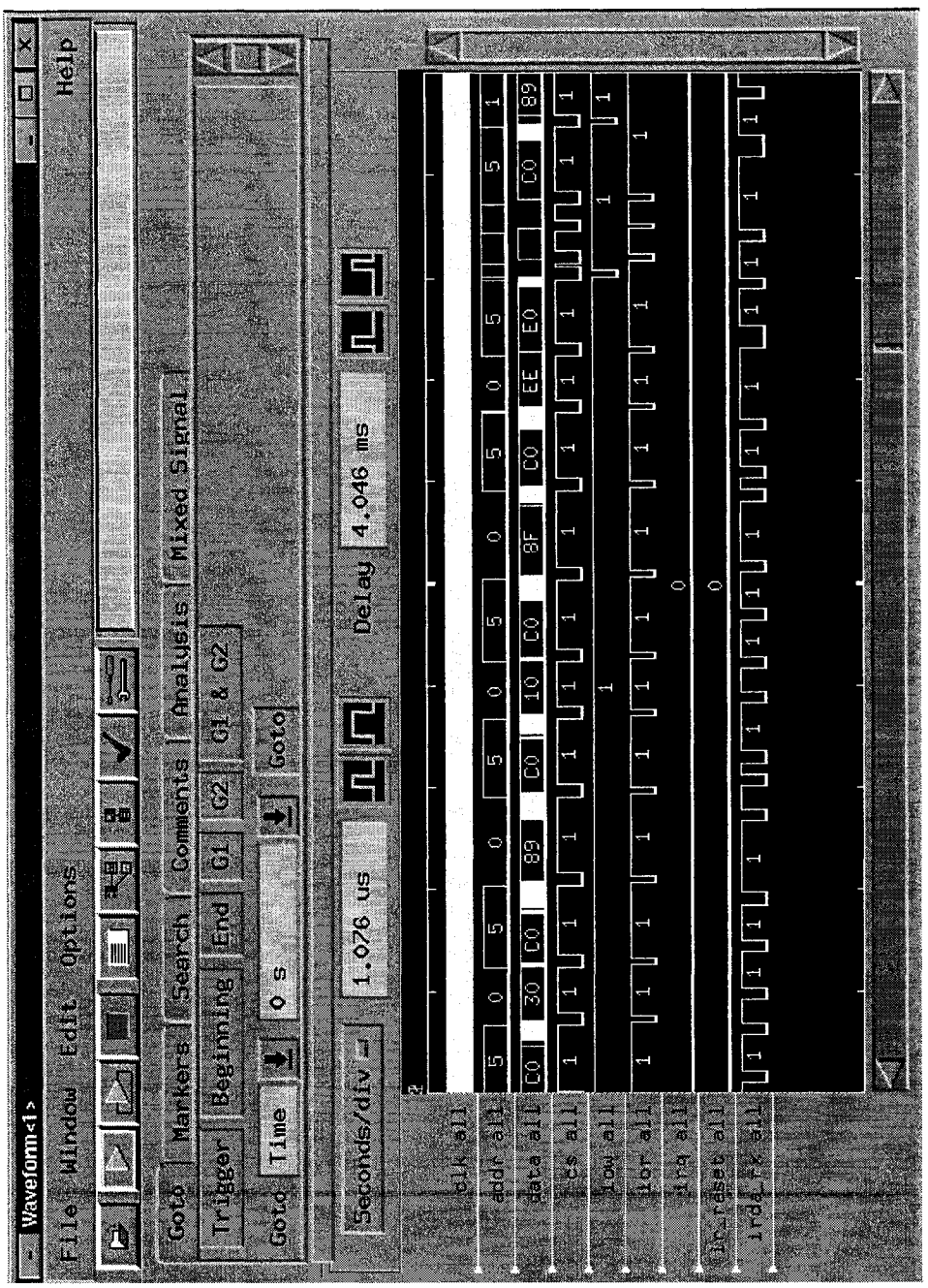


Figure VI.4 Zoom sur l'arrivée du bit **EOF** et sur le CRC

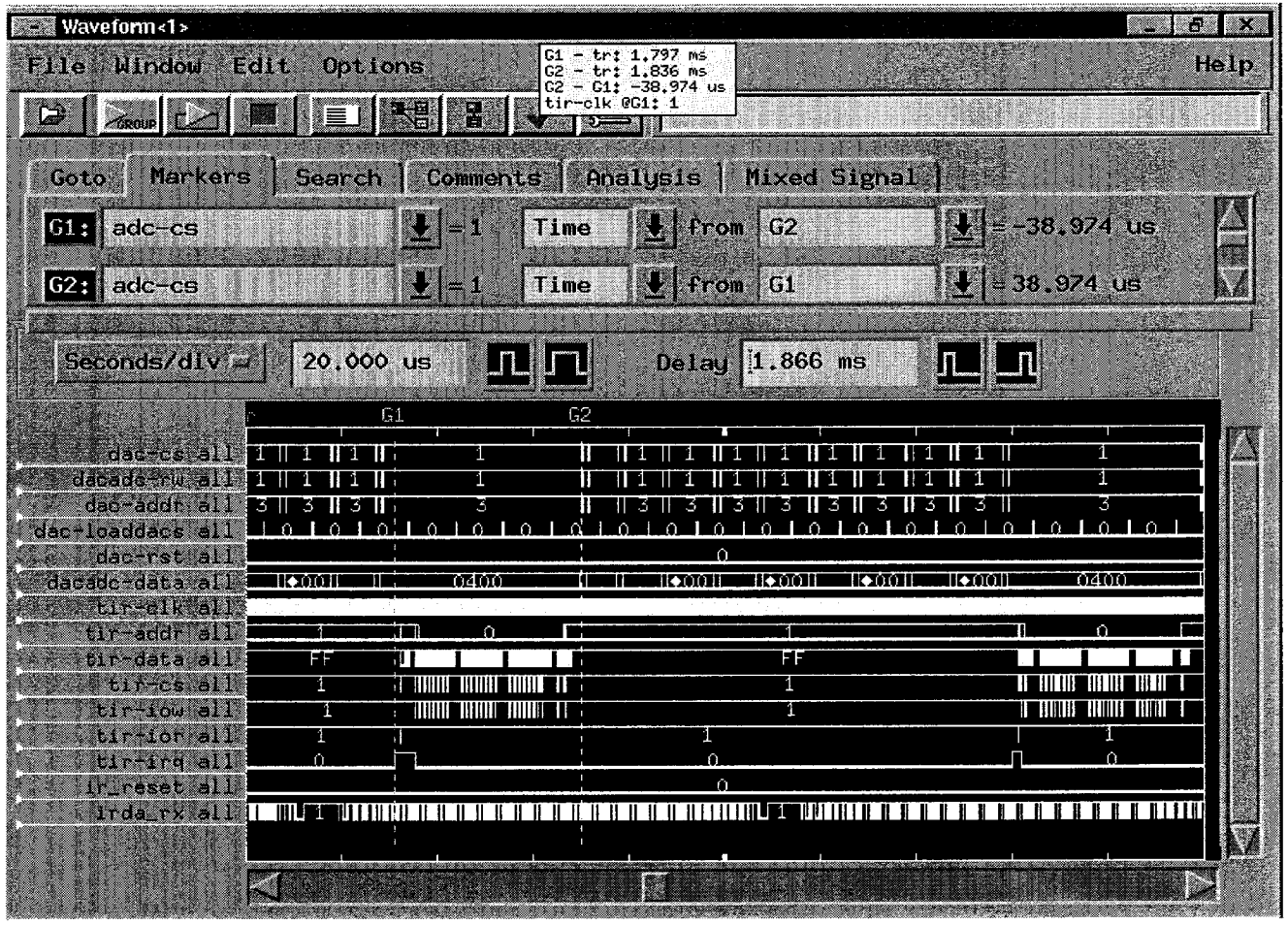


Figure VI.5 Effet de la transmission de données IR avec un seuil de 48 octets sur le PiezoScan

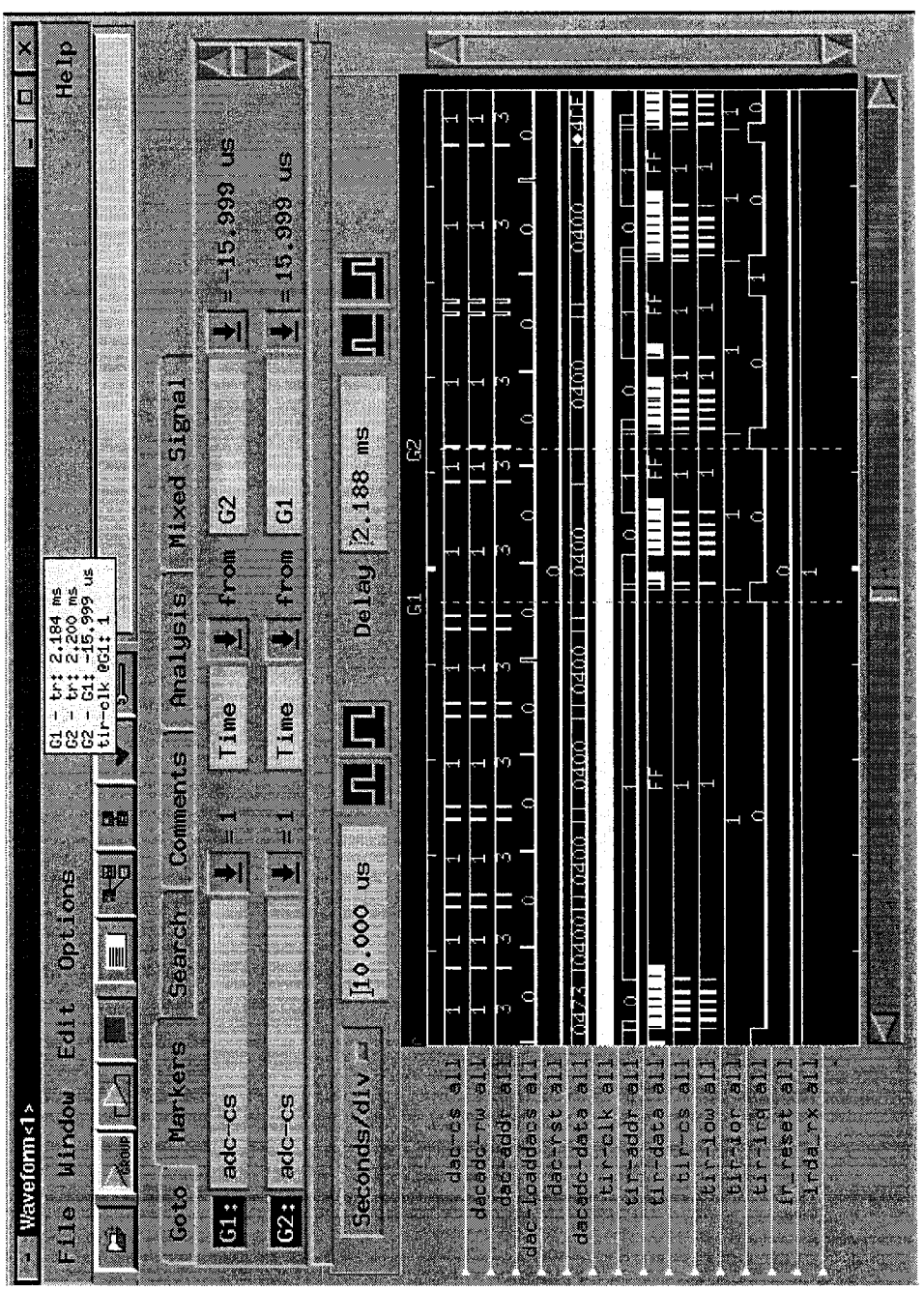


Figure VI.6 Effet de la transmission de données IR avec un seuil de 8 octets sur le PiezoScan